D03 NumPy 陣列運算及數學 Universal Functions (ufunc)

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 陣列運算 - 四則運算
- NumPy 陣列運算 – 次方 np.power()

重要知識點

- NumPy 提供許多數學及統計的函式，這些函式的用法都很相似，針對陣列進行 element-wise 的操作，並回傳陣列做為輸出，稱為 Universal Functions (ufunc)。

NumPy 陣列運算 - 四則運算

陣列的加減乘除四則運算，可以使用運算子 (+, -, *, /) 或是呼叫函式來進行，語法及對照如下表：

運算子	函式	
a + b	np.add(a, b)	加法
a - b	np.substract(a, b)	減法
a * b	np.multiply(a, b)	乘法
a / b	np.divide(a, b)	除法
a % b	np.mod(a, b)	求餘數

以上運算子的部分，使用與 Python 相同的運算子。

運算的時候，陣列的形狀 (shape) 必須相同，或是遵循廣播 (broadcasting) 規則，才能正確進行 element-wise 運算。規則如下：

- 兩個陣列形狀完全相同
- 比較兩個陣列的維度，如果維度的形狀相同的話，可以進行廣播
- 比較兩個陣列的維度，其中一個維度為 1 的話，可以進行廣播

NumPy 陣列運算 – 次方 np.power()

次方的運算，跟四則運算一樣，也要遵循上述的規則，才能成功進行運算。語法如下表：

運算子	函式	a 的 b 次方
a ** b	np.power(a, b)	a^b

NumPy 陣列運算 – 平方根 np.sqrt()

基本語法: np.sqrt(a)，對陣列進行 element-wise 的平方根。
範例：

```
[18]: np.sqrt(4)
[18]: 2.0

[19]: np.sqrt(a)
[19]: array([1.        , 1.41421356, 1.73205081, 2.        , 2.23606798])
```

```
1 np.sqrt(4)
```

```
1 np.sqrt(a)
```

NumPy 陣列運算 – 歐拉數 (Euler's number) 及指數函式 np.exp()

NumPy 提供歐拉常數 *e* (np.e)，以及指數函式 np.exp()，表示 e^X 。
範例：

```
[20]: np.e
[20]: 2.718281828459045

[21]: np.exp(1)
[21]: 2.718281828459045

[22]: np.exp(np.arange(5))
[22]: array([ 1.        ,  2.71828183,  7.3890561 , 20.08553692, 54.59815003])
```

NumPy 陣列運算 – 對數函式

log 函式如下表：

函式	
np.log(x)	底數為e
np.log2(x)	底數為2
np.log10(x)	底數為10
np.log1p(x)	底數為e，計算log(1+x)

若要使用其他底數，可以用下列的方法 (以底數 3 為例)。

```
[24]: np.log(9)/np.log(3)
[24]: 2.0
```

```
1 np.log(9)/np.log(3)
```

若是 log(負數) 則會產生 nan 常數，NaN / NAN 為 nan (not a number) 的別名。

```
np.log([-1, 1, 2])
c:\python\python36_64\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarning: invalid value encountered in log
  """Entry point for launching an IPython kernel.
array([       nan, 0.        , 0.69314718])
```

NumPy 陣列運算 – 取近似值

取近似值的函式及說明如下表：

函式	常用語法	說明
round(), around()	ndarray.round(decimals=0) numpy.round(a, decimals=0) numpy.around(a, decimals=0)	在 Rounding 的方法部分，與 Python 同樣採用 IEEE 754 規範，四捨、五取最近偶數、六入，而非我們一般講的四捨五入。 round 與 around 用法及結果相同
rint()	numpy.rint(a)	Round至最近的整數
trunc()	numpy.trunc(a)	無條件捨去小數點
floor()	numpy.floor(a)	向下取整數
ceil()	numpy.ceil(a)	向上取整數
fix()	numpy.ceil(a)	向0的方向取整數

NumPy 陣列運算 – 取絕對值：np.abs()，np.absolute()，np.fabs()

- np.abs() 是 np.absolute() 的簡寫，兩者完全相同；np.fabs() 的差異在於無法處理複數 (Complex)。
- 如果傳入複數至 fabs() 的話則會產生錯誤

