18 Einstein Summation in NumPy

NumPy 爱因斯坦求和约定

简化线性代数和张量计算

> 我不能教任何人任何东西。我只能让他们思考。
>
> ***I cannot teach anybody anything. I can only make them think.***
>
> —— 苏格拉底 (Socrates) | 古希腊哲学家 | 470 ~ 399 BC

- ◄ `numpy.average()` 计算平均值
- ◄ `numpy.cov()` 计算协方差矩阵
- ◄ `numpy.diag()` 以一维数组的形式返回方阵的对角线元素，或将一维数组转换成对角阵
- ◄ `numpy.einsum()` 爱因斯坦求和约定
- ◄ `numpy.stack()` 将矩阵叠加
- ◄ `numpy.sum()` 求和

18.1 什么是爱因斯坦求和约定?

NumPy 中还有一个非常强大的函数 numpy.einsum()，它完成的是**爱因斯坦求和约定** (Einstein summation convention 或 Einstein notation)。爱因斯坦求和约定，由阿尔伯特·爱因斯坦于 1916 年提出，是一种数学表示法，用于简化线性代数和张量计算中的表达式。

使用 `numpy.einsum()` 时，大家记住一个要点——输入中重复的索引代表元素相乘，输出中消去的索引意味着相加。

举个例子，矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 相乘用 `numpy.einsum()` 函数可以写成：

```
C = numpy.einsum('ij,jk->ik', A, B)
```

如图 1 所示，“->” 之前分别为矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的索引，它们用逗号隔开。矩阵 $\boldsymbol{A}$ 行索引为 `i`，列索引为 `j`。矩阵 $\boldsymbol{B}$ 行索引为 `j`，列索引为 `k`。`j` 为重复索引，因此在这个方向上元素相乘。

“->” 之后为输出结果的索引。输出结果索引为 `ik`，没有 `j`，因此在 `j` 索引方向上存在求和运算。

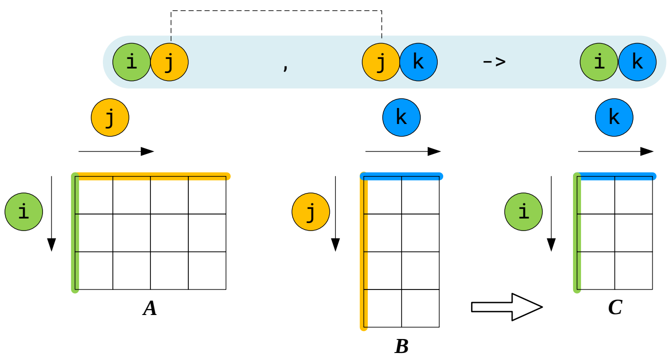

图 1. 利用爱因斯坦求和约定计算矩阵乘法

表 1 总结如何使用 `numpy.einsum()` 完成常见线性代数运算。下面我们选取其中重要的运算配合鸢尾花数据展开讲解。为了方便大家理解，我们在本章中不会介绍爱因斯坦求和约定的具体数学表达，而是通过图解和 Python 实例方式让大家理解这个数学工具。

表 1. 使用 `numpy.einsum()` 完成常见线性代数运算

运算	使用 numpy.einsum()完成运算
向量 $\boldsymbol{a}$ 所有元素求和 (结果为标量)	`numpy.einsum('ij->',a)` `numpy.einsum('i->',a_1D)`
等行数列向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 的逐项积	`numpy.einsum('ij,ij->ij',a,b)` `numpy.einsum('i,i->i',a_1D,b_1D)`
等行数列向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 的向量内积 (结果为标量)	`numpy.einsum('ij,ij->',a,b)` `numpy.einsum('i,i->',a_1D,b_1D)`
向量 $\boldsymbol{a}$ 和自身的张量积	`numpy.einsum('ij,ji->ij',a,a)`

