

Chapter 7

數學陣列與矩陣的擴充程式庫NumPy

Introduction to NumPy

~~※本教材不是以簡報為唯一目的，視為講義會更好！
為了方便初學者把握住重點，快速學習。

所以會補充較多字體較小的相關說明，請細細品嚐~~

Instructor: Jiun-Ting Jiang

Email: rjrj0510@gmail.com

上課，儘量往前座哦！

*Dept. of Computer Science and Information Engineering,
Tamkang University*

Outline

向量與矩陣運算的好用套件--Numpy

- Numpy套件是什麼？
- 7-1: NumPy的核心功能就是多維陣列
- 7-2: numpy 快速測試
- 7-3: numpy重要資料結構ndarray內的關鍵屬性
- 7-4:產生陣列的方法(Creating Arrays)
- 7-5:陣列的索引(index) 和 切片(slice)
- 7-6:矩陣進階操作與指定軸(axis)的用法
- 7-7:創建自己的陣列函數-計算移動平均線 (Moving Average)

Numpy套件是什麼？

- 簡言之，就是「矩陣數學運算」的好用套件。
 - NumPy是數學運算套件，提供陣列 (ndarray)、矩陣 (matrix)... 數學運算，支援高維度運算，方便數學矩陣運算。
 - 速度比Python內建陣列快很多。
- 為什麼快？
 - 因為底層是用C和Fortran的編寫來實現。在NumPy上只要能被表示為「針對陣列或矩陣運算」的演算法，其執行效率幾乎都可以與編譯過的等效C語言程式碼一樣快。
 - 因為NumPy具平行處理能力。可解決python速度不夠快的問題
- NumPy的重要性：
 - 是多數重量級Python數據處理庫的重要基礎模塊，例如：
 - 資料分析程式的Pandas，科學計算的SciPy，機器學習的Scikit-learn ... 等。都是以NumPy的多維度陣列作為重要資料物件。

7-1: NumPy的核心功能就是多維陣列

- NumPy的**關鍵資料結構是ndarray** (*n-dimensional array*):
 - 是一個「**多維度、同質**」並且固定大小的陣列物件。
- 容易使用：就像在Python 執行int或float中一樣。

如下所示：兩個陣列相加，就像兩個整數相加一樣，容易且迅速：

```
In [3]: 1 a=123  
        2 a+a
```

```
Out[3]: 246
```

```
In [4]: 1 import numpy as np    ##※依慣例都是 稱numpy為 np. 注意：為何不可 in  
        2 a = np.array([1, 2, 3])  
        3 a + a  
        4 # output array([2, 4, 6])
```

```
Out[4]: array([2, 4, 6])
```

容易使用！
就像整數在運算

用Numpy的 `np.max()` 函數比用python內建`max()`快100倍

■ 執行速度快！

問題：對於同一個陣列(100萬個元素)，要找出陣列裡的最大元素

```
: 1 a = np.random.rand(1000000)
```

```
: 1 timeit -n 10 max(a)
```

88.4 ms \pm 1.95 ms per loop (mean \pm std. dev. of 7 runs, 10 loops each)

```
: 1 timeit -n 10 a.max()
```

657 μ s \pm 154 μ s per loop (mean \pm std. dev. of 7 runs, 10 loops each)

```
: 1 max(a) == a.max()
```

```
: True
```

執行速度快！
本實驗快約100倍！

用內建 `max` 86ms, 而改用numpy的 `a.max()`則 只 708 us ,

以此題目為例： 📌

用Numpy的 `np.max()` 函數比用python內建`max()`，可以快大約100倍。

Exercise 7-1

- **Ex7-1Add1:** NumPy套件(a)關鍵資料結構是什麼？有何特點？ (b)Numpy套件可提供的主要優點是？
- **Ex7-1Add2:** 要引入NumPy的擴充程式庫，(a)慣例寫法是什麼？(b)是否可以直接寫 `import numpy` 或 `from numpy import *`，若這樣引用，可能分別會有什麼問題呢？
- **Ex7-1Add3:** python原本有怎樣的問題，會因為加入NumPy低階程式庫而改善？

7-2: numpy 快速測試

產生陣列的基本語法

```
In [9]: 1 import numpy as np
        2 x = np.array([1, 2, 3])
        3 x
```

```
Out[9]: array([1, 2, 3])
```

```
In [10]: 1 y = np.arange(10) # np.Array RANGE , 類似 Py
        2 y
```

```
Out[10]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

陣列基本運算

```
In [11]: 1 a = np.array([1, 2, 3, 6])
        2 b = np.linspace(0, 2, 4) # 建立一個array, 在x
        3 b
```

```
Out[11]: array([0.          , 0.66666667, 1.33333333, 2.        ])
```

```
In [12]: 1 c = a - b
        2 c
```

```
Out[12]: array([1.          , 1.33333333, 1.66666667, 4.        ])
```

```
In [13]: 1 a**2 # [1,2,3,6]的平方
```

```
Out[13]: array([ 1,  4,  9, 36], dtype=int32)
```

好用的np全域函數

```
In [14]: 1 a = np.linspace(-np.pi, np.pi, 20) ## np.pi就是π, (在+-pi之間, 採樣20個)
        2 a
```

```
Out[14]: array([-3.14159265, -2.81089869, -2.48020473, -2.14951076, -1.8188168 ,
               -1.48812284, -1.15742887, -0.82673491, -0.49604095, -0.16534698,
                0.16534698,  0.49604095,  0.82673491,  1.15742887,  1.48812284,
                1.8188168 ,  2.14951076,  2.48020473,  2.81089869,  3.14159265])
```

