Linear Algebra in NumPy

17 NumPy 线性代数

NumPy 中的重要线性代数计算

我的大脑只是一个接收器。宇宙中有一个核心，我们从中获得知识、力量和灵感。这个核心的秘密我没有深入了解，但我知道它的存在。

My brain is only a receiver, in the Universe there is a core from which we obtain knowledge, strength and inspiration. I have not penetrated into the secrets of this core, but I know that it exists.

—— 尼古拉·特斯拉 (Nikola Tesla) | 发明家、物理学家 | 1856 ~ 1943

- `numpy.linalg.cholesky()` 计算 Cholesky 分解
- `numpy.linalg.dot()` 计算向量的点积
- `numpy.linalg.eig()` 计算矩阵的特征值和特征向量
- `numpy.linalg.inv()` 计算矩阵的逆
- `numpy.linalg.lstsq()` 求最小二乘解
- `numpy.linalg.norm()` 计算向量的范数
- `numpy.linalg.pinv()` 计算矩阵的 Moore-Penrose 伪逆
- `numpy.linalg.solve()` 求解线性方程组
- `numpy.linalg.svd()` 计算奇异值分解

17.1 NumPy 的 linalg 模块

NumPy 库的 **linalg** 模块提供了许多用于线性代数计算的函数，包括矩阵分解和向量计算。

以下是一些常见的 **linalg** 函数：

- **numpy.linalg.inv**()：计算矩阵的逆。
- **numpy.linalg.pinv**()：计算矩阵的 Moore-Penrose 伪逆。
- **numpy.linalg.solve**()：求解线性方程组 $\boldsymbol{Ax} = \boldsymbol{b}$，其中 $\boldsymbol{A}$ 是一个矩阵，$\boldsymbol{b}$ 是一个向量。
- **numpy.linalg.lstsq**()：最小二乘解。

linalg 模块还提供了许多向量计算函数，包括：

- **numpy.linalg.norm**()：计算向量的范数。
- **numpy.linalg.dot**()：计算向量的点积。

以下是 **linalg** 中常用的矩阵分解函数：

- **numpy.linalg.cholesky**()：计算 Cholesky 分解。
- **numpy.linalg.eig**()：计算矩阵的特征值和特征向量。
- **numpy.linalg.svd**()：计算奇异值分解。

这些函数在许多科学计算中都非常有用，例如，在机器学习中，可以使用矩阵分解函数进行降维和特征提取，而向量计算函数则可用于计算距离和相似性度量等。需要注意的是，这些函数都要求输入参数为 NumPy 数组，并返回 NumPy 数组作为输出。

什么是矩阵分解？

矩阵分解是一种将一个矩阵分解为若干个矩阵的乘积的数学技术。这种分解可以帮助我们更好地理解和处理矩阵数据。常见的矩阵分解包括 Cholesky 分解、特征值分解 (EVD)、奇异值分解 (SVD) 等等。矩阵分解在很多领域都有广泛的应用，比如在机器学习、数据分析、信号处理、图像处理等方面。

本节配套的 Jupyter Notebook 文件是 Bk1_Ch17_01.ipynb。

17.2 拆解矩阵

一组行向量

本书前文提到鸢尾花数据矩阵 $\boldsymbol{X}$ 的形状为 150 × 4。也就是说，如图 1 热图所示，$\boldsymbol{X}$ 可以看成是由 150 个行向量上下堆叠而成。每个行向量的形状为 1 × 4。图 1 特别展示了 $\boldsymbol{x}^{(1)}$ (X 第 1 行，数组第 0 行)、$\boldsymbol{x}^{(2)}$ ($\boldsymbol{X}$ 第 2 行，数组第 1 行)、$\boldsymbol{x}^{(51)}$ ($\boldsymbol{X}$ 第 51 行，数组第 50 行)、$\boldsymbol{x}^{(101)}$ ($\boldsymbol{X}$ 第 101 行，数组第 100 行)。

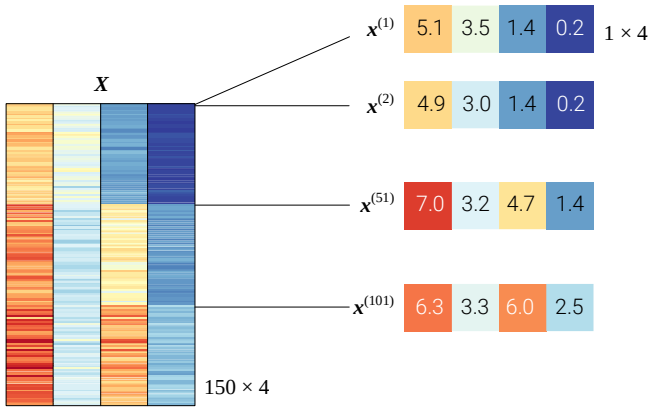


图 1. $\boldsymbol{X}$ 可以看做由一组行向量构成

如图 2 代码所示，ⓐ从 sklearn.datasets 模块导入 load_iris 函数，而ⓑload_iris() 可以用来加载鸢尾花数据集。ⓒ从 iris 对象中提取数据集的特征数据，并将其存储在一个名为 X 的变量中。数据格式为二维 NumPy 数组，相当于一个矩阵。ⓓ提取矩阵第一行 (行向量)，即 NumPy 数组第 0 行。本书前文反复提过，采用ⓓ这种双层中括号切片的结果还是一个二维 NumPy 数组。

为了节省篇幅，本章代码不展示可视化环节。可视化对应的代码，请大家参考本书配套代码文件。

```
# 导入包
import numpy as np
from sklearn.datasets import load_iris

# 从sklearn导入鸢尾花数据
iris = load_iris()
X = iris.data

# 提取四个行向量 (二维数组)
x_row_1   = X[[1 - 1], :]
x_row_2   = X[[2 - 1], :]
x_row_51  = X[[51 - 1], :]
x_row_101 = X[[101 - 1], :]
```

图 2. 提取行向量；Bk1_Ch17_01.ipynb

一组列向量

此外，如图 1 热图所示，$\boldsymbol{X}$ 可以看成是由 4 个列向量左右排列而成，即 $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4]$。每个列向量的形状为 150 × 1。请大家回忆，我们如何设定索引获得 NumPy 数组的行向量、列向量。

