Lab 5: Handling Quantitative Data: Introduction to NumPy

1 What is NumPy?

NumPy (standing for Numerical Python) is a popular library in Python that can efficiently process **multidimensional data arrays**. It is also the foundation of other advanced Python libraries such as **Pandas**.

Examples of 1-dimensinoal array: the elements must have the same data type

- [2, 7, 10, 6] # an array of integers, shape is (4)
- [73.5, 67.0, 87.5, 58.5, 92.0] # an array of student grades or stock prices, shape is (5)
- ['Sam', 'John', 'Zoe'] # an array of names, shape is (3)

Examples of 2-diminsionial array: an array of array with the same shape and data type

- [[3, 6, 4, 8],
  [2, 7, 5, 9],
  [4, 8, 6, 1]] # an array with 3 elements; each element is an array with 4 integers. Shape = (3, 4)

The major features of NumPy include:

- Easily generate and store data in memory in the form of multidimensional array
- Easily load and store data on disk in binary, text, or CSV format
- Support **efficient** operations on data arrays, including basic arithmetic and logical operations, shape manipulation, data sorting, data slicing, linear algebra, statistical operation, discrete Fourier transform, etc.
- Vectorised computation: simple syntax for elementwise operations without using loops (e.g., **a** = **b** + **c** where **a**, **b** and **c** are three multidimensional arrays with the same shape).

In order to use NumPy, we need to import the module **numpy** first. A widely used convention is to use **np** as a short name of **numpy**. In this labsheet, when you see **np.xxx**, it is the same as **numpy.xxx**. Remark: the module name of NumPy library is **numpy** (i.e., lowercase).

```
import numpy as np
```

2 The data type of N-dimensional array: ndarray

The core of NumPy is the N-dimensional array datatype **ndarray**. It can store a collection of data items **with the same type**, i.e., the array is **homogeneous**. This makes it very different from list. A list is more flexible, but less efficient. ndarray can only store items with the same type, but its performance is much better than list (i.e., it takes a shorter time to process the same amount of data).

An ndarray object has the following properties:

ndarray.ndim	The number of dimensions of the array (i.e., 1 or 2 or 3 …)
ndarray.shape	The dimensions of the array (i.e., number of elements in each dimension)
ndarray.size	The total number of elements of the array
ndarray.dtype	The data type of the array
ndarray.itemsize	The number of bytes of each data element
ndarray.data	The buffer that stores the data elements of the array

NumPy supports the following popular data types:

- Integers with different sizes: int8 / int16 / int32 / int64 / uint8 / uint16 / uint32 / unit64
- Real numbers with different sizes: float16 / float32 / float64 / float128
- Complex numbers with different sizes: complex64 / complex128 / complex256
- bool
- Traditional ASCII string with constant length (one byte per character): S10 means a sting with 10 characters
- Unicode string with constant length: U10 means a string with 10 unicode characters

Example 1: Try the following statements about a two-dimensional array:

```
a = np.arange(20) # generate a one-dimensional array first

print(a) # [ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]

a = a.reshape(4, 5) # generate a two-dimensional array from a, and assign it to a

print(a)

type(a) # numpy.ndarray

print(a.ndim) # 2

print(a.shape) # (4, 5)

print(a.dtype.name) # int32

print(a.itemsize) # 4

print(a.size) # 20
```

Exercise 1: Use numpy.arange() to generate a 1-dimenstional array with 100 odd numbers 1, 3, 5, …, 199. Then use numpy.reshape() to generate a two-dimensional array with shape (10, 10). Print out the shape, size, and data of the two-dimensional array.

3 How to create ndarray objects?

We will study four different approaches to creating ndarray objects.

(1) Use the **numpy.array( )** function to generate an ndarray object from any sequence-like object (e.g., list and tuple)

Example 2: Try the following statements about creating ndarrays from lists

```
# import numpy as np

#generate a one-dimensional array from a sequence of data

data1 = [1, 2, 3, 4, 5, 6]

arr1 = np.array(data1)

print(arr1)

print(arr1.ndim)

print(arr1.shape)

#generate a two-dimensional array from a sequence of sequence of data

data2 = [ [1, 2, 3, 4], [5, 6, 7, 8] ]

arr2 = np.array(data2)

print(arr2)

print(arr2.ndim)

print(arr2.shape)

#generate a three-dimensional array from a sequence of sequence of sequence of data

data3 = [ [ [1, 2, 3, 4], [5, 6, 7, 8] ],

          [ [9, 10, 11, 12], [13, 14, 15, 16] ],

          [ [17, 18, 19, 20], [21, 22, 23, 24] ] ]

arr3 = np.array(data3)

print(arr3)

print(arr3.ndim)

print(arr3.shape)
```

(2) Use the following functions to generate some special ndarray object. Use **help( )** to find out the details of each function.