```
In [15]: 1 b = np.sin(a)
        2 c = np.cos(a)
        3 b
```

```
Out[15]: array([-1.22464680e-16, -3.24699469e-01, -6.14212713e-01, -8.37166478e-01,
               -9.69400266e-01, -9.96584493e-01, -9.15773327e-01, -7.35723911e-01,
               -4.75947393e-01, -1.64594590e-01,  1.64594590e-01,  4.75947393e-01,
                7.35723911e-01,  9.15773327e-01,  9.96584493e-01,  9.69400266e-01,
                8.37166478e-01,  6.14212713e-01,  3.24699469e-01,  1.22464680e-16])
```

線性代數的運算

```
In [16]: 1 from numpy.random import rand
        2 from numpy.linalg import solve, inv
        3 a = np.array([[1, 2, 3], [3, 4, 6.7], [5, 9.0, 5]])
        4 a.transpose() # 矩陣 轉置運算
        5 # array([[ 1. ,  3. ,  5. ],
        6 #         [ 2. ,  4. ,  9. ],
        7 #         [ 3. ,  6.7,  5. ]])
```

```
Out[16]: array([[1. ,  3. ,  5. ],
               [2. ,  4. ,  9. ],
               [3. ,  6.7,  5. ]])
```

Exercise 7-2

- **Ex7-2Add1:numpy**除了有陣列基本運算，也增加很多全域使用的函數？又有那些線性代數的運算 (linear algebra)?請簡單舉例說明即可。

7-3: numpy重要資料結構ndarray內的關鍵屬性

ndarray的關鍵是 維度(ndim, dimension), 形狀(shape, 維度是幾乘幾), 數值型態(dtype, data type)

```
In [22]: 1 a64 = np.array([1, 2, 3, 4, 5, 6, 7, 8],dtype=np.int64) ##直接指定 元素的資料型態為 int64
          2 a64
```

```
Out[22]: array([1, 2, 3, 4, 5, 6, 7, 8], dtype=int64)
```

```
In [23]: 1 a32 = np.array([1, 2, 3, 4, 5, 6, 7, 8],dtype=np.int32) ##直接指定 元素的資料型態為 int32. 則省略顯示
          2 a32
```

```
Out[23]: array([1, 2, 3, 4, 5, 6, 7, 8])
```

```
In [24]: 1 a = np.array([1, 2, 3, 4, 5, 6, 7, 8])
          2 a
```

```
Out[24]: array([1, 2, 3, 4, 5, 6, 7, 8])
```

```
In [25]: 1 a.dtype
```

```
Out[25]: dtype('int32')
```

```
In [26]: 1 a.shape
```

```
Out[26]: (8,)
```

```
In [27]: 1 a.ndim
```

```
Out[27]: 1
```

```
In [28]: 1 ##印ndarray 的三個關鍵屬性。 「維度 ndim」 「矩陣形狀 shape」 「資料型態 dtype」
          2 def print_array_details(a):
          3     print('維度(ndim): %d, 矩陣形狀(shape): %s, 資料型態(dtype): %s' %(a.ndim, a.shape, a.dtype))
```

```
In [29]: 1 ### !!!!!!!!!!!!!!! dtype 竟是 int32 因為 是 32位元哦!
          2 print_array_details(a)
```

```
維度(ndim): 1, 矩陣形狀(shape): (8,), 資料型態(dtype): int32
```

Reshape:改變陣列形狀(調整各維度)

```
In [35]: 1 a = np.array([1, 2, 3, 4, 5, 6, 7, 8])
          2 a = a.reshape([2, 4])
          3 a
```

```
Out[35]: array([[1, 2, 3, 4],
                [5, 6, 7, 8]])
```

```
In [36]: 1 print_array_details(a)
          2 #Dimensions: 2, shape: (2, 4), dtype: int64
```

維度(ndim): 2, 矩陣形狀(shape): (2, 4), 資料型態(dtype): int32

```
1 An eight-member array can also be reshaped into a three-dimensional array:
```

```
In [37]: 1 a = a.reshape([2, 2, 2])
          2 a
          3
```

```
Out[37]: array([[[1, 2],
                 [3, 4]],
                [[5, 6],
                 [7, 8]]])
```

```
In [38]: 1 print_array_details(a)
          2 #Dimensions: 3, shape: (2, 2, 2), dtype: int64
```

維度(ndim): 3, 矩陣形狀(shape): (2, 2, 2), 資料型態(dtype): int32

Exercise 7-3

- Ex7-3Add1:numpy的重要資料結構是ndarray, 裡面有那些關鍵屬性？

7-4:產生陣列的方法(Creating Arrays)

- 創建Numpy陣列的主要三類方法
 - 第一類產生法：轉換法，用函數`np.array`將Python的列表(list)直接轉換
 - 第二類產生法：方便的內置函數產生法,如 `zeros()`, `ones()`, `arange()`, ... 等,特別的也包含：`linspace()` (線性空間取樣法)。
 - 第三類產生法：隨機庫函數法 (※`numpy.random`裡的函數)
- NumPy對於現成的陣列，可以用`reshape()`調整形狀。
- NumPy也可直接產生特定形狀的陣列，例如： 零(`Zeros`)和壹(`ones`)是最常用的函數，用於創建並且會預填數值的陣列。
 - 這些方法的默認`dtype`是64位浮點數 (`float64`)：

零(Zeros),壹(ones),陣列範圍(arange)是最常用的產生陣列函數:

```
In [42]: 1 a = np.zeros([2,3])
```

```
Out[42]: array([[0., 0., 0.],
               [0., 0., 0.]])
```

```
In [43]: 1 a.dtype
          2 #Out[34]: dtype('float64')
          3
```

```
Out[43]: dtype('float64')
```

```
In [44]: 1 np.ones([2, 3])
          2 # Out[35]:
```

```
Out[44]: array([[1., 1., 1.],
               [1., 1., 1.]])
```

好用又常見的組合拳產生法

```
1 import numpy as np
2 np.arange(12).reshape(3,4)

array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
1 創建指定形狀，但無預設值的空陣
2 更快的產生方法empty()，只配置
```

```
In [45]: 1 empty_array = np.empty((2,3))
          2 empty_array
          3 # Out[3]:
          4 # array([[ 6.93185732e-310,
          5 #          [ 2.38085057e-316,
```

```
Out[45]: array([[1., 1., 1.],
               [1., 1., 1.]])
```

```
1 創建指定形狀的隨機陣列(random)
2 在NumPy的random模塊中找到同樣
```

```
In [46]: 1 np.random.random((2,3))
          2 # >>> Out:
          3 # array([[ 0.97519667,  0.9
          4 #          [ 0.10407003,  0.3
```

```
Out[46]: array([[0.20195972, 0.19926877,
               [0.27966589, 0.26425093,
```

```
1 A 2x3 array of random numbers within the range 0 <= x < 1.
```

```
1 方便的線性空間(linspace creates)在指定的間隔內創建「指定數量的均勻間隔」的「樣本數量」。
2 與arange相似，但是有一個參數是明顯不同：設定「樣本數」而不是「步長step」。
3 ※註：range()不含end，而linspace會包含end的值！
```

```
1 np.linspace(2, 10, 5) # 5 numbers in range 2-10
2 #Out: array([2., 4., 6., 8., 10.])
3
```

```
Out[47]: array([ 2.,  4.,  6.,  8., 10.]])
```

```
In [49]: 1 a = np.ones([3,5])
          2 a
          3 # Out[33]:
          4 # array([[ 0.,  0.,  0.],
          5 #          [ 0.,  0.,  0.]])
          6
          7
```

```
Out[49]: array([[1., 1., 1., 1., 1.],
               [1., 1., 1., 1., 1.],
               [1., 1., 1., 1., 1.]])
```

```
In [50]: 1 a[:,1:4]=np.nan ##(index是由0開始。end不
          2 a
```

```
Out[50]: array([[ 1., nan, nan, nan,  1.],
               [ 1., nan, nan, nan,  1.],
               [ 1., nan, nan, nan,  1.]])
```

```
In [51]: 1 a = np.ones([2,3])
          2 a
```

```
Out[51]: array([[1., 1., 1.],
               [1., 1., 1.]])
```

```
1
2 ※注意：請注意，linspace與arange不同，因為
```

Exercise 7-4

- Ex7-4Add1:創建Numpy數組的主要三種方法
- Ex7-4Add2:請用陣列範圍(`arange`)，產生18個資料(0~17)而且`shape`是`3*6`，請寫出這段可產生Numpy數組的程式。

7-5:陣列的索引(index) 和 切片(slice)

7-5 陣列的索引(index) 和 列表的切片(slice)

一維陣列的索引(index)和切片(slice)就與Python列表用法一樣：

```
1 a = np.array([1, 2, 3, 4, 5, 6])
2 a[2] # Out: 3
3
```

```
1 a[3:5] # Out: array([4, 5])
```

array([4, 5])

可以透過切片的區域，來調整矩陣原來的值。(※要先確定會選到的位置哦！)

```
1 a = np.ones([4,6])
2 a
```

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

```
1 a[:4:2] = 3 #只寫第1維,表示其他全部都選到。就是a[:4:2, :]的寫法
2 a
```

```
array([[3., 3., 3., 3., 3., 3.],
       [1., 1., 1., 1., 1., 1.],
       [3., 3., 3., 3., 3., 3.],
       [1., 1., 1., 1., 1., 1.]])
```

```
1 # a = np.ones([4,6])
2 # a[:4:2] = 3
3 a[:,1:4]=np.nan
4 a
```

```
array([[ 3., nan, nan, nan,  3.,  3.],
       [ 1., nan, nan, nan,  1.,  1.],
       [ 3., nan, nan, nan,  3.,  3.],
       [ 1., nan, nan, nan,  1.,  1.]])
```

reversed反轉法