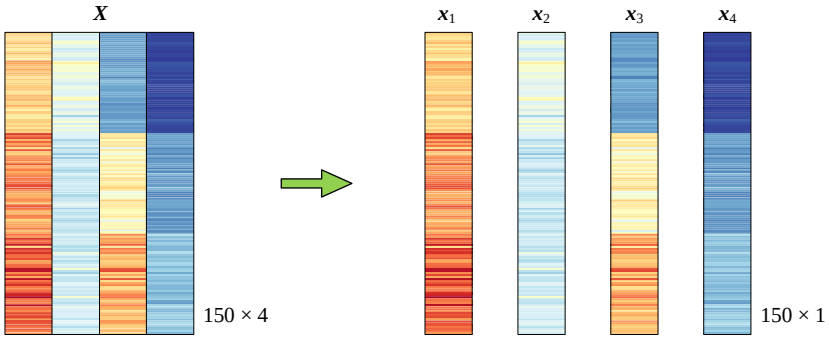


图 3. $\boldsymbol{X}$ 可以看做由一组列向量构成

```
# 提取四个列向量（二维数组）
x_col_1 = X[:, [0]]
x_col_2 = X[:, [1]]
x_col_3 = X[:, [2]]
x_col_4 = X[:, [3]]
```

图 4. 提取列向量，使用时配合前文代码；Bk1_Ch17_01.ipynb

17.3 向量运算

几何角度看向量

在二维空间中，一个向量 $\boldsymbol{a}$ 可以表示为一个有序的数对 (a_1, a_2)、$[a_1, a_2]$、$[a_1, a_2]^T$。向量也可以用一个有向线段来表示，线段的起点为原点 (0, 0)，终点为 (a_1, a_2)。其中，a_1 表示向量在水平方向上的投影；y 表示向量纵轴方向上的投影。

用勾股定理，我们可求得图 5 中向量 $\boldsymbol{a}$ 的长度，即向量的模，为 $\|\boldsymbol{a}\| = \sqrt{a_1^2 + a_2^2}$。在 NumPy 中计算向量模的函数为 **numpy.linalg.norm**()。

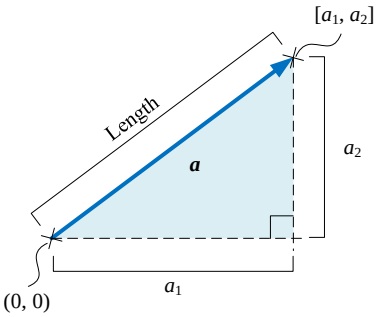


图 5. 向量起点、终点、大小和方向

在 Bk1_Ch17_01.ipynb 中，我们还计算了 $\boldsymbol{x}^{(1)}$、$\boldsymbol{x}^{(2)}$、$\boldsymbol{x}^{(51)}$、$\boldsymbol{x}^{(101)}$这四个向量的单位向量。单位向量是长度为 1 的向量，可以用来表示某个向量方向。比如，的单位向量就是 $\boldsymbol{x}^{(1)}$除以自己的模 $\left\|\boldsymbol{x}^{(1)}\right\|$，即 $\boldsymbol{x}^{(1)} \Big/ \left\|\boldsymbol{x}^{(1)}\right\|$。

图 6 中 a 利用 numpy.linalg.norm() 计算向量或矩阵范数。简单来说，范数用来衡量向量或矩阵在空间中的大小或长度。在机器学习和数学建模中，范数经常用于正则化、距离计算和优化问题等方面。如果输入为一维数组，numpy.linalg.norm() 默认计算 L2 范数，即 Euclidean 范数。

b 计算非零向量的单位向量。

《矩阵力量》第 3 章将专门介绍向量范数，《矩阵力量》第 18 章将介绍矩阵范数。

```
# 向量模（L2范数）

a norm_x_row_1 = np.linalg.norm(x_row_1)
  norm_x_row_2 = np.linalg.norm(x_row_2)
  norm_x_row_51 = np.linalg.norm(x_row_51)
  norm_x_row_101 = np.linalg.norm(x_row_101)

  # 计算单位向量
b unit_x_row_1   = x_row_1 / norm_x_row_1
  unit_x_row_2   = x_row_2 / norm_x_row_2
  unit_x_row_51  = x_row_51 / norm_x_row_51
  unit_x_row_101 = x_row_101 / norm_x_row_101
```

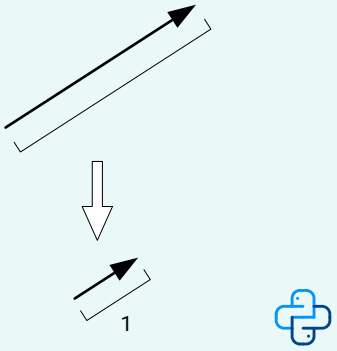


图 6. 计算向量模和单位向量，使用时配合前文代码；Bk1_Ch17_01.ipynb

什么是向量的模？

向量的模 (也称为向量的长度) 是指一个向量从原点到其终点的距离，它是一个标量，表示向量的大小。向量的模通常用两个竖线 $\|\boldsymbol{a}\|$ 来表示，其中 $\boldsymbol{a}$ 表示向量。对于 n 维向量 $\boldsymbol{a} = [a_1, a_2, ..., a_n]$，它的模定义为 $\|\boldsymbol{a}\| = \sqrt{a_1^2 + a_2^2 + \cdots a_n^2}$ 。$\|\boldsymbol{a}\|$ 就是向量各个分量的平方和的平方根。这个公式可以用勾股定理推导得出，因为一个向量的模就是从原点到它的终点的距离，而这个距离可以用勾股定理计算。比如，2 维向量 $\boldsymbol{a} = [3, 4]$ 的模 (长度) 为 $\|\boldsymbol{a}\| = \sqrt{3^2 + 4^2} = 5$ 。

向量内积

本书前文在讲 for 循环时介绍过**向量内积** (inner product)，又叫**标量积** (scalar product)、**点积** (dot product)。给定两个 n 维向量 $\boldsymbol{a} = [a_1, a_2, ..., a_n]$ 和 $\boldsymbol{b} = [b_1, b_2, ..., b_n]$，它们的内积定义为 $\boldsymbol{a} \cdot \boldsymbol{b} = a_1b_1 + a_2b_2 + ... + a_nb_n$。内积结果 $\boldsymbol{a} \cdot \boldsymbol{b}$ 显然为标量。

如图 7 所示，我们分别计算向量内积 $\boldsymbol{x}^{(1)} \cdot \boldsymbol{x}^{(2)}$、$\boldsymbol{x}^{(1)} \cdot \boldsymbol{x}^{(51)}$、$\boldsymbol{x}^{(1)} \cdot \boldsymbol{x}^{(101)}$。建议大家在 JupyterLab 中用手输入算式计算图中三个向量内积。