	numpy.einsum('i,j->ij',a_1D,a_1D)
向量 ***a*** 和 ***b*** 的张量积	numpy.einsum('ij,ji->ij',a,b) numpy.einsum('i,j->ij',a_1D,b_1D)
矩阵 ***A*** 的转置	numpy.einsum('ji',A) numpy.einsum('ij->ji',A)
矩阵 ***A*** 所有元素求和 (结果为标量)	numpy.einsum('ij->',A)
矩阵 ***A*** 对每一列元素求和	numpy.einsum('ij->j',A)
矩阵 ***A*** 对每一行元素求和	numpy.einsum('ij->i',A)
提取方阵 ***A*** 的对角元素 (结果为标量)	numpy.einsum('ii->i',A)
计算方阵 ***A*** 的迹 trace(***A***) (结果为标量)	numpy.einsum('ii->',A)
计算矩阵 ***A*** 和 ***B*** 乘积	numpy.einsum('ij,jk->ik', A, B)
乘积 ***AB*** 结果所有元素求和 (结果为标量)	numpy.einsum('ij,jk->', A, B)
矩阵 ***A*** 和 ***B*** 相乘后再转置，即$(\boldsymbol{AB})^{\mathrm{T}}$	numpy.einsum('ij,jk->ki', A, B)
形状相同矩阵 ***A*** 和 ***B*** 逐项积	numpy.einsum('ij,ij->ij', A, B)

18.2 二维数组求和

本节介绍二维数组求和。图 5 代码 a 导入鸢尾花数据矩阵；b 提取四个特征样本数据，保存在 X，结果二维数组；c 提取标签数据。

每一列求和

d 中 np.einsum('ij->j', X) 的含义是对输入数组 X 进行一个特定的操作，其中 'ij->j' 是一个描述操作的字符串。下面，让我们来分解这个字符串。

如图 2 所示，'ij' 表示输入数组 X 的维度索引。'i' 和 'j' 是两个维度索引，通常表示二维数组的行和列。

'->j' 表示输出的维度索引。在这里，'j' 是输出数组的维度索引，表示最终结果的维度。也就是说，'i' 这个索引被压缩、折叠。

所以，np.einsum('ij->j', X) 的操作是将输入二维数组 X 沿着 'i' 维度求和，然后返回一维数组，其维度只有 'j'。执行了列求和操作，将二维数组的每一列相加。

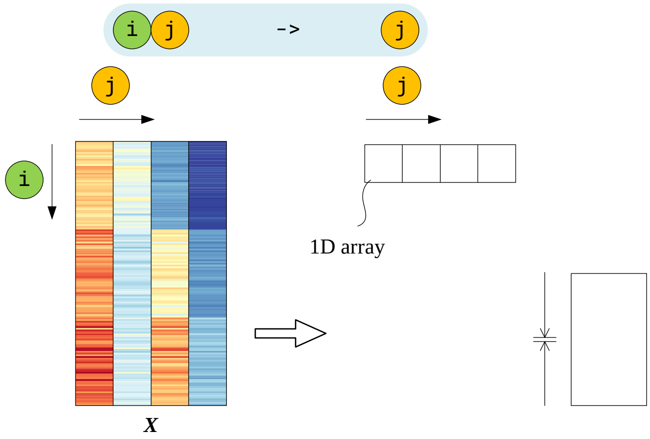

图 2. 利用爱因斯坦求和约定计算每一列求和

每一行求和

类似地，ⓔ将输入二维数组 X 沿着 'j' 维度求和，然后返回一维数组，其维度只有 'i'。执行了行求和操作，将二维数组的每一行相加。

如图 3 所示，'ij' 表示输入数组 X 的维度索引。'i' 和 'j' 是两个维度索引。'->i' 表示输出的维度索引。

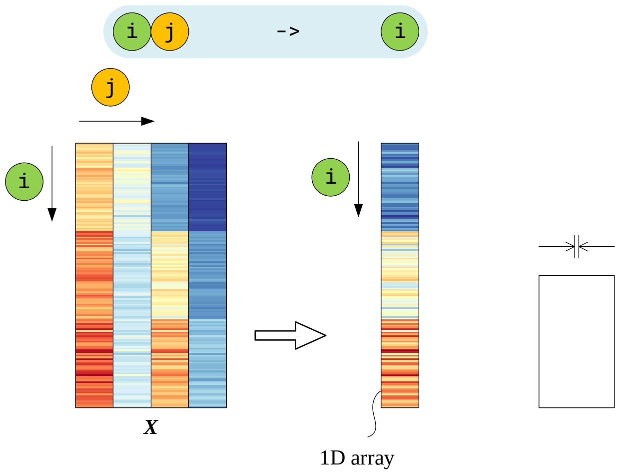

图 3. 利用爱因斯坦求和约定计算每一行求和

所有元素求和

ⓕ中 np.einsum('ij->', X) 的操作是对整个输入二维数组 X 进行汇总，具体操作是将矩阵中的所有元素相加，最终返回一个标量值，表示所有元素的总和。如图 4 所示，'i' 和 'j' 这两个维度索引都被折叠。