```
1 #reversed反轉法
2 a = np.arange(1,7) #此時是 [1,2,3,4,5,6]
3 a[::-1] # 反轉輸出的slice方法:Out: array([6, 5, 4, 3, 2, 1]),
```

array([6, 5, 4, 3, 2, 1])

多維陣列的索引方法，切片(slice)的方法：

多維陣列與一維陣列的索引方法相似。只是每個維都有其自己的「索引/切片」操作，這些操作可以在「以逗號分隔的元組(tuple)」中指定。

多維陣列的索引方法，切片(slice)的方法

- 用「逗號,」區分不同的維數軸。
- 「...」與「:」，
- 「省略號...」與「冒號:」，都是可用的符號！
- 冒號: 的操作 是**slice**對每個維度軸的範圍指定方法，與**list**的**slice**操作方式相同，就是 **start: end:step**
 - 注意1:其中的 **start:end :step** 表示是 **start:end**, 由0開始，不包含**end**, 若只一個冒號表示範圍是「全部選擇(:)」
 - 注意2:如果在冒號指定資料數少於維數軸時，則假定其餘沒有指定的維，都視為「全部選擇(:)」。
- 「省略號...」的操作 是表示省略多個軸的指定，代表這些省略的維，都視為「全部選擇(:)」。
 - 注意3: 只想直接選最後一個維的軸，就適合用「省略號...」
- 實驗在下一頁：

先看(8)個資料選到的範圍。
再看16個資料選到的範圍

```
1 a = np.arange(8) #[0,1,2,3,4,5,6,7]
2 a.shape = (2, 2, 2)
3 a
```

```
array([[[0, 1],
        [2, 3]],

       [[4, 5],
        [6, 7]]])
```

```
1 a[1]
```

```
array([[4, 5],
       [6, 7]])
```

```
1 a[1,:]
```

```
array([[4, 5],
       [6, 7]])
```

```
1 a[1] == a[1,:] ## 注意：沒寫的維，表示省略，就視為全部選！
```

```
array([[ True,  True],
       [ True,  True]])
```

```
1 a[1] == a[1,:,:] ## 注意：沒寫的維，表示省略，就視為全部選！
```

```
array([[ True,  True],
       [ True,  True]])
```

```
1 a[1] == a[1,...] ## 註：「...」 「省略號...」 表示省略多個軸的指定
```

```
array([[ True,  True],
       [ True,  True]])
```

a[0]

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

a[1, 0]

a[0,:,1:3]

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

a[1, 1, :-1]

```
1 a[...,0] == a[:, :, 0]
```

```
array([[ True,  True],
       [ True,  True]])
```

```
1 import numpy as np
2 a=np.arange(16)
3 a=a.reshape([2,2,4])
4 a
```

```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7]],

       [[ 8,  9, 10, 11],
        [12, 13, 14, 15]])
```

```
1 a[0]
```

```
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

```
1 a[1,0]
```

```
array([ 8,  9, 10, 11])
```

```
1 a[0, :, 1:3]
```

```
array([[1, 2],
       [5, 6]])
```

```
1 a[1,1, :-1]
```

```
array([12, 13, 14])
```

Exercise 7-5

- **Ex7-5Add1:** 如果numpy有以下三個指令，請問執行後的矩陣結果是？
 - ❑ `a = np.ones([5,7])`
 - ❑ `a[:5:3] = 4`
 - ❑ `a[:,2:5]=np.nan`
- **Ex7-5Add2:** Numpy的的slice操作時，「省略號...」與「冒號:」，都是可用又重要的切片操作，請問分別代表什麼效果？又有那些要注意的地方？
- **Ex7-5Add3:** 若有以下指令，請問執行後的結果，u,v,w,x分別是什麼？
 - ❑ `a=np.arange(24)`
 - ❑ `a=a.reshape([2,3,4])`
 - ❑ `u=a[1]`
 - ❑ `v=a[1,0]`
 - ❑ `w=a[0, :, 1:3]`
 - ❑ `x=a[1,1, :-1]`
 - ❑ `print("u=",u, "\nv=",v, "\nw=\n",w,"\nx=",x,)`
- **● 【check1】 Ex7-5Add3_切片(slice)的運算實驗.**

7-6:矩陣進階操作與指定軸(axis)的用法

7-6 一些基本操作，與「軸(axis)」參數的指定

■ 執行Numpy陣列的基本操作，很容易，就像「基本數學運算」。

■ 以下是在二維陣列上使用一些重載算術運算符測試

■ 簡單的數學運算將應用於陣列的所有成員。

■ 請注意，在將陣列除以浮點值（2.0）的情況下，結果，會自動轉換為浮點類型（float64）。

■ 因為就像簡單數字一樣容易操作陣列，是NumPy的一大優點。

```
1 執行Numpy陣列的基本操作，很容易，就像「基本數學運算」。  
2  
3 以下是，在二維陣列上使用一些重載算術運算符的測試：  
4  
5 簡單的數學運算將應用於陣列的所有成員。  
6 請注意，在將陣列除以浮點值（2.0）的情況下，結果，會自動轉換為浮點類型（float64）  
7 因為就像簡單數字一樣容易操作陣列，是NumPy的一大優點。
```

```
1 a=np.array([1,2,3,4,5,6])  
2 a=a.reshape([2,3])  
3 a
```

```
array([[1, 2, 3],  
       [4, 5, 6]])
```

```
1 a*2
```

```
array([[ 2,  4,  6],  
       [ 8, 10, 12]])
```

```
1 a~2
```

```
array([[ -1,  0,  1],  
       [ 2,  3,  4]])
```

```
1 a/2.0 #陣列除以浮點值（2.0）的情況下，結果，會自動轉換為浮點類型（float64）
```

```
array([[0.5, 1. , 1.5],  
       [2. , 2.5, 3. ]])
```

```
1 a=np.array([1,2,3,4,5,6]).reshape([2,3])  
2 z=(a*2-2)/2.0  
3 print(z)  
4 # rray([[0., 1., 2.],  
5 #       [3., 4., 5.]])
```