再次强调，向量内积的运算前提是两个向量维数相同，结果为标量。NumPy 中计算向量内积的函数为 **numpy.dot**()。

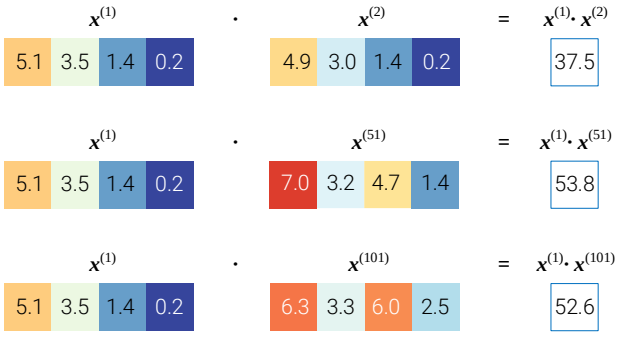


图 7. 向量内积

图 8 中 a 用 numpy.dot() 计算向量内积，x_row_1[0] 结果为一维数组。

注意，如果输入的两个数组都是一维，返回向量内积。如果输入的两个数组都是二维，则返回结果为矩阵乘积，相当于 numpy.matmul() 或 @ 运算符。

```
# 计算向量内积

a inner_prod_x_row_1_2 = np.dot(x_row_1[0], x_row_2[0])
  inner_prod_x_row_1_51 = np.dot(x_row_1[0], x_row_51[0])
  inner_prod_x_row_1_101 = np.dot(x_row_1[0], x_row_101[0])
```

图 8. 计算向量内积，使用时配合前文代码；Bk1_Ch17_01.ipynb

向量夹角

在 Bk1_Ch17_01.ipynb 中，我们计算得到 $\boldsymbol{x}^{(1)}$、$\boldsymbol{x}^{(2)}$的夹角约为 3°，$\boldsymbol{x}^{(1)}$、$\boldsymbol{x}^{(51)}$的夹角约为 22°，$\boldsymbol{x}^{(1)}$、$\boldsymbol{x}^{(101)}$的夹角约为 31°。

这显然不是巧合，$\boldsymbol{x}^{(1)}$、$\boldsymbol{x}^{(2)}$分别代表两朵鸢尾花，它们同属 Setosa，因此最为相似。而 $\boldsymbol{x}^{(51)}$属于 Versicolour，$\boldsymbol{x}^{(101)}$属于 Virginica。这就是向量夹角在机器学习中的一个应用举例。

如图 9 代码所示，a 先计算两个单位向量的内积，即向量夹角的余弦值。b 用 numpy.arccos() 将余弦值转化为弧度 (radian)。c 再用 numpy.rad2deg() 将弧度转化为角度 (degree)。

```
# 计算单位向量内积
a dot_product_1_51 = np.dot(unit_x_row_1[0],
                            unit_x_row_51[0])
# 将结果转化为弧度
b angle_1_51 = np.arccos(dot_product_1_51)
# 将结果转化为角度
c angle_1_51 = np.rad2deg(angle_1_51)
```

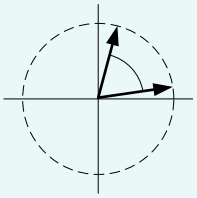


图 9. 计算向量夹角，使用时配合前文代码；Bk1_Ch17_01.ipynb

什么是向量夹角？

向量夹角是指两个向量之间的夹角，它是一个标量，通常用弧度或角度来表示。向量夹角的计算是通过向量内积和向量模的关系得出的。对于两个非零向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$，它们的夹角 θ 定义为 $\cos(\theta) = (\boldsymbol{a} \cdot \boldsymbol{b}) / (\|\boldsymbol{a}\| \|\boldsymbol{b}\|)$。其中 $\boldsymbol{a} \cdot \boldsymbol{b}$ 是向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 的内积，$\|\boldsymbol{a}\|$ 和 $\|\boldsymbol{b}\|$ 分别是向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 的模。注意，这个公式只适用于非零向量，因为对于零向量，它没有方向，因此无法定义夹角。此外，$\cos(\theta)$ 可以看成是 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 的单位向量的向量内积，即 $\cos(\theta) = (\boldsymbol{a}/\|\boldsymbol{a}\|) \cdot (\boldsymbol{b}/\|\boldsymbol{b}\|)$。

通过向量夹角的计算，我们可以判断两个向量之间的相对方向。如果两个向量的夹角为零度，表示它们的方向相同；如果夹角为 90 度，表示它们互相垂直；如果夹角为 180 度，表示它们的方向相反。在机器学习中，可以通过计算向量夹角来度量两个样本之间的相似性。

试想，如果要求大家计算鸢尾花数据矩阵 X 所有行向量之间两两夹角，又要避免使用 for 循环，该怎么办？

我们就需要借助向量化运算 (vectorization)。如图 11 代码所示，a 先对矩阵 $\boldsymbol{X}$ 的每一行向量求模。

其中，axis=1 参数指示 numpy.linalg.norm() 在二维数组的每一行上计算向量范数，而不是在整个数组上计算。这意味着函数将返回一个新的数组，其中每个元素代表了 $\boldsymbol{X}$ 的相应行的向量模。

参数 keepdims=True 这个设置非常重要当设置为 True，则结果将具有与输入数组相同的维度，这意味着结果不再是一维数组，而是二维数组，便在后续除法操作中能够正确地广播。

b 对矩阵 $\boldsymbol{X}$ 的每一行向量进行单位化。好奇的话，大家可以随机选择几行向量，并计算它们的模，看看是不是都为 1。

c 利用矩阵乘法计算行向量单位化矩阵的格拉姆矩阵。格拉姆矩阵为对称方阵，在本例中这个格拉姆矩阵的大小为 150 × 150。如图 12 所示，这个格拉姆矩阵每个元素代表一对行向量的余弦值。

特别地，如图 12 所示，这个格拉姆矩阵的对角线元素都为 1，原因很简单，任一行向量和自身的夹角为 0 度，因此余弦值为 1。格拉姆矩阵 (Gram matrix) 这个概念非常重要，本章后文会深入介绍。

d 将 row_cos_matrix 数组中大于 1 的元素替换为 1，其他元素保持不变。这是一种非常常见的操作，用于数据修正。在数值运算时计算误差不可避免，本例中有些数值略大于 1，需要做出一定调整；否则，下一步求反余弦会报错。