NumPy 陣列運算 – 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。
- 點積運算示意圖：

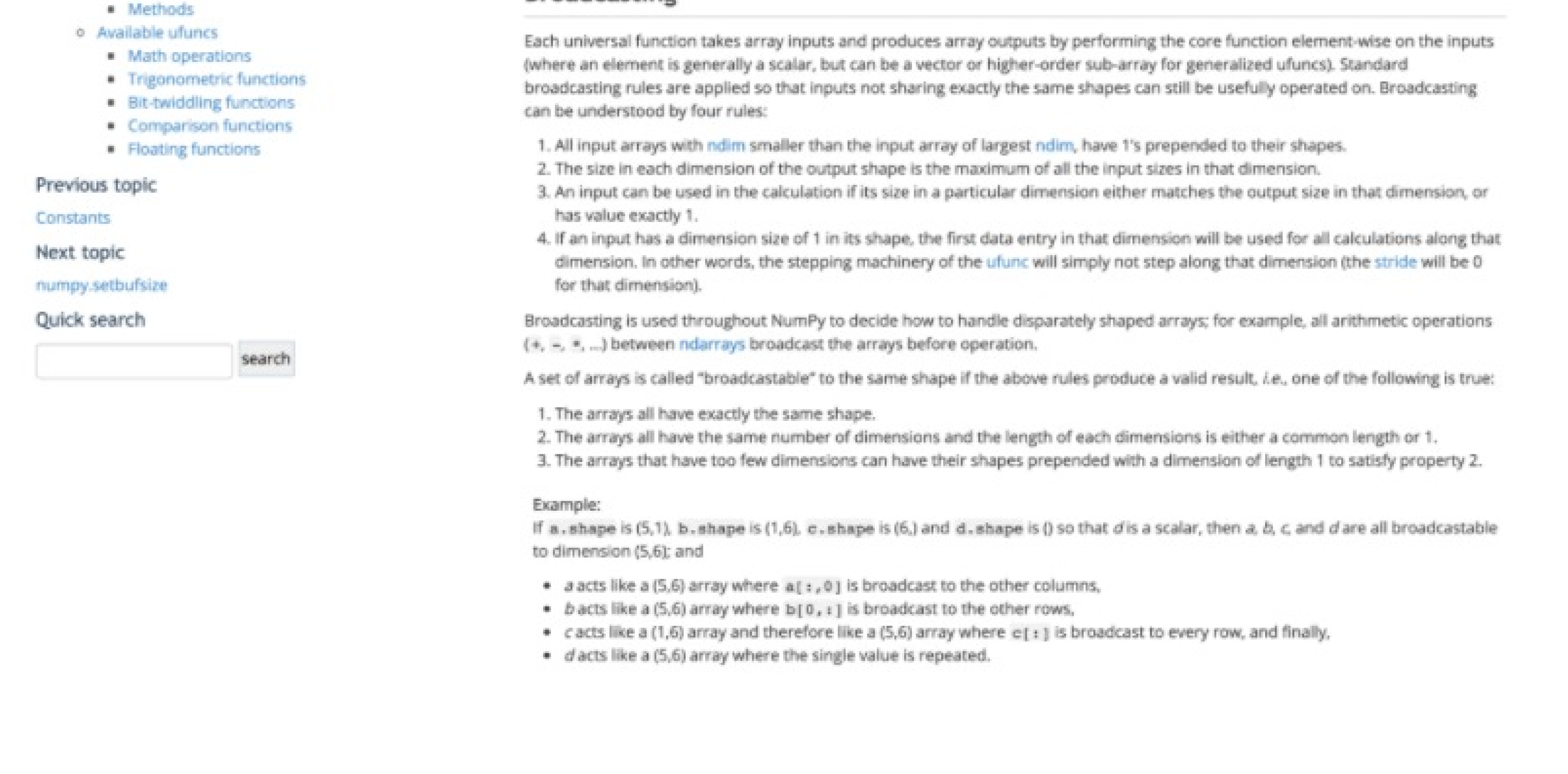
$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \end{bmatrix}$$