布林邏輯運算符的工作方式與算術運算符相似。

布林邏輯運算符的工作方式與算術運算符相似。
就像pandas, 是很常用的方法。

```
1 a = np.array([45, 65, 76, 32, 99, 22])
2 a < 50
```

```
array([ True, False, False,  True, False,  True])
```

```
1 #三角函數的測試
2 # Trigonometric functions
3 pi = np.pi
4 a = np.array([pi, pi/2, pi/4, pi/6])
5 np.degrees(a) # radians to degrees
6 # Out: array([ 180.,  90.,  45.,  30.,])
```

```
array([180.,  90.,  45.,  30.])
```

```
1 sin_a = np.sin(a)  #※注意: sin(180度)=0 , 但是顯示的是: 1.22464680e-16 , 可發現會有一些小誤差!
2 sin_a
```

```
array([1.22464680e-16, 1.00000000e+00, 7.07106781e-01, 5.00000000e-01])
```

```
1 # Rounding
2 np.round(sin_a, 7) #np的 round() 可以四捨五入到小數第7位, 可發現sin(180度)就是0.
3 # Out: array([ 0.,  1.,  0.7071068,  0.5 ])
```

```
array([0.          , 1.          , 0.7071068, 0.5       ])
```

```
1
```

矩陣進階用法(指定axis)

矩陣還有一些較進階的用法(可以指定特定的軸axis)

以使用整數對軸進行索引，例如0用於列，1用於行，

```
1 a = np.arange(24).reshape((4,2,3)) # 建立矩陣，立刻調整形狀 reshape (Y軸2 ,
2 a
```

```
array([[[ 0,  1,  2],
        [ 3,  4,  5]],

       [[ 6,  7,  8],
        [ 9, 10, 11]],

       [[12, 13, 14],
        [15, 16, 17]],

       [[18, 19, 20],
        [21, 22, 23]]])
```

```
1 #因為 shape(4,2,3)，所以第1個軸 axis=0，會有4個元素，axis=1會有2個元素，axis
```

```
1 #axis=0 號,第1個軸 shape是4，表示本軸有4個元素
2 #，加總後，會有2*3的矩陣，第1個，第2個 元素是 0+6+12+18=36, 1+7+13+19=40
3 a.sum(axis=0)
```

```
array([[36, 40, 44],
       [48, 52, 56]])
```

```
1 #axis=1 號,第3個軸 shape是2，表示本軸有2個元素
2 #，加總後，會有4*3的矩陣，第1個元素是 0+3 = 3 , 1+4=5
3 a.sum(axis=1)
```

```
array([[ 3,  5,  7],
       [15, 17, 19],
       [27, 29, 31],
       [39, 41, 43]])
```

```
1 #axis=2 號,第3個軸 shape是3，表示本軸有3個元素
2 #，加總後，會有4*2的矩陣，第1個元素是 0+1+3 = 3 , 3+4+5 = 12
3 a.sum(axis=2)
```

```
array([[ 3, 12],
       [21, 30],
       [39, 48],
       [57, 66]])
```

```
1 #axis=2 號,第3個軸 shape是3，表示本軸有3個元素
2 #，加總後，會有4*2的矩陣，第1個元素是 0+1+3 = 3 , 3+4+5 = 12
3 a.sum(axis=2)
```

```
array([[ 3, 12],
       [21, 30],
       [39, 48],
       [57, 66]])
```

```
1 a.min(axis=2) #把sum() 改為 min()
```

```
array([[ 0,  3],
       [ 6,  9],
       [12, 15],
       [18, 21]])
```

```
1 print(a.min()) #無參數的a.min()，所有中的最小值
2 print("-"*10)
3 print(a.min(axis=0)) #註: axis=0; 每行的最小值 axis= 可省略
4 print("-"*10)
5 print(a.min(1)) # axis=1; 每列的最小值
6 print("-"*10)
7 print(a.min(2)) # axis=2; 每列的最小值
```

```
0
-----
[[0 1 2]
 [3 4 5]]
-----
[[ 0  1  2]
 [ 6  7  8]
 [12 13 14]
 [18 19 20]]
```

```

1 a = np.arange(8).reshape((2,4)) # 建立矩陣，立刻調整形狀 reshape (Y軸2 ,X軸4)
2 a
3 # array([[0, 1, 2, 3],
4 #        [4, 5, 6, 7]])

```

```

array([[0, 1, 2, 3],
       [4, 5, 6, 7]])

```

```

1 print(a.min()) #無參數的a.min()，所有中的最小值
2 print(a.min(axis=0)) #註：axis= 是可以省略，
3 print(a.min(0)) # axis=0; 每行的最小值
4 print(a.min(1)) # axis=1; 每列的最小值

```

```

0
[0 1 2 3]
[0 1 2 3]
[0 4]

```

```

1 a.min(axis=0)

```

```

array([0, 1, 2, 3])

```

```

1 a.sum(axis=0) # [0+4, 1+5, 2+6, 3+7]
2 # array([4, 6, 8, 10])

```

```

array([ 4,  6,  8, 10])

```

```

1 a.min(axis=1) #直接指定某個維數軸的方法，同時得到
2 # array([0, 4])

```

```

array([0, 4])

```

```

1 a.mean(axis=1) #可以沿著第二軸取平均值。
2 # array([ 1.5,  5.5 ]) [0,1,2,3]的平均是1.5,

```

```

array([1.5, 5.5])