e 将余弦值转化为弧度。f 将弧度值转化为角度。

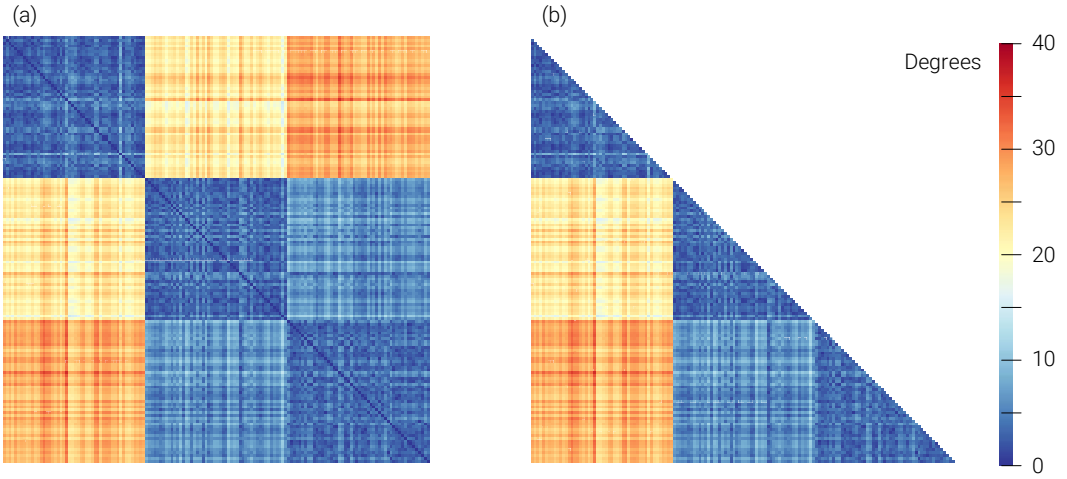


图 10. 鸢尾花数据行向量成对夹角角度矩阵

```
# 计算行向量模
a X_row_norm = np.linalg.norm(X, axis=1,
                              keepdims=True)

# 行向量单位化，利用广播原则
b X_row_unit = X/X_row_norm

# 计算单位行向量的格拉姆矩阵
# 相当于计算成对内积（余弦值）
c row_cos_matrix = X_row_unit @ X_row_unit.T

# 调整计算误差
d row_cos_matrix[row_cos_matrix>1] = 1

# 将余弦值转化为弧度
e row_radian_matrix = np.arccos(row_cos_matrix)

# 将弧度转化为
f row_degree_matrix = np.rad2deg(row_radian_matrix)
```

图 11. 计算成对行向量夹角矩阵，使用时配合前文代码； Bk1_Ch17_01.ipynb

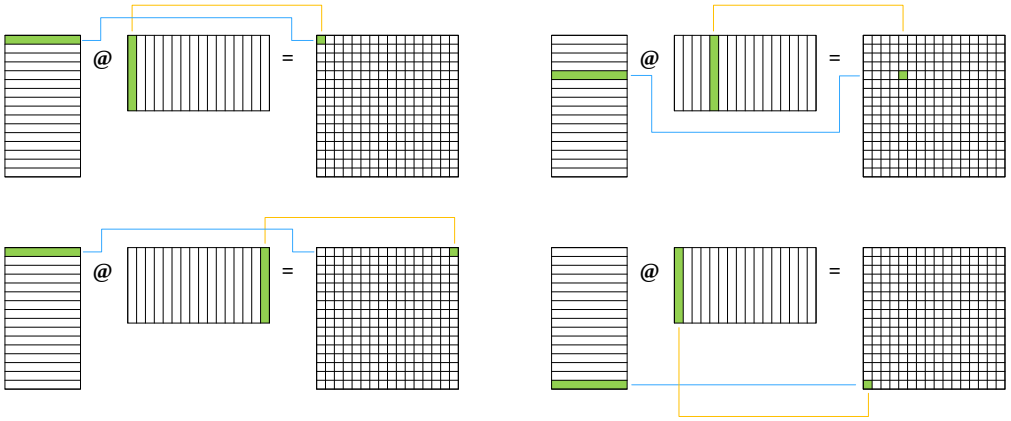


图 12. 格拉姆矩阵对角线元素和非对角元素

请大家修改图 11 代码，自行计算矩阵 $\boldsymbol{X}$ 成对列向量的角度矩阵。

什么是格拉姆矩阵？

格拉姆矩阵 (Gram matrix) 是一个重要的矩阵，它由向量集合的内积组成。给定一个向量集合 $\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\}$，则其对应的格拉姆矩阵 $\boldsymbol{G}$ 定义为 $\boldsymbol{G} = [g_{i,j}]$，其中 $g_{i,j} = \boldsymbol{x}_i \cdot \boldsymbol{x}_j$，表示第 i 个向量和第 j 个向量的内积。格拉姆矩阵是对称矩阵。

格拉姆矩阵在许多应用中都有广泛的应用，例如在机器学习中的支持向量机 (Support Vector Machine, SVM) 算法和核方法 (kernel method) 中，格拉姆矩阵可以用来计算向量之间的相似度和距离，从而实现非线性分类和回归。此外，格拉姆矩阵也可以用于矩阵分解、图像处理、信号处理等领域。

17.4 矩阵运算

矩阵乘法

本书前文介绍过矩阵乘法，假设 $\boldsymbol{A}$ 是一个 $m \times n$ 的矩阵，$\boldsymbol{B}$ 是一个 $n \times p$ 的矩阵，则它们的乘积 $\boldsymbol{C} = \boldsymbol{AB}$ 是一个 $m \times p$ 的矩阵，相当于“消去”n。Python 中可以使用@作为 NumPy 的矩阵乘法运算符。

