D07 NumPy 的矩陣函式與線性代數應用

簡報閱讀 範例與作業 問題討論 學習心得(完成)

重要知識點

NumPy 線性代數函式

NumPy 矩陣乘積 – 點積 (dot product)

NumPy 矩陣乘積 – 內積 (inner product)

重要知識點

學習 NumPy 線性代數相關函式，分為下列主題進行介紹。

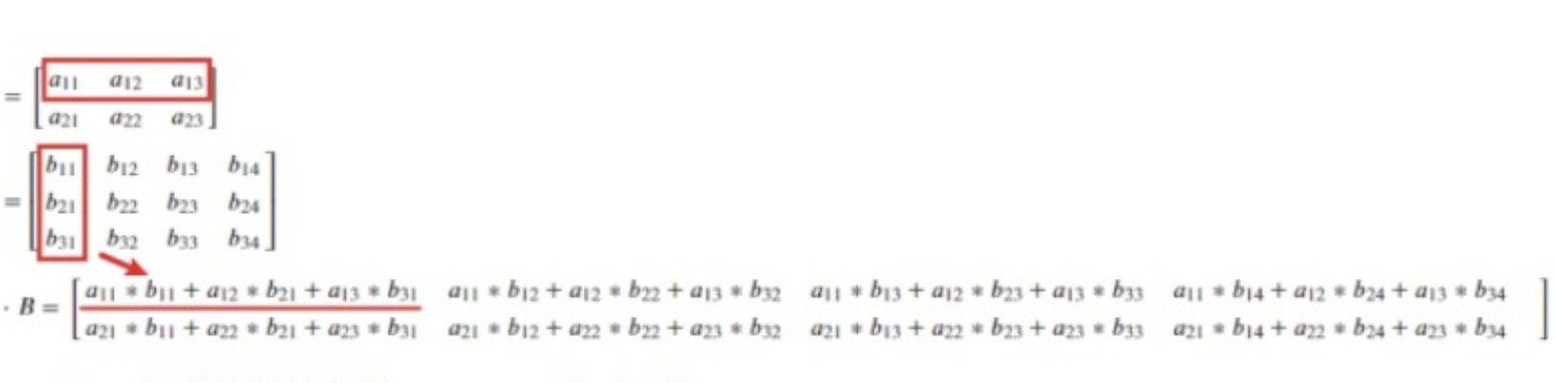
主題	函式
矩陣乘積	點積、內積、外積、矩陣乘法
矩陣操作	跡、行列式、反矩陣、轉置、特徵值與特徵向量、秩、線性系統求解
特殊矩陣	單位矩陣(identity)、單位矩陣(eye)、三角矩陣、單對角陣列、上三角矩陣、下三角矩陣
矩陣分解 (Matrix Decomposition)	Cholesky、QR、SVD

NumPy 線性代數函式

- numpy.linalg 是 NumPy 線性代數模組，提供線性代數函式相關的函式。
- NumPy 線性代數函式提供高效能的線性代數演算法，並且可透 OpenBLAS、MKL、ATLAS 等程式庫進行多線程的運算。
- 除了 NumPy 之外，SciPy 提供更多線性代數函式，可以搭配應用。

NumPy 矩陣乘積 – 點積 (dot product)

- 進行點積運算須注意形狀必須注意形狀 (shape)。若是兩個向量的點積，兩個向量的元素數目也須相同，或其中一個數目為 1 (廣播)。
- 若是兩個多維陣列 (矩陣) 的點積，則中間兩個大小要相同才能進行點積，例如：(2,3)·(3,4)→ 成為 (2,4)
- 如果形狀不符合則無法進行點積。



- NumPy點積函式語法：numpy.dot(a, b)

NumPy 矩陣乘積 – 內積 (inner product)

一般來說內積是 2 個一維陣列(向量)做內積。內積運算公式如下圖：

$$\vec{a} = [a_0, a_1, \dots, a_{j-1}]$$

$$\vec{b} = [b_0, b_1, \dots, b_{j-1}]$$

$$\vec{a} \cdot \vec{b} = \sum a_0b_0 + a_1b_1 + \dots + a_{j-1}b_{j-1}$$

- 如果是多維陣列做乘積，那麼最後維度必須要符合，而內積結果就是其元素相乘之和。
- NumPy 內積函式語法：numpy.inner(a, b)