$$A \cdot B = \begin{bmatrix} a_{11}*b_{11}+a_{12}*b_{21}+a_{13}*b_{31} & a_{11}*b_{12}+a_{12}*b_{22}+a_{13}*b_{32} & a_{11}*b_{13}+a_{12}*b_{23}+a_{13}*b_{33} & a_{11}*b_{14}+a_{12}*b_{24}+a_{13}*b_{34} \\ a_{21}*b_{11}+a_{22}*b_{21}+a_{23}*b_{31} & a_{21}*b_{12}+a_{22}*b_{22}+a_{23}*b_{32} & a_{21}*b_{13}+a_{22}*b_{23}+a_{23}*b_{33} & a_{21}*b_{14}+a_{22}*b_{24}+a_{23}*b_{34} \end{bmatrix}$$

延伸閱讀

ufunc的清單及文件
網站：[Universal functions (ufunc)]



NumPy

NumPy.org | Docs | NumPy v1.19 Manual | NumPy Reference

Table of Contents
- Universal functions (ufunc)
  - Broadcasting
  - Output type determination
  - Use of internal buffers
  - Error handling
  - Casting Rules
  - Overriding Ufunc behavior
  - ufunc
    - Optional keyword arguments
    - Attributes
    - Methods
  - Available ufuncs
    - Math operations
    - Trigonometric functions
    - Bit-twiddling functions
    - Comparison functions
    - Floating functions

Previous topic
Constants
Next topic
numpy.setbufsize
Quick search
search

Universal functions (ufunc)

A universal function (or ufunc for short) is a function that operates on ndarrays in an element-by-element fashion, supporting array broadcasting, type casting, and several other standard features. That is, a ufunc is a "vectorized" wrapper for a function that takes a fixed number of specific inputs and produces a fixed number of specific outputs.

In NumPy, universal functions are instances of the numpy.ufunc class. Many of the built-in functions are implemented in compiled C code. The basic ufuncs operate on scalars, but there is also a generalized kind for which the basic elements are sub-arrays (vectors, matrices, etc.), and broadcasting is done over other dimensions. One can also produce custom ufunc instances using the frompyfunc factory function.

Broadcasting

Each universal function takes array inputs and produces array outputs by performing the core function element-wise on the inputs (where an element is generally a scalar, but can be a vector or higher-order sub-array for generalized ufuncs). Standard broadcasting rules are applied so that inputs not sharing exactly the same shapes can still be usefully operated on. Broadcasting can be understood by four rules:

1. All input arrays with ndim smaller than the input array of largest ndim, have 1's prepended to their shapes.
2. The size in each dimension of the output shape is the maximum of all the input sizes in that dimension.
3. An input can be used in the calculation if its size in a particular dimension either matches the output size in that dimension, or has value exactly 1.
4. If an input has a dimension size of 1 in its shape, the first data entry in that dimension will be used for all calculations along that dimension. In other words, the stepping machinery of the ufunc will simply not step along that dimension (the stride will be 0 for that dimension).

Broadcasting is used throughout NumPy to decide how to handle disparately shaped arrays; for example, all arithmetic operations (+, -, *, ...) between ndarrays broadcast the arrays before operation.

A set of arrays is called "broadcastable" to the same shape if the above rules produce a valid result, *i.e.*, one of the following is true:

1. The arrays all have exactly the same shape.
2. The arrays all have the same number of dimensions and the length of each dimensions is either a common length or 1.
3. The arrays that have too few dimensions can have their shapes prepended with a dimension of length 1 to satisfy property 2.

Example:
If `a.shape` is (5,1), `b.shape` is (1,6), `c.shape` is (6,) and `d.shape` is () so that *d* is a scalar, then *a*, *b*, *c*, and *d* are all broadcastable to dimension (5,6); and

- *a* acts like a (5,6) array where `a[:,0]` is broadcast to the other columns,
- *b* acts like a (5,6) array where `b[0,:]` is broadcast to the other rows,
- *c* acts like a (1,6) array and therefore like a (5,6) array where `c[:]` is broadcast to every row, and finally,
- *d* acts like a (5,6) array where the single value is repeated.

下一步：閱讀範例與完成作業