在本节配套的 Jupyter Notebook 文件中大家可以看到两个有趣的矩阵乘法。

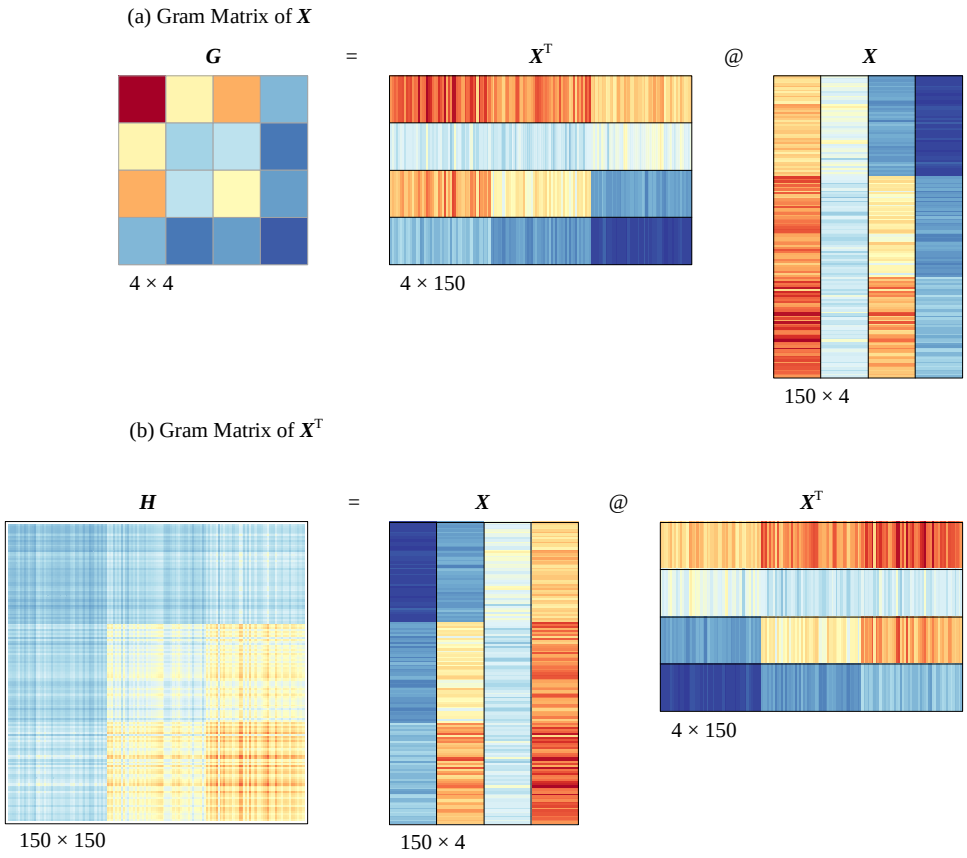


图 13. 两个格拉姆矩阵

如图 13 (a) 所示，鸢尾花数据矩阵的转置 $\boldsymbol{X}^{\mathrm{T}}$ 乘 $\boldsymbol{X}$ 得到 $\boldsymbol{G}$。$\boldsymbol{X}^{\mathrm{T}}$ 的形状为 4×150，$\boldsymbol{X}$ 的形状为 150×4。$\boldsymbol{G} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ 的结果形状为 4×4。$\boldsymbol{G}$ 有自己的名字，叫 $\boldsymbol{X}$ 的格拉姆矩阵 (Gram matrix)。图 13 (b) 所示的 $\boldsymbol{H} = \boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ 的结果形状为 150×150。$\boldsymbol{H}$ 相当是 $\boldsymbol{X}^{\mathrm{T}}$ 的格拉姆矩阵。

格拉姆矩阵有很多有趣的性质，《矩阵力量》一册将详细介绍。

本章中，大家仅仅需要知道格拉姆矩阵为对称矩阵。$\boldsymbol{G}$ 的主对角线上元素是 $\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_i$，即 $\boldsymbol{x}_i \cdot \boldsymbol{x}_i$。如图 14 上图所示，$\boldsymbol{G}$ 的主对角线第一元素 $g_{1,1} = \boldsymbol{x}_1^{\mathrm{T}}\boldsymbol{x}_1 = \boldsymbol{x}_1 \cdot \boldsymbol{x}_1$。如图 14 下图所示，$\boldsymbol{G}$ 的主对角线第二元素 $g_{2,2} = \boldsymbol{x}_2^{\mathrm{T}}\boldsymbol{x}_2 = \boldsymbol{x}_2 \cdot \boldsymbol{x}_2$。请大家自行计算 $\boldsymbol{G}$ 的主对角线剩余两个元素。

如图 15 所示，显然 $g_{2,1} = g_{1,2}$。也就是说，$\boldsymbol{x}_2^{\mathrm{T}}\boldsymbol{x}_1 = \boldsymbol{x}_1^{\mathrm{T}}\boldsymbol{x}_2 = \boldsymbol{x}_2 \cdot \boldsymbol{x}_1 = \boldsymbol{x}_1 \cdot \boldsymbol{x}_2$。