```

```

1 a.std(axis=1) # [0、1、2、3]，...的標準偏差
2 # array([ 1.11803399,  1.11803399])

```

```

array([1.11803399, 1.11803399])

```

```

1 a = np.arange(8).reshape((2,4))
2 a

```

```

1 a = np.arange(8).reshape((2,4))
2 a
3 # array([[0, 1, 2, 3],
4 #        [4, 5, 6, 7]])

```

```

array([[0, 1, 2, 3],
       [4, 5, 6, 7]])

```

```

1 np.cumsum(a, axis=1) # cumulative sum, 累積總和 在 1號軸 axis=1; 每列的累積
2 # array([[ 0,  1,  3,  6],    #[ 0, 0+1, 0+1+2, 0+1+2+3]
3 #        [ 4,  9, 15, 22]])  #[4, 4+5, 4+5+6, 4+5+6+7]

```

```

array([[ 0,  1,  3,  6],
       [ 4,  9, 15, 22]], dtype=int32)

```

```

1 np.cumsum(a) # 若省略軸的參數指定(axis)，那麼就視為是 一維陣列，全部累積。
2 # array([ 0,  1,  3,  6, 10, 15, 21, 28])

```

```

array([ 0,  1,  3,  6, 10, 15, 21, 28], dtype=int32)

```

Exercise 7-6

- Ex7-6Add1: Numpy的基本操作如下，結果為何？
 - `a=np.arange(6).reshape([2,3])`
 - `z=((a+1) *2-2)/2.0`
 - `print(z)`
- Ex7-6Add2: Numpy的基本操作，包括有三角函數。應該要注意一件什麼事？
- Ex7-6Add3: Numpy在進行矩陣運算時，是否可以指定要在某個軸索引？要如何指定？

7-7:創建自己的陣列函數-計算移動平均線 (Moving Average)

```
2 def moving_average(a, n=3):
3     ret = np.cumsum(a) #ret = np.cumsum(a, dtype=float)
4     ret[n:] = ret[n:] - ret[:-n]
5     return ret[n - 1:] / n
```

```
1 該函數接收指定移動窗口大小的陣列a和數字n。
2 先使用NumPy的內置方法來計算數組的累積和(np.cumsum, cumulative sum)。
```

```
1 #以下解釋 這個 moving_average的工作原理
```

```
1 #假設a是(0~5) 的 List
2 a = np.arange(6)
3 a
4 # array([0, 1, 2, 3, 4, 5])
```

```
array([0, 1, 2, 3, 4, 5])
```

```
1 csum = np.cumsum(a)
2 csum
3 # Out: array([0, 1, 3, 6, 10, 15])
```

```
array([ 0,  1,  3,  6, 10, 15], dtype=int32)
```

```
1 csum[3:]
```

```
array([ 6, 10, 15], dtype=int32)
```

```
1 csum[:-3]
```

```
array([0, 1, 3], dtype=int32)
```

```
1 從累積和數組的第n個索引開始，我們減去所有i的第i-n個值，這意味著i現在具有a的最後n個值的和。
2 以下是具有3個窗口的例子：
```

```
1 csum[3:] = csum[3:] - csum[:-3]
2 # csum = array([0, 1, 3, 6, 9, 12]) #※ 前三個不變(因為比n小無法計算，後面是 6-0, 10-2, 15-3)
3 # [0+1+2, 1+2+3, 2+3+4, 3+4+5]
4 csum
```

```
array([ 0,  1,  3,  6,  9, 12], dtype=int32)
```

```
1 csum[2:] / 3
```

```
array([1., 2., 3., 4.])
```

- 移動平均值是基於最近 **n** 個值的移動窗口的一系列平均值，其中 **n** 是可變的，也稱為移動平均值或滾動平均值。
- 陣列 **a** 與最終陣列 **csum** 進行比較，索引 **5** 現在會是窗口 **[3, 4, 5]** 的總和。
- 由於移動平均線僅對向前的索引 (**n-1**) 有意義，因此僅保留返回這些值，在最後除以窗口大小 **n**，就可得出平均值。

○

```
1 #所以執行moving的副程式 funtion
2 a = np.arange(6)
3 moving_average(a, 3)
```

```
: array([1., 2., 3., 4.])
```



```
1 #所以執行moving的副程式 funtion
2 a = np.arange(6)
3 moving_average(a, 3)
```

```
array([1., 2., 3., 4.])
```

1 陣列a與最終陣列csum進行比較，索引5現在會是 窗口[3, 4, 5]的總和。
2 由於移動平均線僅對向前的索引 (n-1) 有意義，因此僅保留返回這些值，在最後 除以窗口大小n，就得出平均值。

```
1 #改為 10個元素，n=4，則執行moving的結果是：
2 a = np.arange(10)
3 moving_average(a, 4)
4 #Out[98]: array([ 1.5,  2.5,  3.5,  4.5,  5.5,  6.5,  7.5])
```

```
array([1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5])
```

```
1 #與統計相關的 好用函數
2 x=np.arange(1,11)
3 x.mean() #取平均值：
4 x.sum() #取總和
5 x.min() #取最小值
6 x.max() #取最大跟
7 np.cumsum(x)# 這個比較特別，會回傳累積的數值：
8 ## std = sqrt(mean(abs(x - x.mean())**2)) #標準差
```

```
array([ 1,  3,  6, 10, 15, 21, 28, 36, 45, 55], dtype=int32)
```