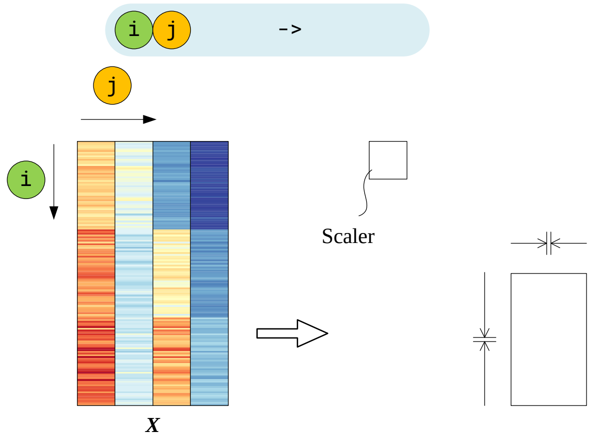

图 4. 利用爱因斯坦求和约定计算矩阵所有元素之和

```python
# 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
y = iris.target

# 每一列求和
np.einsum('ij->j',X)
# np.sum(X, axis = 0)

# 每一行求和
np.einsum('ij->i',X)
# np.sum(X, axis = 1)

# 矩阵所有元素求和
np.einsum('ij->',X)
# np.sum(X, axis = (0,1))
```