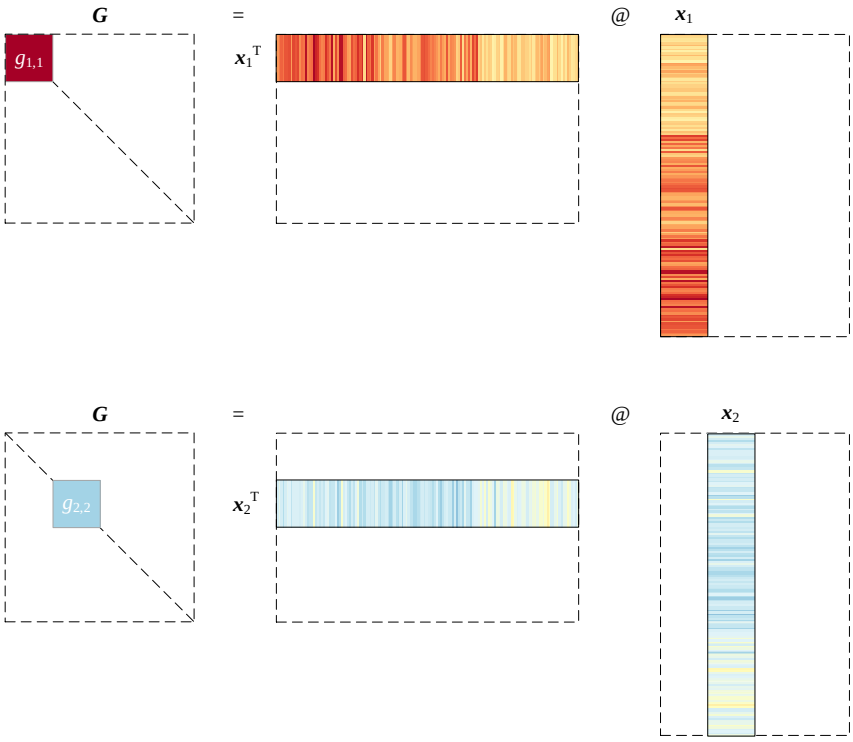


图 14. 格拉姆矩阵 $\boldsymbol{G}$ 主对角线元素

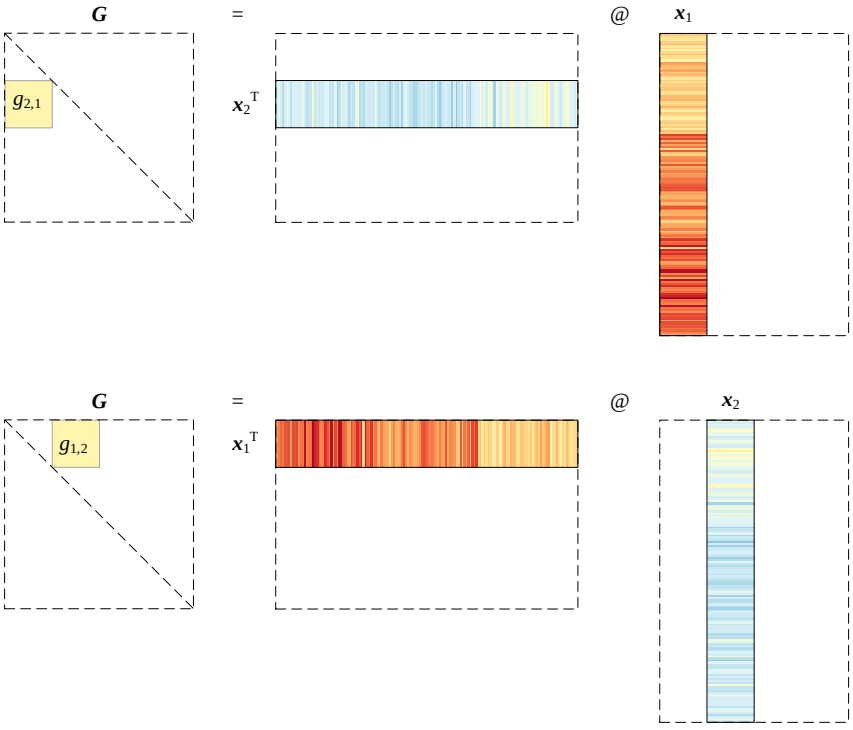


图 15. $\boldsymbol{G}$ 为对称矩阵

a 计算格拉姆矩阵 $\boldsymbol{G}$。b 计算格拉姆矩阵 $\boldsymbol{H}$。

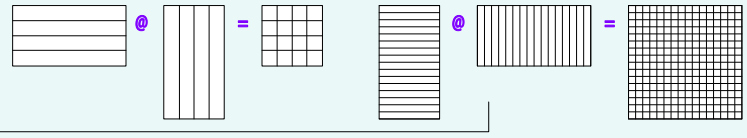
c 创建一个包含 1 行 5 列子图的图形，并且指定每个子图的宽度比例。

1, 5 表示创建一个包含 1 行和 5 列子图的图形布局。

figsize=(8, 3) 指定图形的大小为宽度 8 英寸、高度 3 英寸。

gridspec_kw={'width_ratios': [3, 0.5, 3, 0.5, 3]}使用 gridspec_kw 来指定网格参数。在这里，width_ratios 是一个列表，指定了每列子图的宽度比例。具体来说，第 1、3、5 列子图的宽度比例为 3，而第 2 和第 4 列子图的宽度比例为 0.5。

d 中 sca 是 "Set Current Axes" 的缩写，它是 Matplotlib 提供的一个函数，用于将绘图的当前坐标轴切换到指定的坐标轴。e 利用 seaborn.heatmap() 在第 1 幅子图绘制格拉姆矩阵 $\boldsymbol{G}$。f 在绘图中隐藏第 2 幅子图坐标轴。g 在第 3 幅子图绘制矩阵 $\boldsymbol{X}^{\mathrm{T}}$ 的热图。h 在绘图中隐藏第 4 幅子图坐标轴。i 在第 5 幅子图绘制矩阵 $\boldsymbol{X}$ 的热图。



```
# 第一个格拉姆矩阵
G = X.T @ X
# 第二个格拉姆矩阵
H = X @ X.T

# 可视化第一个格拉姆矩阵运算
fig,axs = plt.subplots(1,5,figsize = (8,3),
                       gridspec_kw={'width_ratios':
                                    [3, 0.5, 3, 0.5, 3]})

# 图形状态切换到第1幅子图
plt.sca(axs[0])

# 绘制格拉姆矩阵
ax = sns.heatmap(G, cmap = 'RdYlBu_r',
                 vmax = 5000, vmin = 0,
                 annot = False,
                 fmt=".0f",
                 cbar_kws = {'orientation':'horizontal'},
                 xticklabels = False,
                 yticklabels=False,
                 square = 'equal')
plt.title('$G$')

# 图形状态切换到第2幅子图
plt.sca(axs[1])
plt.title('=')
plt.axis('off')

# 图形状态切换到第3幅子图
plt.sca(axs[2])
# 绘制X转置
ax = sns.heatmap(X.T, cmap = 'RdYlBu_r',
                 vmax = 0, vmin = 8,
                 cbar_kws = {'orientation':'horizontal'},
                 xticklabels = False,
                 yticklabels = False,
                 annot=False)
plt.title('$X^T$')

# 图形状态切换到第4幅子图
plt.sca(axs[3])
plt.title('@')
plt.axis('off')

# 图形状态切换到第5幅子图
plt.sca(axs[4])
# 绘制X
ax = sns.heatmap(X, cmap = 'RdYlBu_r',
                 vmax = 0, vmin = 8,
                 cbar_kws = {'orientation':'horizontal'},
                 xticklabels = False,
                 yticklabels=False,
                 annot=False)
plt.title('$X$')
```

图 16. 可视化格拉姆矩阵计算过程，使用时配合前文代码；Bk1_Ch17_01.ipynb

矩阵的逆

矩阵的逆可以被看作是一种倒数的概念。并不是所有格拉姆矩阵，恰好前文的格拉姆矩阵 $\boldsymbol{G}$ 存在逆，记做 $\boldsymbol{G}^{-1}$。如图 17 所示，$\boldsymbol{G}$ 乘 $\boldsymbol{G}^{-1}$ 结果为单位阵 $\boldsymbol{I}$。不难看出来，$\boldsymbol{G}^{-1}$ 也是个对称矩阵。

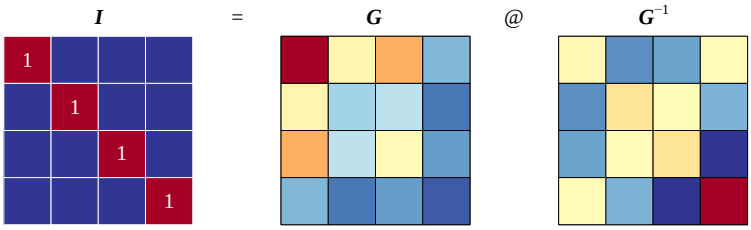


图 17. 格拉姆矩阵 $\boldsymbol{G}$ 的逆

```
# 计算格拉姆矩阵G的逆矩阵
G_inv = np.linalg.inv(G)
```

图 18. 计算格拉姆矩阵逆矩阵，使用时配合前文代码；Bk1_Ch17_01.ipynb

什么是矩阵的逆？

矩阵的逆是一个重要的概念，它是指对于一个可逆的 (即非奇异的) $n \times n$ 矩阵 $\boldsymbol{A}$，存在一个 $n \times n$ 矩阵 $\boldsymbol{B}$，使得 $\boldsymbol{AB} = \boldsymbol{BA} = \boldsymbol{I}$，其中 $\boldsymbol{I}$ 是单位矩阵。$\boldsymbol{B}$ 被称为 $\boldsymbol{A}$ 的逆矩阵，通常用 $\boldsymbol{A}^{-1}$ 表示。矩阵的逆可以被看作是一种倒数的概念，它可以使我们在矩阵运算中除以矩阵，从而解决线性方程组和其他问题。如果我们需要求解一个线性方程组 $\boldsymbol{Ax} = \boldsymbol{b}$，其中 $\boldsymbol{A}$ 是一个可逆矩阵，那么可以使用矩阵的逆来计算 $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$，从而得到方程的解。需要注意的是，并非所有矩阵都有逆矩阵，只有可逆矩阵才有逆矩阵。对于一个不可逆矩阵，它可能是奇异的 (即行列式为 0)，也可能是非方阵。在实际应用中，矩阵的逆通常通过 LU 分解、QR 分解、Cholesky 分解等方法来计算，而不是直接求解逆矩阵。

17.5 几个常见矩阵分解

Cholesky 分解

所幸前文的格拉姆矩阵 G 也是个正定矩阵 (positive definite matrix)，我们可以对它进行 Cholesky 分解。如图 19 所示，$\boldsymbol{L}$ 是个下三角矩阵，它的转置 $\boldsymbol{L}^{\mathrm{T}}$ 为上三角矩阵。$\boldsymbol{L}$ 和 $\boldsymbol{L}^{\mathrm{T}}$ 的乘积也相当于“平方”。NumPy 中完成 Cholesky 分解的函数为 **numpy.linalg.cholesky**()。