NumPy 矩陣乘積 – 外積 (outer product)

- 在線性代數中外積也稱為張量積。
- 外積是 2 個一維陣列(向量)做計算，若非一維陣列的話會先進行展平(flatten)再進行外積。公式及範例如右圖：

$$\vec{a} = [a_0, a_1, \dots, a_{i-1}]$$

$$\vec{b} = [b_0, b_1, \dots, b_{j-1}]$$

$$\vec{a} \otimes \vec{b} = \begin{bmatrix} a_0b_0 & a_0b_1 & \dots & a_0b_{j-1} \\ a_1b_0 & a_1b_1 & \dots & a_1b_{j-1} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ a_{i-1}b_0 & a_{i-1}b_1 & \dots & a_{i-1}b_{j-1} \end{bmatrix}$$

```
np.outer(a, b)
```

NumPy 矩陣乘法 – matmul 與 @

matmul 與 dot 都是矩陣乘法 (Matrix Multiplication)，兩者非常類似，其相同及不同點如下：

- 如果 2 個都是二維陣列的話，matmul 與 dot 相同。
- 在 matmul 中，多維的矩陣，將前 n-2 維視為後 2 維的元素後，進行乘法運算。
- matmul 不允許矩陣與純量相乘。

在 Python 3.5 版本之後提供 @ 做為矩陣相乘運算子，在多個矩陣相乘時提供更簡潔的語法。

matmul 與 @ 的計算結果完全相同。

```
A = np.arange(6).reshape(2, 3)
B = np.arange(12).reshape(3, 4)

np.matmul(A, B)
```

```
array([[20, 23, 26, 29],
       [56, 68, 80, 92]])
```

```
A @ B
```

```
array([[20, 23, 26, 29],
       [56, 68, 80, 92]])
```

NumPy 矩陣操作 – 跡(trace)

跡的運算說明如下圖：

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$trace(A) = a_{11} + a_{22} + a_{33}$$

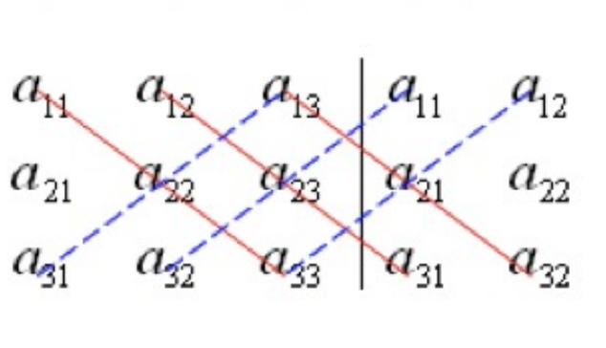
跡的運算可以透過 trace() 函式求得：np.trace(A)

NumPy 矩陣操作 – 行列式(determinant)

行列式的運算說明如下圖：

$$|A| = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

$$= a_{11}\,a_{22}\,a_{33} + a_{12}\,a_{23}\,a_{31} + a_{13}\,a_{21}\,a_{32} - a_{13}\,a_{22}\,a_{31} - a_{23}\,a_{32}\,a_{11} - a_{33}\,a_{12}\,a_{21}$$



圖解行列式運算 (Source: Wikipedia)

行列式的運算可以透過 numpy.linalg.det() 函式求得：np.linalg.det(A)

NumPy 矩陣操作 – 反矩陣(inverse)

反矩陣的運算說明如下圖：

以三階矩陣為例：

$$A = \left[\begin{array}{ccc|ccc} a_{11} & a_{12} & a_{13} & 1 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 1 & 0 \\ a_{31} & a_{32} & a_{33} & 0 & 0 & 1 \end{array}\right]$$

透過高斯約當法 (Gaussian Elimination)，可以得到右邊的矩陣即為 A 的反矩陣

$$A^{-1} = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & b_{11} & b_{12} & b_{13} \\ 0 & 1 & 0 & b_{21} & b_{22} & b_{23} \\ 0 & 0 & 1 & b_{31} & b_{32} & b_{33} \end{array}\right]$$