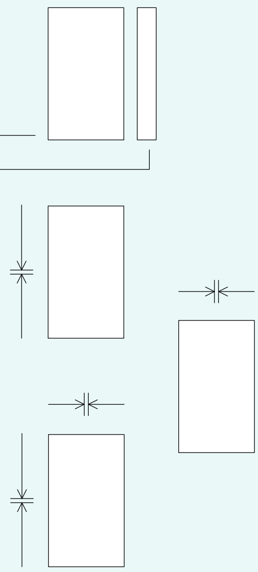

图 5. 爱因斯坦求和约定代码，求和

18.3 转置

二维数组

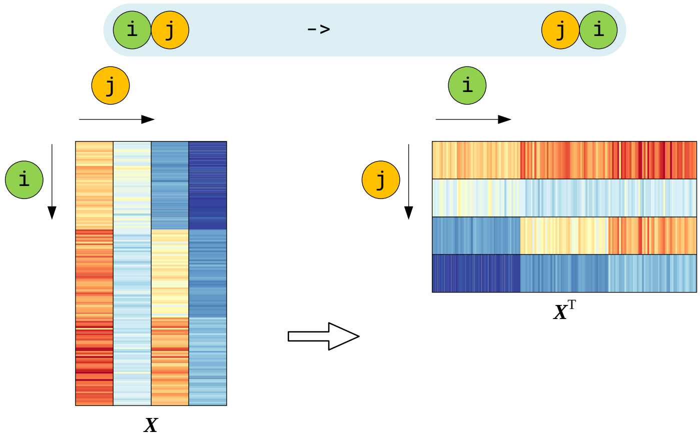

图 6. 利用爱因斯坦求和约定计算二维数组转置

三维数组

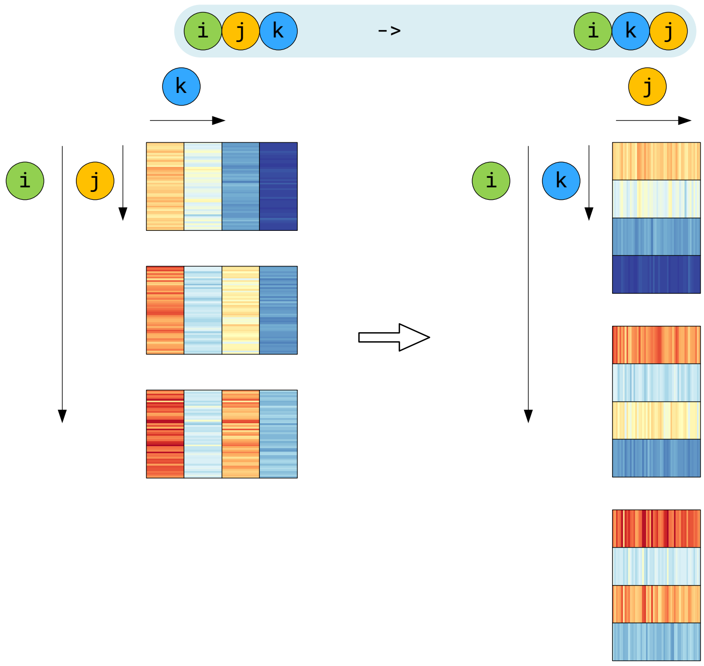

图 7. 利用爱因斯坦求和约定计算三维数组转置

```
# 二维数组转置
np.einsum('ij->ji',X)
# X.T
# np.transpose(X)

# 三维数组
X3D = np.stack([X[y == 0],
                X[y == 1],
                X[y == 2]], axis=0)

# 三维数组转置
X3D_T = np.einsum('ijk->ikj', X3D)
# np.transpose(X3D,(0, 2, 1))
```