《矩阵力量》第 12 章专门讲解 Cholesky 分解，这本书第 21 章将介绍正定性。

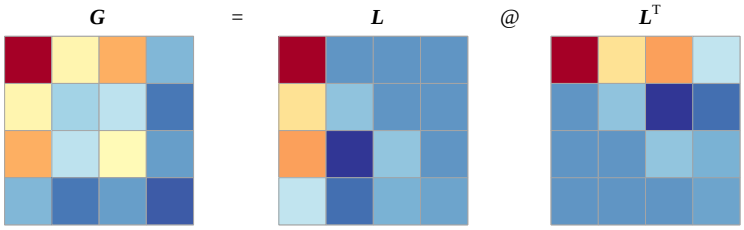


图 19. 对格拉姆矩阵 G 进行 Cholesky 分解

```
# 对格拉姆矩阵G进行Cholesky分解
L = np.linalg.cholesky(G)
```

图 20. 格拉姆矩阵 Cholesky 分解，使用时配合前文代码；Bk1_Ch17_01.ipynb

什么是 Cholesky 分解？

Cholesky 分解是一种将对称正定矩阵分解为下三角矩阵和其转置矩阵乘积的数学技术。给定一个对称正定矩阵 A，Cholesky 分解可以将其表示为 $A = LL^T$，其中 L 是下三角矩阵，L^T 是其转置矩阵。Cholesky 分解是一种高效的矩阵分解方法，它可以在数值计算中减少误差，同时可以加速线性方程组的求解，特别是对于大型的稠密矩阵。因此，Cholesky 分解在很多领域都有广泛的应用，例如统计学、金融学、物理学、工程学等。Cholesky 分解也是一些高级技术的基础，例如蒙特卡洛模拟、Kalman 滤波等等。

什么是正定矩阵？

Cholesky 分解是一种将对称正定矩阵分解为下三角矩阵和其转置矩阵乘积的数学技术。给定一个对称正定矩阵 A，Cholesky 分解可以将其表示为 $A = LL^T$，其中 L 是下三角矩阵，L^T 是其转置矩阵。Cholesky 分解是一种高效的矩阵分解方法，它可以在数值计算中减少误差，同时可以加速线性方程组的求解，特别是对于大型的稠密矩阵。因此，Cholesky 分解在很多领域都有广泛的应用，例如统计学、金融学、物理学、工程学等。Cholesky 分解也是一些高级技术的基础，例如蒙特卡洛模拟、Kalman 滤波等等。

特征值分解 EVD

图 21 所示为对格拉姆矩阵 G 的特征值分解。V 的每一列对应特征向量，Λ 的主对角线元素为特征值。

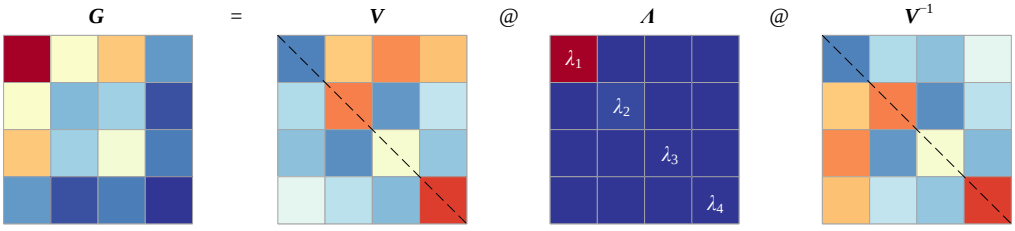


图 21. 对格拉姆矩阵 G 进行 EVD 分解

```
# 对格拉姆矩阵G进行特征值分解
Lambdas, V = np.linalg.eig(G)
```

图 22. 格拉姆矩阵特征值分解，使用时配合前文代码；Bk1_Ch17_01.ipynb

什么是特征值分解？

特征值分解 (Eigenvalue Decomposition, EVD) 是一种将一个方阵分解为一组特征向量和特征值的数学技术。对于一个 $n \times n$ 的矩阵 $\boldsymbol{A}$，如果存在非零向量 $\boldsymbol{v}$ 和常数 λ，使得 $\boldsymbol{Av} = \lambda \boldsymbol{v}$，那么 $\boldsymbol{v}$ 就是矩阵 $\boldsymbol{A}$ 的特征向量，λ 就是对应的特征值。将所有特征向量排列成一个矩阵 $\boldsymbol{V}$，将所有特征值排列成一个对角方阵 $\boldsymbol{\Lambda}$，那么矩阵 $\boldsymbol{A}$ 就可以表示为 $\boldsymbol{A} = \boldsymbol{V\Lambda V}^{-1}$。特征值分解可以帮助我们理解矩阵的性质和结构，以及实现很多数学算法。它在很多领域都有广泛的应用，比如图像处理、机器学习、信号处理、量子力学等。特征值分解也是一些高级技术的基础，例如奇异值分解、QR 分解、LU 分解等。

仔细观察，大家可以已经发现图 21 中 $\boldsymbol{V}$ 和 $\boldsymbol{V}^{-1}$ 关于主对角线对称，即 $\boldsymbol{V}^{\mathrm{T}} = \boldsymbol{V}^{-1}$。这并不是巧合，原因是格拉姆矩阵 $\boldsymbol{G}$ 为对称矩阵。而对称矩阵的特征值分解又叫谱分解 (spectral decomposition)。也就是说，$\boldsymbol{G}$ 的谱分解可以写成 $\boldsymbol{G} = \boldsymbol{V\Lambda V}^{\mathrm{T}}$。

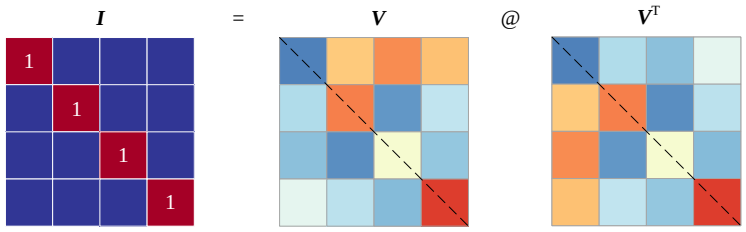


图 23. 谱分解中 $\boldsymbol{V}$ 的特点

《矩阵力量》第 13、14 章专门讲解特征值分解。

什么是谱分解？

谱分解 (Spectral Decomposition) 是将对称矩阵分解为一组特征向量和特征值的数学技术，即对称矩阵的特征值分解。对于一个对称矩阵 $\boldsymbol{A}$，谱分解可以将其分解为 $\boldsymbol{A} = \boldsymbol{Q\Lambda Q}^{\mathrm{T}}$，其中 $\boldsymbol{Q}$ 是由矩阵 $\boldsymbol{A}$ 的特征向量组成的正交矩阵，$\boldsymbol{\Lambda}$ 是由矩阵 $\boldsymbol{A}$ 的特征值组成的对角矩阵。谱分解在很多领域都有广泛的应用，例如图像处理、信号处理、量子力学等。谱分解可以帮助我们理解对称矩阵的性质和结构，从而帮助我们分析和处理各种问题。谱分解也是很多高级技术的基础，例如奇异值分解、主成分分析、矩阵函数等。