Function	Example	Description
arange	arr = np.arange(20)	Return evenly spaced values within a given interval, simiar to the built-in **range( )** function.
ones	arr = np.ones(10)	An array of all 1's with the given shape
zeros	arr = np.zeros((2, 3))	An array of all 0's with the given shape
full	arr = np.full((3, 4), 1.2)	An array of all specified value with the given shape
empty	arr = np.empty((2, 5))	An array with the given shape without initial data
eye	arr = np.eye(6)	A square NxN ***identity matrix***

Example 3: Try the following statements to learn different array generating functions

```
arr = np.ones(10)

print(arr)

arr = np.zeros( (2, 3) )

print(arr)

arr = np.full( (3, 4), 1.2)

print(arr)

arr = np.empty( (2, 5) )

print(arr)

arr = np.eye(6)

print(arr)
```

(3) Generate ndarray with random numbers (**random sampling**)

The **numpy.random** module provides functions to generate arrays of sample values from popular probability distributions.

Reference: https://docs.scipy.org/doc/numpy/reference/routines.random.html

Example 4: Try the following statements to generate different random arrays. Use **help( )** to understand the functions **random()**, **randint()**, **randn()**, and **uniform()** in numpy.random module.

```
help(np.random.random) #
```

```
arr = np.random.random((2,3)) # Return 2x3 random floats in half-open interval [0.0, 1.0)
print(arr)
arr = np.random.randint(10, 100, 10) # Return 10 random integers in half-open interval [10, 100)
arr = np.random.randn(6, 3) # Draw 6x3 samples from standard normal distribution
print(arr)
arr = np.random.uniform(-1, 1, 10) # Draw 10 samples from a uniform distribution in (-1, 1)
print(arr)
```

(4) Save ndarray to disk file, and ndarray from disk file

Example 5: Try the following statements to save ndarray as **binary file** and load ndarray from binary file (which was previously created by **numpy.save()** function).

a. Binary format (which is not suitable for human to read)

```
arr1 = np.arange(2, 100, 2) # Return an array of [2, 4, 6, …, 96, 98] with stepsize of 2
np.save('even.npy', arr1) # save ndarray to file even.npy
arr2 = np.load('even.npy') # load data from even.npy and create an ndarray
print(arr2)
```

Example 6: Try the following statements to save ndarray as **txt file** and load ndarray from txt file.

b. Txt format (which is suitable for human to read)

```
arr1 = np.arange(0.0, 10.0, 0.5)
np.savetxt('half.txt', arr1, fmt='%.6f') # You can use a text editor to open half.txt
arr2 = np.loadtxt('half.txt')
print(arr2)
```

Exercise 2: Create the following ndarray objects:

- Create an ndarray of shape (8, 8) and all data are 2.5
- Create an ndarray of shape (4, 4) whose values range from 0 to 15
- Create a 6 × 6 identity matrix
- Create a random array of size 20 with standard normal distribution and find its mean value
- Create a random array of shape (3, 6) with random integers in the range of [1, 50).
- Create a random array of shape (4, 5) with uniform distribution in the range of [0, 10). Find its maximum and minimum values and the mean value.

4 How to access and manipulate data in ndarray?

Accessing data in one-dimensional ndarray is similar to the case of list.

Array indexing: use the square brackets ([ ]) to index array values. To access a single data element in two-dimensional ndarray, you need to specify the coordinate of the element (i.e., the indices on the two axes).

If you index a multidimensional array with fewer indices than dimensions, you will get a sub-dimensional array.

Example 7: Try the following statements to access items in ndarray

```
arr2d = np.array([ [1,2,3], [4,5,6], [7,8,9] ])

arr1d = arr2d[2]

print(arr1d) # [7, 8, 9]

print(arr2d[1]) # [4, 5, 6]

arr2d[1][2] = 10 # we can change the values in ndarray

print(arr2d[1]) # [4, 5, 10]

print(arr2d[0][2]) # 3

print(arr2d[0,2]) # 3, the same as the previous print()
```

Slicing: Slicing on ndarray is similar to sequence slicing, but more complicated for 2 or 3-dimensions.

```
arr1d = np.arange(20)

print(arr1d)

arr1d[:10] = 20 # change of first 10 values in arr1d to 20

print(arr1d)

arr1d[10:15] = -1 # change the next 5 values to -1

print(arr1d)

arr1d[-5:] = 0 # change the last 5 values in arr1d to 0

print(arr1d)
```

5 More data processing in ndarray

(1) Universal functions: perform elementwise operations on data in ndarrays

Mathematical functions: https://docs.scipy.org/doc/numpy/reference/routines.math.html

Example 8: Try some vectorised operations and universal functions on ndarrays:

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

y = np.array([5, 6, 6, 8])

print(x+y) # [6 8 9 12]

print(x*y) # [5 12 18 32]

arr = np.arange(10)

print(arr) # [0 1 2 3 4 5 6 7 9 9]

print(np.sqrt(arr)) # [0.   1.    1.41421356 1.73205081 2.  2.23606798  … ]

print(np.exp(arr))
```

(2) Statistics
Reference: https://docs.scipy.org/doc/numpy/reference/routines.statistics.html

Example 9: Try some statistical methods of ndarrays

```
arr = np.random.randn(20, 5)

print(arr)

print("The mean is", arr.mean())

print("The standard deviation is", arr.std())

print("The max and min are:", arr.max(), arr.min())

print("The index of the min is {} and the index of the max is {}".format(arr.argmin(), arr.argmax()))
```

Additional Resources:

If you want to learn more about NumPy, please try the following series of tutorials:

https://www.tutorialspoint.com/numpy/index.htm