INTRODUCTION TO PYTHON

SCHOOL OF COMPUTER SCIENCE, UNIVERSITY OF NOTTINGHAM NINGBO CHINA

LESSON 3: NUMPY

INTRODUCTION TO NUMPY

The fundamental package for scientific computing with Python.

- Matrices and matrix operations
- Linear algebra
- Logic
- Basic statistics
- Multidimensional array object
- Designed for fast and efficient operation

Website: `numpy.org`

OVERVIEW

In this lesson, we will teach you enough to begin writing your own code using NumPy. Topics covered:

1. Matrix representation
2. Accessing elements, rows and columns
3. Views and copies of ndarray objects
4. For loops over matrices
5. Matrix operations

PRE-REQUISITES

- This lesson assumes you have installed Python3, run a Python session and can save your work.
- It also assumes you have basic knowledge of the core Python language.
- If not, then go back to Lesson 1 & 2.

- Follow these examples and exercises on your own computer.

USING NUMPY

NumPy is a package that needs to be imported into Python using the command:

```
import numpy as np
```

The “as np” is optional but makes it easy to refer to NumPy objects later.

VECTORS AND MATRICES IN NUMPY

- Vectors and matrixes can be created using the np.array function.
- For example, create the vector (1 2 2 4 8):

```
x = np.array([1, 2, 2, 4, 8])
print("variable x is", x)
```

- Create the matrix $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$:

```
M1 = np.array([[1, 2, 3], [4, 5, 6]])
print(M1)
```

ACCESS ELEMENTS, ROWS AND COLUMNS

- Each individual element can be accessed by indexing a matrix:

```
print(M1[0,2])
```

  returns 3.

- Each row can also be accessed using a colon ":". Try

```
print(M1[1,:])
```

- This is called *slicing*.
- It can also be done to access a column. Try

```
print(M1[:,2])
```

- Note that the vector output from slicing is also type ndarray.

EXERCISE 1

1. Write code to construct these two matrices:

$$A = \begin{pmatrix} 2 & 3 \\ 5 & 6 \\ 1 & 7 \end{pmatrix}, \qquad B = \begin{pmatrix} 4 & 1 & 5 & 0 \\ 3 & 2 & 6 & 3 \end{pmatrix}$$

2. Write code to determine if the second row of A is the same as the third column of B. Return True if and only if it is.

- *Hint: You can use the "==" operator to compare, but check how it works.*

UPDATE VALUES IN A MATRIX

- Index elements may be updated. Try:

```
M1[0,2] = 99
print(M1[0,2])
```

- Also slices may be updated. Try:

```
M1[:,2] = [12,15]
print(M1)
```

and

```
M1[1,:] = [0, -1, 0]
print(M1)
```

ZERO MATRIX

- Create a matrix of zeroes of any given size. For example:

```
nrows = 3
ZZ = np.zeros( (nrows, 4) )
print(ZZ)
```

- This is useful if we want to create a matrix and fill in the values later.

WHAT IS THE TYPE OF A MATRIX IN NUMPY?

- Find out using the type command:

```
print(type(M1))
```

- This shows it as ndarray class type:

```
<class 'numpy.ndarray'>
```

- Note that “nd” means “N-dimensional”. As the name suggests, you can use it create higher dimensional arrays in NumPy. We will not cover this in this introduction.

THE SHAPE OF A NDARRAY

- The size, or dimension, of an ndarray object can be found using the shape method. Try:

```
print(M1.shape)
```

- This outputs a *tuple* which is a bit like a list but cannot be updated:

```
(2,3)
```

- This tells us that M1 is a 2x3 matrix.
- The elements in a tuple can be accessed like a list. Try:

```
print(M1.shape[1])
```

- Note that a *method* is a function or command that *belongs to* the object.

VIEWS AND COPIES OF NDARRAYS

- The assignment operator in Python allows *referencing* to objects. Try:

```
M2 = M1
print(M2)
```

- However, the two variables then share the same object. So now try:

```
M1[1,0] = 10
print(M2)
```

  and you see that the change in M1 is also seen in M2.

- This is because assignment "=" creates a *view*.

VIEWS AND COPIES OF NDARRAYS

- To make a distinct copy of an object use the copy function. Try:

```
M3 = M1.copy()
M1[1,0] = 88
print(M1)
print(M3)
```

CONCATENATING MATRICES

- Matrices can be concatenated using the concatenate function:

```
M4 = np.concatenate((M1,M3))
print(M4)
print(M4.shape)
```

- Note that this makes a copy of the matrices concatenated.

EXERCISE 2

1. Concatenate matrix B, from Exercise 1, to itself to create a new matrix C.
2. Update the 2nd column of C with the vector (4, 3, 2, 1).
3. Verify that the update to C has not affected B.

FOR LOOP OVER A MATRIX

- For loop can be written that process each row of a matrix.
- For example, this code will print the sum of each row of the matrix:

```
for row in M4:
    print(sum(row))
```

MATRIX OPERATIONS: ADDITION, TRANSPOSE

- Matrix addition is simply done with the "+" operator. Try:

```
M5 = np.array([[1,-1,2], [0,2,3]])
print(M1+M5)
```

- For matrix transpose, use the transpose method. Try:

```
print(M5.transpose())
```

- The matrix product is done using the NumPy function matmul. Try:

```
M6 = np.matmul(M1, M5.transpose() )
print(M6)
```

MATRIX OPERATIONS: LINALG

- The linalg module provides several functions for matrix operations and other linear algebra algorithms.

Task	linalg function	Example
Compute the determinant	`np.linalg.det`	`np.linalg.det(M6)`
Compute the inverse	`np.linalg.inv`	`np.linalg.inv(M6)`
Compute the matrix rank	`np.linalg.matrix_rank`	`np.linalg.matrix_rank(M6)`

EXERCISE 3

1. Create the matrix

$$D = \begin{pmatrix} 0 & 3 & 4 & 0 \\ -1 & 0 & 2 & 1 \\ 3 & 1 & -1 & -2 \\ -1 & 1 & 1 & 0 \end{pmatrix}$$

2. Compute the determinant, inverse and rank of this matrix in NumPy.

EXERCISE 4

- The secondary diagonal of a matrix is the diagonal from the top-right element, downward and leftward.
- For example, it is the bold line in $\begin{pmatrix} 1 & 2 & \mathbf{3} \\ 4 & \mathbf{5} & 6 \\ \mathbf{7} & 8 & 9 \end{pmatrix}$; i.e. (3 5 7).

1. Write a function that computes the secondary diagonal of any given matrix and returns it as a matrix with one row.
2. Test your function on matrix D, from Exercise 3.
3. Ensure your function also works for matrix B, from Exercise 1.

NUMPY: END OF TUTORIAL

- NumPy contains many more components.
- Do some further reading to find out more.