奇异值分解 SVD

奇异值分解可谓“最重要的矩阵分解，没有之一”。图 24 所示为对鸢尾花数据矩阵 $\boldsymbol{X}$ 的奇异值分解。图 24 中 $\boldsymbol{S}$ 的主对角线上的元素叫奇异值。大家会在 Bk1_Ch17_01.ipynb 看到，图 21 中特征值开方的结果就是图 24 中的奇异值，这当然不是巧合！

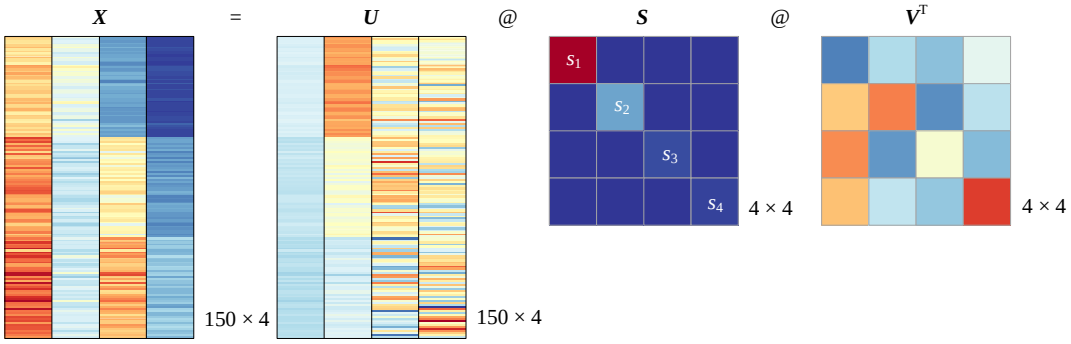


图 24. 对 $\boldsymbol{X}$ 进行 SVD 分解

```
# 鸢尾花数据矩阵X奇异值分解
U,S,VT = np.linalg.svd(X, full_matrices = False)
```

图 25. 鸢尾花数据矩阵奇异值分解，使用时配合前文代码；Bk1_Ch17_01.ipynb

什么是奇异值分解？

奇异值分解 (Singular Value Decomposition, SVD) 是一种将一个矩阵分解为三个矩阵乘积的数学技术。给定一个矩阵 $\boldsymbol{A}$，它可以表示为 $\boldsymbol{A} = \boldsymbol{USV}^{\mathrm{T}}$，其中 $\boldsymbol{U}$ 和 $\boldsymbol{V}$ 是正交矩阵，$\boldsymbol{S}$ 是对角矩阵，对角线上的元素称为奇异值。SVD 可以将一个矩阵的信息分解为不同奇异值所对应的向量空间，并按照奇异值大小的顺序进行排序，使得我们可以仅使用前面的奇异值和相应的向量空间来近似地表示原始矩阵。这种分解在降维、压缩、数据处理和模型简化等领域中有着广泛的应用，例如推荐系统、图像压缩、语音识别等。

如图 26 所示，$\boldsymbol{U}$ 的转置 $\boldsymbol{U}^{\mathrm{T}}$ 和自己乘积为单位阵。如图 27 所示，$\boldsymbol{V}$ 和自身转置 $\boldsymbol{V}^{\mathrm{T}}$ 乘积为单位阵。大家是否已经发现图 23 和图 27 竟然相同，这当然也不是巧合。

实际上，图 12 是四种奇异值分解中的一种。《矩阵力量》第 15、16 章专门讲解奇异值分解，并揭开各种“巧合”背后的数学原理。

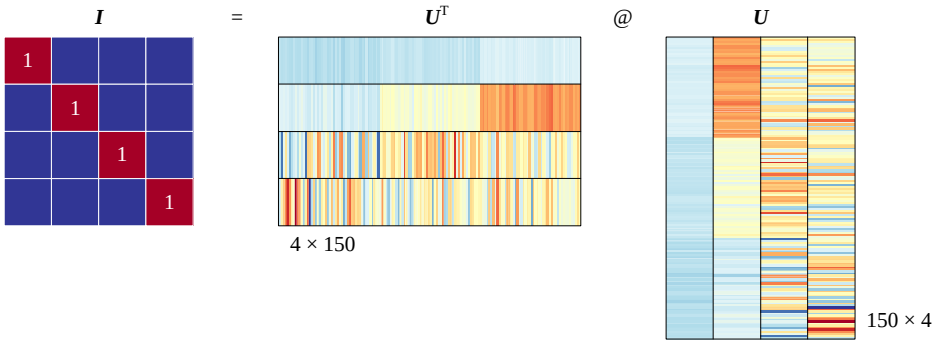


图 26. $\boldsymbol{U}$ 的特点

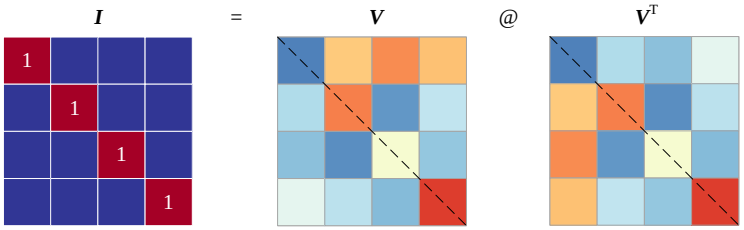


图 27. $\boldsymbol{V}$ 的特点

请大家完成下面 5 道题目。

Q1. 本节配套笔记计算了鸢尾花数据矩阵 $\boldsymbol{X}$ 的若干行向量的模、单位向量、夹角，请大家计算 $\boldsymbol{X}$ 的 4 个列向量的模、单位向量、两两列向量内积、两两夹角。并说明两两列向量内积和图 13 (a) 中格拉姆矩阵的关系。

Q2. 请大家用热图可视化图 13 (a) 中的 $\boldsymbol{G}$ 的第 2 行第 3 列元素如何计算得到。

Q4. 请对图 13 (b) 中的格拉姆矩阵进行 Cholesky 分解，并解释报错的原因。

Q4. 请对图 13 (b) 中的格拉姆矩阵进行特征值分解，并比较其特征值和图 21 中特征值关系。

Q5. 请对 $\boldsymbol{X}^{\mathrm{T}}$ 进行奇异值分解，比较和图 24 中 SVD 分解的关系。

* 本节不提供答案。