反矩陣的運算可以透過 numpy.linalg.inv() 函式求得：np.linalg.inv(A)

NumPy 矩陣操作 – 轉置 (Transpose)

轉置的運算說明如下圖：

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$A^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}$$

轉置矩陣可透過呼叫 numpy.transpose() 函式達成。

NumPy 矩陣操作 – 特徵值與特徵向量(eig)

eig() 函式可以用來計算方陣的特徵值(eigenvalue)與特徵向量(eigenvector)。
eig() 回傳值有2個：eigenvalues, eigenvectors。

另外 eigvals() 函式也可以計算特徵值。

```
w, v = np.linalg.eig(B)
```

```
# eigenvalues
w
```

```
array([-5.,  3.,  6.])
```

```
# eigenvectors
v
```

```
array([[ 0.81649658,  0.53452248,  0.05842062],
       [ 0.40824829, -0.80178373,  0.35052374],
       [-0.40824829, -0.26726124,  0.93472998]])
```

```
np.linalg.eigvals(B)
```

```
array([-5.,  3.,  6.])
```

NumPy 矩陣操作 – 秩(rank)

呼叫 matrix_rank() 回傳矩陣的秩(rank)。

```
A = np.array([[3, 0, -1],[2, 4, 1],[7, 2, 3]])
np.linalg.matrix_rank(A)
```

```
3
```

NumPy是使用SVD (奇異值分解；singular value decomposition) 來計算秩。

NumPy 矩陣操作 – 線性系統求解(solve)

求解線性系統可以使用 solve() 函式。

```
a = np.array([[3,1], [1,2]])
b = np.array([9,8])
x = np.linalg.solve(a, b)
x
```

```
array([2., 3.])
```

NumPy 特殊矩陣 – 單位矩陣(identity)

單位矩陣是對角線元素值為 1 的方陣。
範例：

```
np.identity(5)
```

```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

NumPy 特殊矩陣 – 單位矩陣(eye)

eye() 也是用來產生對角線元素值為 1 的陣列，與 identity() 不同的地方在於 eye() 產生的可以不是方陣，也可以指定對角線開始的索引。
範例：

```
np.eye(5, 4, dtype=int)
```

```
array([[1, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1],
       [0, 0, 0, 0]])
```

```
np.eye(3, k=1)
```

```
array([[0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 0.]])
```

NumPy 特殊矩陣 – 單對角陣列(Diagonal)

diagonal() 與 diag() 語法基本上功能相同，差別在於 diagonal() 可以另行指定軸做為第一個軸或第二個軸。

```
A = np.array([[3, 4, 0], [4, 4, 6], [0, 6, 5]])
```

```
np.diagonal(A, offset=1)
```

```
array([4, 6])
```

也可以使用 array.diagonal() 語法。

```
A.diagonal(offset=1)
```

```
array([4, 6])
```

NumPy 特殊矩陣 – 三角矩陣(tri)

tri() 可以用來產生對角線以下都是 1 的元素，下面的例子是產生形狀 (3,5) 的陣列，從 Column 索引 1 開始的對角線以下元素值均為 1。

```
np.tri(3, 5, 1, dtype=int)
```

```
array([[1, 1, 0, 0, 0],
       [1, 1, 1, 0, 0],
       [1, 1, 1, 1, 0]])
```

NumPy 特殊矩陣 – 上三角矩陣(Upper Triangular)、下三角矩陣(Lower Triangular)

triu()、numpy.tril() 可將對角線以上或以下的元素值設為 0，也可以指定對角線起始的索引。

```
A = np.array([[3, 4, 0], [4, 4, 6], [0, 6, 5]])
A
```

```
array([[3, 4, 0],
       [4, 4, 6],
       [0, 6, 5]])
```

```
np.triu(A)
```

```
array([[3, 4, 0],
       [0, 4, 6],
       [0, 0, 5]])
```

```
np.tril(A, 1)
```

```
array([[3, 4, 0],
       [4, 4, 6],
       [0, 6, 5]])
```

NumPy 矩陣分解 (Matrix Decomposition)

NumPy 線性代數函式中提供3種矩陣分解的函式：

- Cholskey Decomposition
- QR Factorization
- Singular Value Decomposition (SVD)

NumPy 矩陣分解 - Cholskey Decomposition

Cholesky分解語法：numpy.linalg.cholesky()
範例：