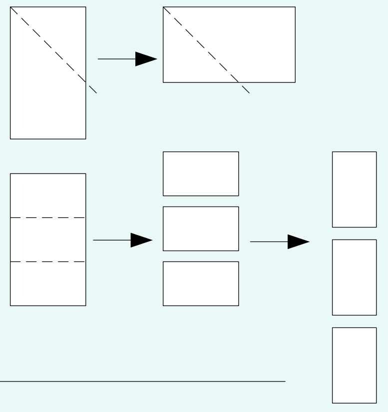

图 8. 爱因斯坦求和约定代码，转置

18.4 矩阵乘法

格拉姆矩阵

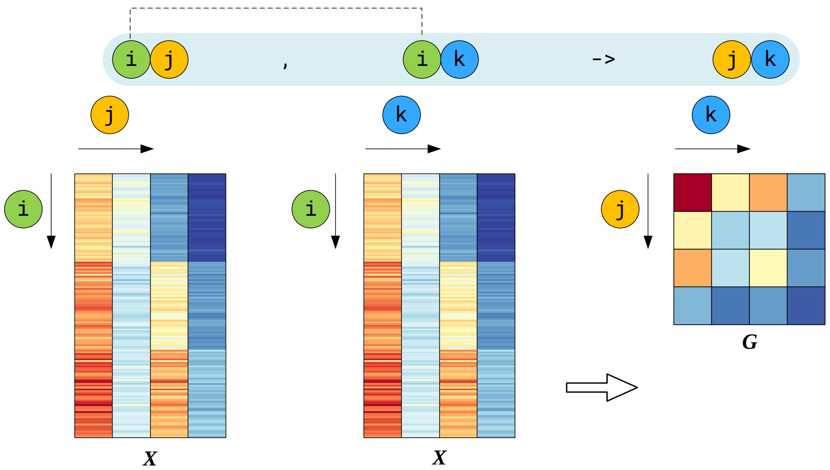

图 9. 利用爱因斯坦求和约定计算格拉姆矩阵 *G*

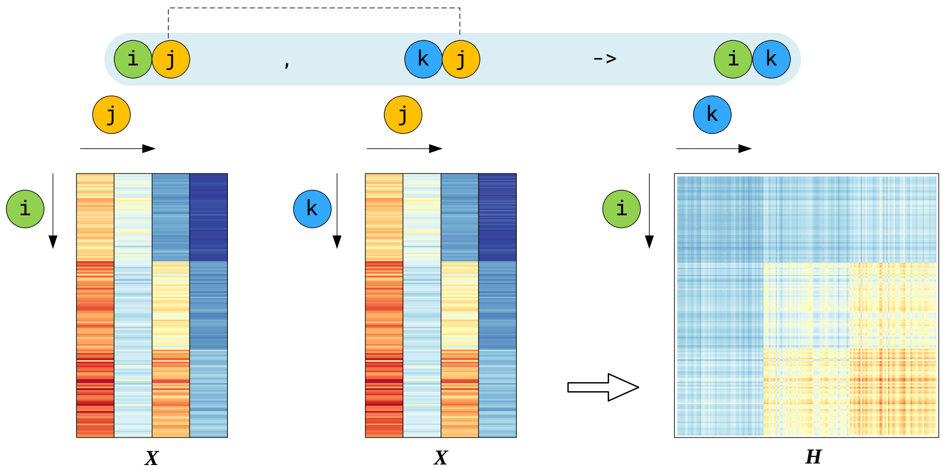

图 10. 利用爱因斯坦求和约定计算格拉姆矩阵 ***H***

分类乘法

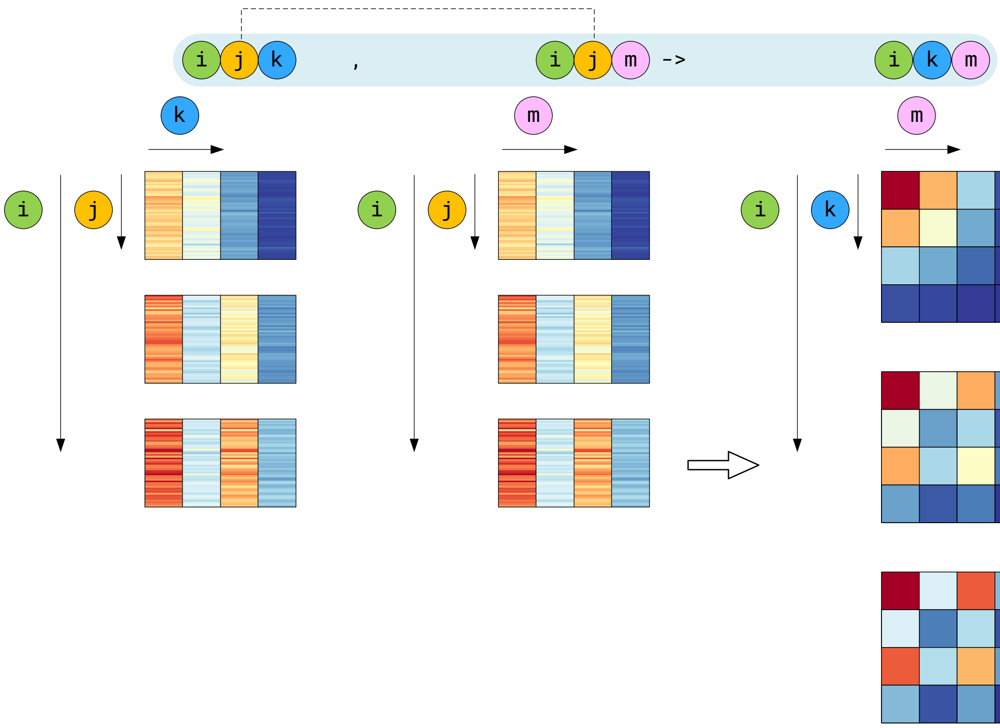

图 11. 利用爱因斯坦求和约定计算格拉姆矩阵，不同鸢尾花类别

```python
# 计算矩阵乘法 X @ X.T
np.einsum('ij,kj->ik', X, X)
# np.einsum('ij,jk->ik', X, X.T)
# X @ X.T

# 计算矩阵乘法 X.T @ X
G = np.einsum('ij,ik->jk', X, X)
# np.einsum('ij,jk->ik', X.T, X)
# X.T @ X

# 三维矩阵乘法
G_3D = np.einsum('ijk,ijm->ikm', X3D, X3D)
# np.einsum('mij,mjk->mik', X3D_T, X3D)
```