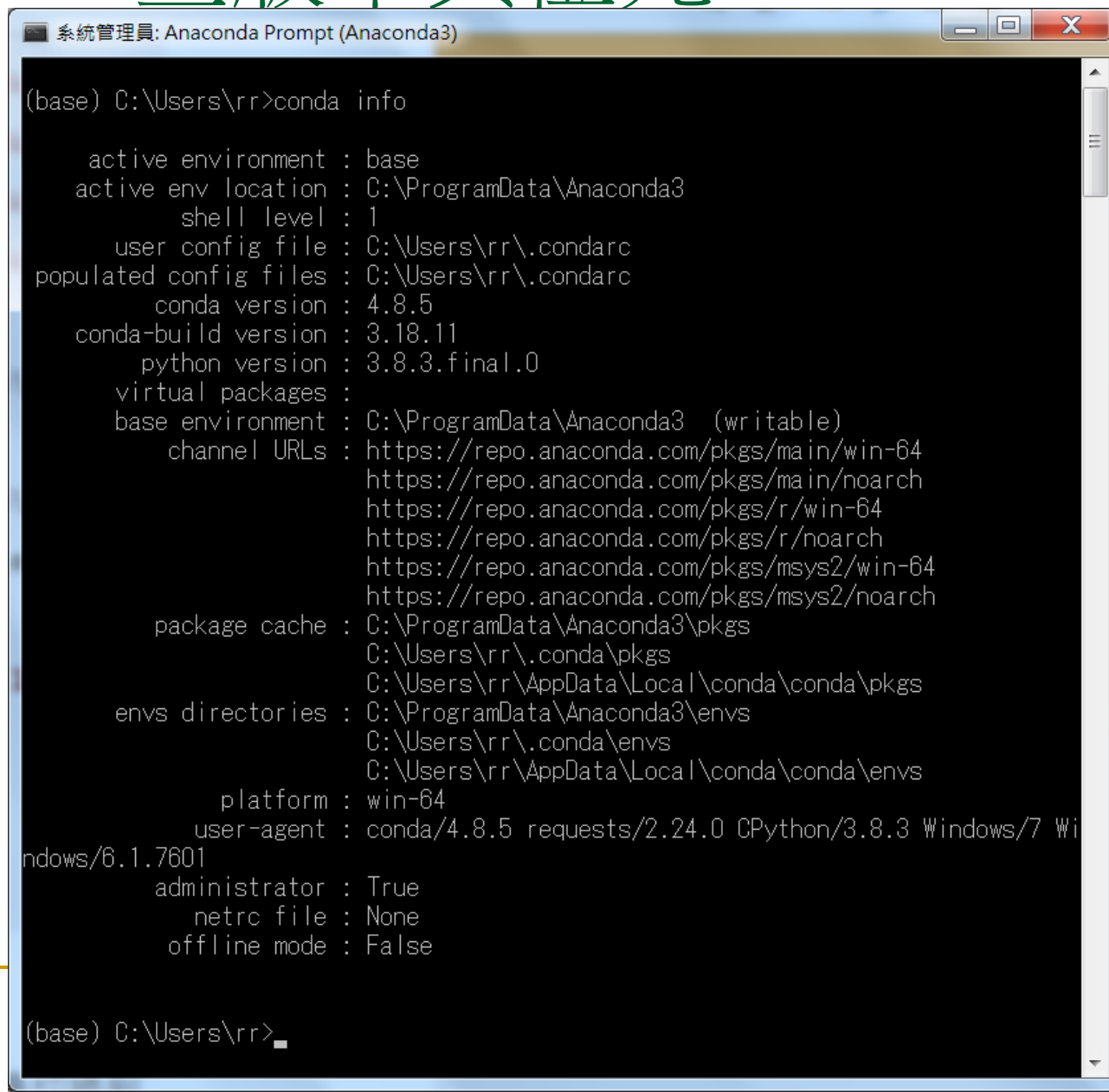
Exercise 7-7

- ● **【check2】** Ex7-7Add1: 移動平均值是基於最近n個值的移動窗口的一系列平均值，也稱為移動平均值或滾動平均值。
 - (a)請利用numpy設計這個函數。
 - (b1)如果`a = np.arange(6)`，而且`n=3`的時候，最後會得到的結果是？
 - (b2)另外，如果 `a = np.arange(10)`, 而且`n=4`，，最後會得到的結果是？
 - ※實作畫面的檢查重點：只要(b1)(b2)的最終結果即可。
- ● **【check3】** Ex7-7Add2: (Ai提示語) 作者：江俊廷，課程：「資料視覺化」要製作Jupyter Notebook教學檔，主題是:「移動平均(Moving Average)函數的製作」至少有四種常用實作方法：(A)純Python迴圈版本
(B) NumPy + `convolve` 捲積版(※捲積運算是深度學習cnn的計算方法)
(C) NumPy + `sliding_window_view` 版，這個輔助函數是「同時切出所有符合分段長度的全部片段」 (D) NumPy + `cumsum` 「累積和差分法」
請實作後用簡單初值`[1, 2, 3, 4, 5, 6, 7]`，視窗大小(`window`) = 3測試。
最後，把這四種版本配合相關說明。放入一個 Jupyter Notebook 教學檔，包括：四種方法程式碼、實際運算結果對照效能比較圖(使用 `%timeit`)、可視化示意圖(折線圖)，超過1000000的大量隨機測資進行效能測試的比較圖。最後，整理成一個ipynb的JSON文字檔。

-
- Thank for your attention.