```
B = np.array([[1, 2],[2, 50]])
B
```

```
array([[ 1,  2],
       [ 2, 50]])
```

```
np.linalg.cholesky(B)
```

```
array([[1.        , 0.        ],
       [2.        , 6.78232998]])
```

NumPy 矩陣分解 – QR分解

QR分解語法：numpy.linalg.qr()
範例：

```
q, r = np.linalg.qr(B)
```

```
q
```

```
array([[-0.4472136 , -0.89442719],
       [-0.89442719,  0.4472136 ]])
```

```
r
```

```
array([[ -2.23606798, -45.61578674],
       [  0.        ,  20.57182539]])
```

NumPy 矩陣分解 – SVD分解

SVD分解語法：numpy.linalg.svd()
範例：

```
u, s, vh = np.linalg.svd(B)
```

```
u
```

```
array([[ 0.04071476,  0.99917081],
       [ 0.99917081, -0.04071476]])
```

```
s
```

```
array([50.08149711,  0.91850289])
```

```
vh
```

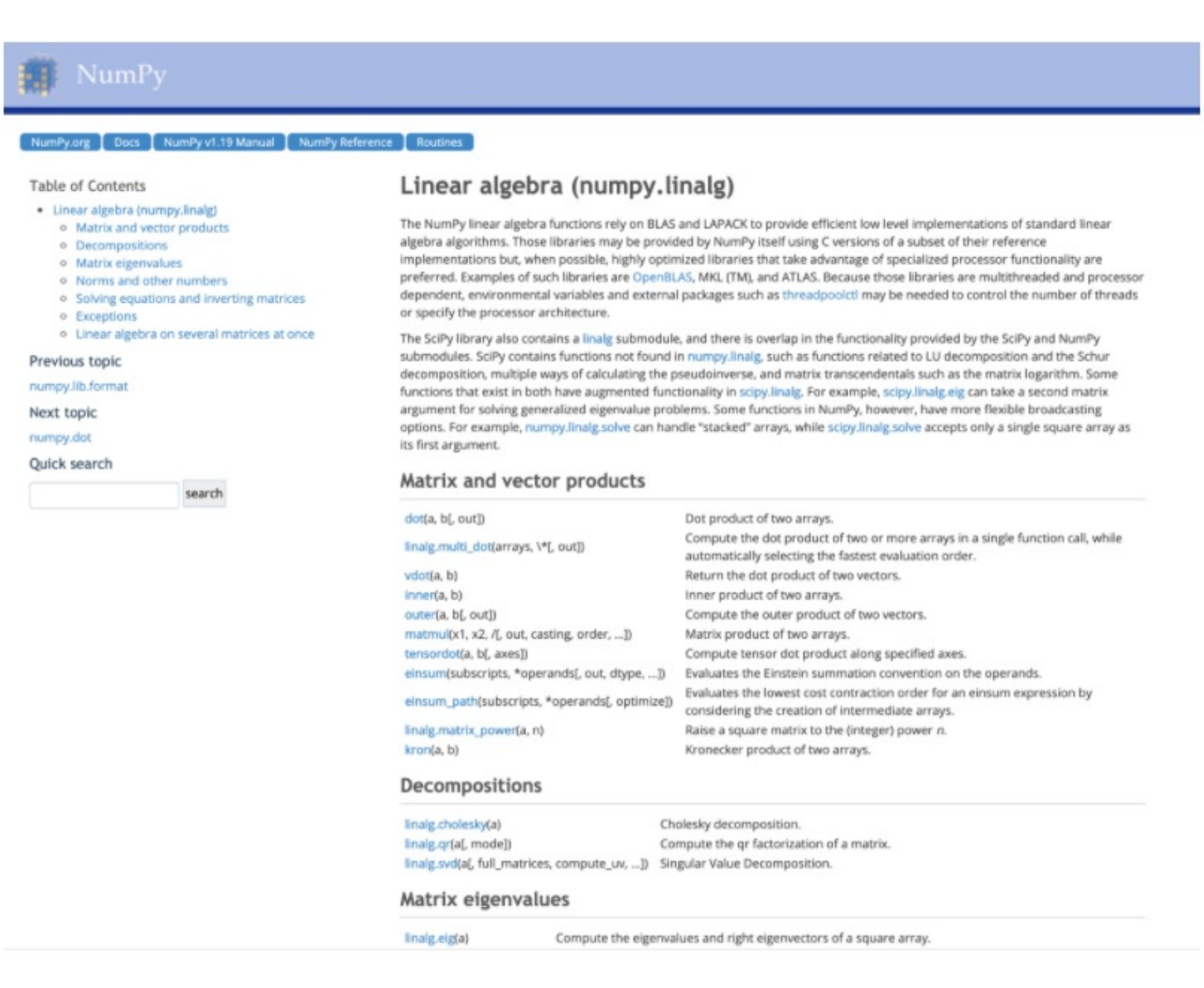
```
array([[ 0.04071476,  0.99917081],
       [ 0.99917081, -0.04071476]])
```

知識點回顧

- 在今天的內容中介紹 NumPy 線性代數模組，示範矩陣的建立、操作、eigenvalues / eigenvectors、特殊矩陣、以及矩陣分解，需要用到線性代數相關運算的話，可以應用這些函式。
- 另外在範例中使用 LaTeX 撰寫矩陣範例，有興趣的話可以在 Jupyter Notebook 中查看語法。

延伸閱讀

NumPy Linear algebra (numpy.linalg) 官方文件
網站：numpy



Linear algebra (numpy.linalg)

The NumPy linear algebra functions rely on BLAS and LAPACK to provide efficient low level implementations of standard linear algebra algorithms. Those libraries may be provided by NumPy itself using C versions of a subset of their reference implementations but, when possible, highly optimized libraries that take advantage of specialized processor functionality are preferred. Examples of such libraries are OpenBLAS, MKL (TM), and ATLAS. Because those libraries are multithreaded and processor dependent, environmental variables and external packages such as threadpoolctl may be needed to control the number of threads or specify the processor architecture.

The SciPy library also contains a linalg submodule, and there is overlap in the functionality provided by the SciPy and NumPy submodules. SciPy contains functions not found in numpy.linalg, such as functions related to LU decomposition and the Schur decomposition, multiple ways of calculating the pseudoinverse, and matrix transcendentals such as the matrix logarithm. Some functions that exist in both have augmented functionality in scipy.linalg. For example, scipy.linalg.eig can take a second matrix argument for solving generalized eigenvalue problems. Some functions in NumPy, however, have more flexible broadcasting options. For example, numpy.linalg.solve can handle "stacked" arrays, while scipy.linalg.solve accepts only a single square array as its first argument.

Matrix and vector products

dot(a, b[, out])	Dot product of two arrays.
linalg.multi_dot(arrays, *[, out])	Compute the dot product of two or more arrays in a single function call, while automatically selecting the fastest evaluation order.
vdot(a, b)	Return the dot product of two vectors.
inner(a, b)	Inner product of two arrays.
outer(a, b[, out])	Compute the outer product of two vectors.
matmul(x1, x2, /[, out, casting, order, …])	Matrix product of two arrays.
tensordot(a, b[, axes])	Compute tensor dot product along specified axes.
einsum(subscripts, *operands[, out, dtype, …])	Evaluates the Einstein summation convention on the operands.
einsum_path(subscripts, *operands[, optimize])	Evaluates the lowest cost contraction order for an einsum expression by considering the creation of intermediate arrays.
linalg.matrix_power(a, n)	Raise a square matrix to the (integer) power n.
kron(a, b)	Kronecker product of two arrays.

Decompositions

linalg.cholesky(a)	Cholesky decomposition.
linalg.qr(a[, mode])	Compute the qr factorization of a matrix.
linalg.svd(a[, full_matrices, compute_uv, …])	Singular Value Decomposition.

Matrix eigenvalues

linalg.eig(a)	Compute the eigenvalues and right eigenvectors of a square array.

下一步：閱讀範例與完成作業