图 12. 爱因斯坦求和约定代码，矩阵乘法

18.5 一维数组

求和

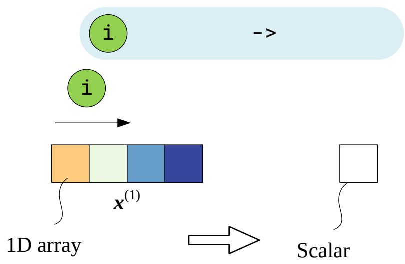

图 13. 利用爱因斯坦求和约定计算一维数组求和

逐项积

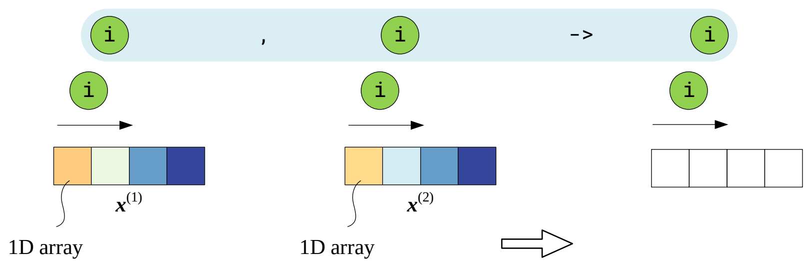

图 14. 利用爱因斯坦求和约定计算一维数组向量逐项积

向量内积

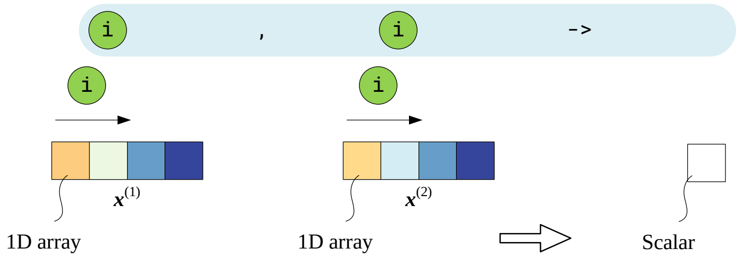

图 15. 利用爱因斯坦求和约定计算一维数组向量逐项积

向量外积

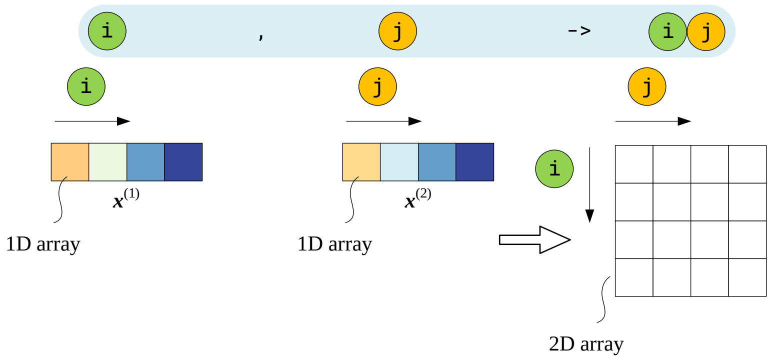

图 16. 利用爱因斯坦求和约定计算一维数组向量逐项积

```python
# 提取两个行向量
a_1D = X[0]
b_1D = X[1]

# 一维向量求和
np.einsum('i->',a_1D)

# 一维向量逐项积
np.einsum('i,i->i',a_1D,b_1D)

# 一维向量内积
np.einsum('i,i->',a_1D,b_1D)

# 一维向量外积
np.einsum('i,j->ij',a_1D,b_1D)
```

图 17. 爱因斯坦求和约定代码，一维数组

18.6 对角方阵

取出对角元素

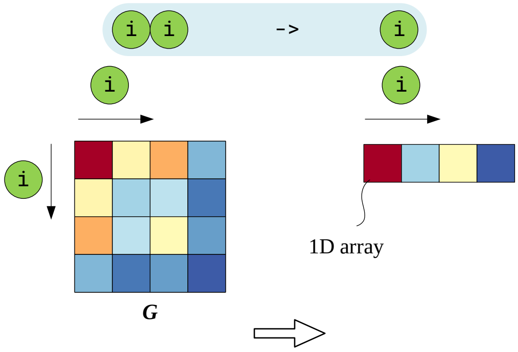

图 18. 利用爱因斯坦求和约定提取对角元素

计算迹

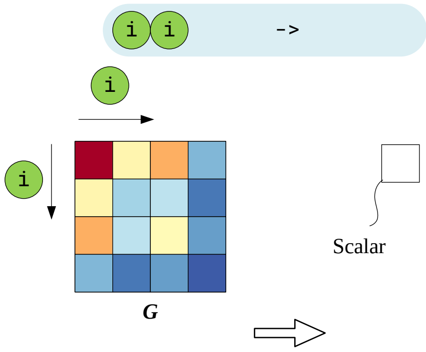

图 19. 利用爱因斯坦求和约定计算方阵迹

```python
#%% 取出方阵对角
np.einsum('ii->i',G)        # a
# np.diag(G)

#%% 计算方阵迹
np.einsum('ii->',G)         # b
# np.trace(G)
```

图 20. 爱因斯坦求和约定代码，对角方阵

18.7 统计运算

计算均值

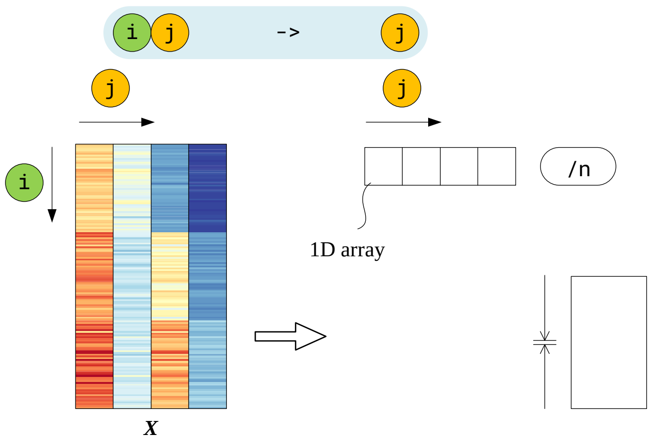

图 21. 利用爱因斯坦求和约定计算每一列均值

计算方差

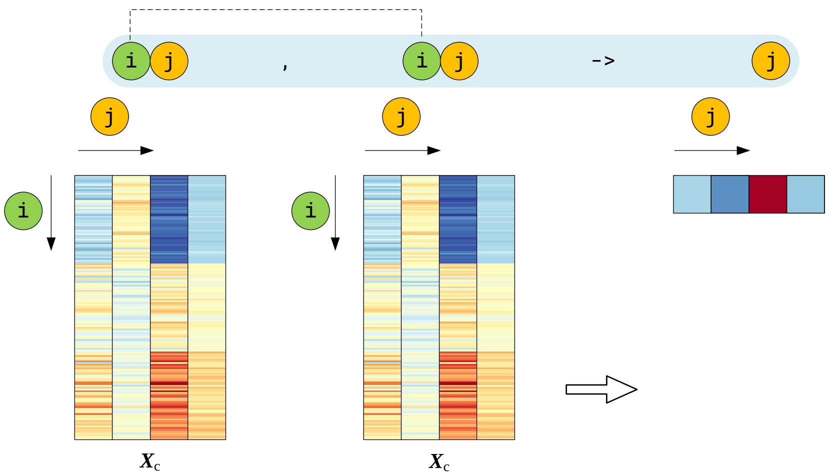

图 22. 利用爱因斯坦求和约定计算方差

计算协方差矩阵

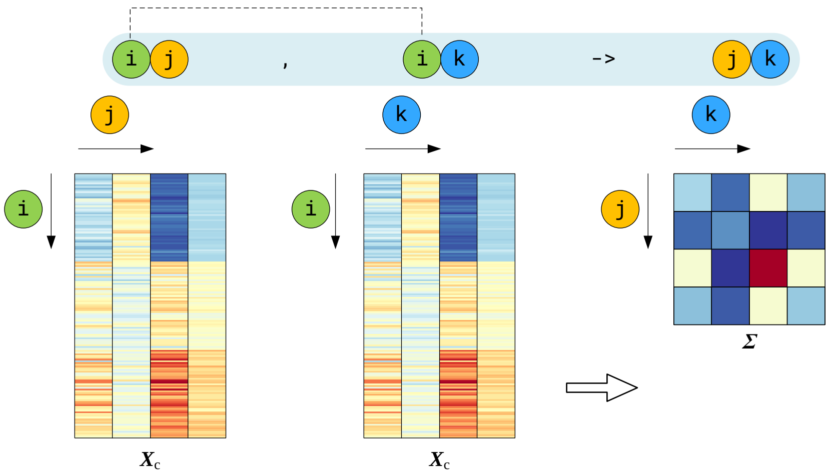

图 23. 利用爱因斯坦求和约定计算协方差矩阵

```
# 计算列均值，质心
n = X.shape[0]  # 样本数量
mean_X = np.einsum('ij->j', X) / n
# np.mean(X, axis = 0)

# 计算方差
X_c = X - mean_X  # 中心化数据
variance = np.einsum('ij,ij->j', X_c, X_c) / (n - 1)
# np.var(X, axis = 0, ddof = 1)

# 计算协方差矩阵
cov_matrix = np.einsum('ij,ik->jk', X_c, X_c) / (n - 1)
# np.cov(X.T, ddof = 1)
```

图 24. 爱因斯坦求和约定代码，统计运算