- Ref:
- <https://nbviewer.jupyter.org/github/phelps-sg/python-bigdata/blob/master/src/main/ipynb/numerical-slides.ipynb>
- 數值計算在**Python**：基本數字型態與**NumPy**
- **Numpy performance tricks**
- <https://nbviewer.jupyter.org/gist/rossant/4645217>
- 當資料超大時，怎樣可以跑的更快，如何避改會變慢的情況。

#Conda info查版本與位元



```
系統管理員: Anaconda Prompt (Anaconda3)

(base) C:\Users\rr>conda info

      active environment : base
      active env location : C:\ProgramData\Anaconda3
            shell level : 1
       user config file : C:\Users\rr\.condarc
 populated config files : C:\Users\rr\.condarc
         conda version : 4.8.5
    conda-build version : 3.18.11
         python version : 3.8.3.final.0
    virtual packages :
      base environment : C:\ProgramData\Anaconda3 (writable)
         channel URLs : https://repo.anaconda.com/pkgs/main/win-64
                       https://repo.anaconda.com/pkgs/main/noarch
                       https://repo.anaconda.com/pkgs/r/win-64
                       https://repo.anaconda.com/pkgs/r/noarch
                       https://repo.anaconda.com/pkgs/msys2/win-64
                       https://repo.anaconda.com/pkgs/msys2/noarch
        package cache : C:\ProgramData\Anaconda3\pkgs
                       C:\Users\rr\.conda\pkgs
                       C:\Users\rr\AppData\Local\conda\conda\pkgs
       envs directories : C:\ProgramData\Anaconda3\envs
                       C:\Users\rr\.conda\envs
                       C:\Users\rr\AppData\Local\conda\conda\envs
            platform : win-64
         user-agent : conda/4.8.5 requests/2.24.0 CPython/3.8.3 Windows/7 Wi
ndows/6.1.7601
       administrator : True
          netrc file : None
        offline mode : False

(base) C:\Users\rr>_
```

```
系統管理員: Anaconda Prompt (Anaconda3)

vc >=14.1,<15.0a0

numpy 1.17.0 py36h19fb1c0_0
-----
file name      : numpy-1.17.0-py36h19fb1c0_0.tar.bz2
name           : numpy
version        : 1.17.0
build string   : py36h19fb1c0_0
build number   : 0
channel        : https://repo.anaconda.com/pkgs/r/win-64
size           : 25 KB
arch           : None
constrains     : ()
license        : BSD 3-Clause
md5            : e87a986f9d196755f86b1ebf3ef88c42
platform       : None
sha256         : 4b665b415dfd7b8f78ef2fc43ed5e4f9fb4c9eacf2f31798d37968ead3403d8a
subdir         : win-64
timestamp      : 1566410353540
url            : https://repo.anaconda.com/pkgs/r/win-64/numpy-1.17.0-py36h19fb1c0_0.tar.bz2
dependencies:
  blas 1.0 mkl
  icc_rt >=2019.0.0
  mkl >=2019.4,<2020.0a0
  mkl-service >=2,<3.0a0
  mkl_fft >=1.0.14,<2.0a0
  mkl_random >=1.0.4,<2.0a0
  numpy-base 1.17.0 py36hc3f5095_0
  python >=3.6,<3.7.0a0
  vc >=14.1,<15.0a0
WARNING: 'conda info package_name' is deprecated.
        Use 'conda search package_name --info'.

(base) C:\Users\rr>conda info numpy
```

查虛擬環境名稱

```
]: 1 # !conda info --envs
2 import sys
3 import os
4 #virtual_env_path = sys.prefix # 獲取當前虛擬環境的路徑
5 virtual_env_name = os.path.basename(sys.prefix)# 獲取虛擬環境的名稱
6 print("當前虛擬環境的名稱:", virtual_env_name)#, f'當前虛擬環境的路徑{virtual_env_path}')
7 print('======'*4)
8 import os
9 def list_conda_envs(): # 使用 `conda` 命令列出所有虛擬環境的名稱
10     try:
11         envs = os.popen('conda env list').read()
12         return envs
13     except Exception as e:
14         return str(e)
15 def list_virtualenvwrapper_envs(): # 使用 `virtualenvwrapper` 列出所有虛擬環境的名稱
16     try:
17         envs = os.popen('lsvirtualenv -b').read()
18         return envs
19     except Exception as e:
20         return str(e)
21 conda_envs = list_conda_envs() # 列出所有虛擬環境的名稱
22 virtualenvwrapper_envs = list_virtualenvwrapper_envs()
23 print("Conda 虛擬環境:", conda_envs)
24 print("Virtualenvwrapper 虛擬環境:", virtualenvwrapper_envs)
```

當前虛擬環境的名稱: lab_a01

=====
Conda 虛擬環境: # conda environments:

```
#
Lab_pccu_AIiF          C:\Users\rr\.conda\envs\Lab_pccu_AIiF
lab_pccu_NSer          C:\Users\rr\.conda\envs\lab_pccu_NSer
lab_cpp                C:\Users\rr\.conda\envs\lab_cpp
net20240527            C:\Users\rr\.conda\envs\net20240527
base                   C:\Users\rr\anaconda3
lab_a01                * C:\Users\rr\anaconda3\envs\lab_a01
                       C:\path\to\your\custom\environment
```

Virtualenvwrapper 虛擬環境: