Practical Session 2, 2019

1. Aims of the Session

   In this session, you will learn how to do matrix operations using **Numpy**; how to do image compression using singular value decomposition. Furthermore, you will learn more about Python including how to define your own functions and how to write a program using condition statements and control flow statements.

2. Preparation
   - Click the **window icon** on the bottom left corner:
   - Search **Anaconda Navigator** by typing **Anaconda** from the search box under All Programs.
   - Click on **Anaconda 3.6 Navigator**
   - Click **Launch** under Jupyter from Anaconda Navigator;
   - Once the jupyter notebook is open, what you can see are files and/or folders under C:\Docs
   - Click **New** from Jupyter, then select **Python 3** from the pop-up list. You should see a new page with a blank cell, where you can type in Python code.
   - Click File and select Rename from the pop-up list, you may give the file a name you prefer, for example, practical2.

3. Using matrices in Python

   3.1 How to define matrices

   In this session, you will learn two ways to define a matrix.

   - You can define a matrix using a pair of square brackets. Inside of it, each row is surrounded by one pair of square brackets. Elements in each row are separated by comma; rows in the matrix are also separated by comma. For example, type the following in an empty cell and then click the 'run' button

     Run

     ```
     L =[[1,2,3],[0,10,-5],[-6,-2,11]]
     ```

   To access an element, you can give the corresponding index in its row and column. For example, if you want to access the element in the second row and the third column, you can type the following (As you have learnt from Practical 1, the index of an array counts from **zero**.)

   ```
   print(L[1][2])
   ```

   ```
   -5
   ```

Note that you cannot put two indices in one square as a tuple. For example,

```
L[1,2]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-27-db2a5cd061ea> in <module>()
----> 1 L[1,2]

TypeError: list indices must be integers or slices, not tuple
```

- You can import **NumPy** using np as an alias. Then use **np.array** to define a matrix. For example,

```
L1= np.array([[1, 2, 3],[0, 10, -5],[-6, -2,11]])
```

To access an element, you can do either

```
L1[1][2]
```

or

```
L1[1, 2]
```

.

- There is a class: **numpy.matrix**. However, it is not recommended to use this class, even for linear algebra. Therefore, we will not cover this class in our module. If you are interested in using this class, more details can be found here: https://docs.scipy.org/doc/numpy/reference/generated/numpy.matrix.html

3.2 Basic operation: you will use built-in functions in **NumPy** to do basic computation.

Type the following code in each block into your jupyter notebook and then click the 'run' button. To see what you get, you can either call the variable's name directly or use the print function.

```
a = np.array([10, 15, -2, 9])
b = np.arange(4)
```

```
c= a-b
```

```
a*2
b**2
```

What is the difference between using '*' and '**'?

```
M= np.array([[2, 3], [5, -8]])
```

```
N =np.array([[1, -4],[8, -6]])
```

```
M + N
```

```
M * N
M @ N
M.dot(N)
```

As you can see that '*' returns the product of two matrices, element-wise; '@' and '.dot()' are used for matrix multiplication, which you will learn in the next lecture.

```
np.transpose(M)
```

```
np.trace(M)
```

The trace function is used to compute the sum of elements along the main diagonal line in the matrix.

```
np.diag(M)
```

The diag function is used to get all elements along the main diagonal line in the mtrix.

3.3 numpy.linalg

numpy.linalg is a class having a standard set of matrix decompositions and things like inverse and determinant. More details can be found here: https://docs.scipy.org/doc/numpy/reference/routines.linalg.html

Please not to worry if you do not know these concepts yet. You will learn them in the next lecture. In this practical session, please focus on how to use the correct Python functions to solve each task. You need to import numpy.linalg first.

Task1:

1) compute the inverse of the matrix M defined in 3.2 using **linalg.inv**(a).
2) compute the determinant of matrix N defined in 3.2 using **linalg.det**(a).

Note that the way you call each function can be different depending on how you import the numpy.linalg module, that is, with or without an alias.

4 Define your own functions

You can define your own functions in Python and reuse them throughout the program. The syntax for defining a function is as follows:

```
def  NameofFunction (list of parameters) :
    code detailing what the function does
    return [expression]
```

There are two key keywords here: def and return. def tells the program that the ***indented*** code from the next line onwards is part of the function. return is the keyword that to return an answer from the function. Note that Python uses indentation.

For example, you need to define a function to add two matrices, both of which have a size of 2 by 2. Remember that you can add only the corresponding elements in two matrices. The function is shown as follows, and you need to add it in your own jupyter notebook.

```
def addMatrices(a,b):
    '''adds two 2x2 matrices together'''
    C = [[a[0][0]+b[0][0], a[0][1]+b[0][1]],
         [a[1][0]+b[1][0], a[1][1]+b[1][1]]]
    return C
```

Now suppose that you want to add two matrices using the function **addMatrices**:
A=[[1, 2], [8, -3]]
B=[[0, 10],[-2, 6]]

You need to type in these two matrices first, then type **C = addMatrices(A, B)**. To see the result, you may type **print(C)**.

Task2:

Define a function which can add values along the diagonal line of a given 3 by 3 matrix. Report the result for matrix M = [[1, 2, 3], [0, 10, -5], [-6, -2, 11]] by using your function.

5 Singular Vector Decomposition

Download practical2_SVD.ipynb from Canvas. Open it from Jupyter notebook. The code loads a cat image and converts it into a grey scale format. This image is stored in a python object: **cat_image**.

Task3:

1) Explain the meaning of each line in the first five cells.
2) There is a defined function called **compress_svd**. You need to filled in the ***??*** parts using proper python code. The aim of this function is to use a number of ***k*** singular vectors to reconstruct the image.
3) Run the code with 6 different ***k*** values: 50, 100, 150, 200, 250, and 300.
4) Produce a plot: x-axis is the value of ***k***; y-axis is the corresponding compression ratio. Looking at the plot, what is your conclusion?

6 More about Python: Data Types, Condition Statements and Control Flow

6.1 Data types

Data type simply refers to the type of data that a variable stores. Apart from integer and float you have seen in the previous session, Python provides more types. In this session, you will learn three more: the string, list, and dictionary.

- String
  To declare a string, you can either use single quotes or double quotes as shown as follows:

```
firstName ='David'
surname = "Lee"
```

  You can combine multiple substrings by using the plus sign (+). For instance:

```
username = firstName + " " + surname
```

```
username
```

```
'David Lee'
```

- List

  List refers to a collection of data which are normally related. Square brackets are used when declaring a list. Multiple values are separated by a comma. List elements can be of different data types. For example,

```
myList= [1, 2, 6, 10, 'Hello']
```

  New elements can be added either to the end of a list using the built-in function **append()** or to a list at a particular position using the built-in function **insert()**. For example, ***myList.insert(specified position, new element)***

  Task4:

  Add a string 'e' to **myList** at the second position; add **0** to the end of the list.

- Dictionary

  Dictionary is a collection of related data pairs. For instance, if we want to store the name and score of 5 students, we can store them in a dictionary.

  To declare a dictionary, you write dictionaryName = {dictionary key : data}. For example:

```
myDictionary ={"Peter":80, "Mary":59, "Ann":81, "David":50, "Joe":62}
```

```
myDictionary
```

```
{'Peter': 80, 'Mary': 59, 'Ann': 81, 'David': 50, 'Joe': 62}
```

Note that the dictionary keys must be unique within one dictionary. The following example shows that "Peter" as the dictionary key has been used twice:

```
myDictionary ={"Peter":80, "Mary":59, "Ann":81, "David":50, "Peter":62}
```

```
myDictionary
```

```
{'Peter': 62, 'Mary': 59, 'Ann': 81, 'David': 50}
```

To access individual items in the dictionary, you can use the dictionary key. For instance, to get Ann's score, you write ***myDictionary["Ann"]***

6.2 If statement

The structure of an **if** statement is shown as follows:

if condition 1 is met:

    do A

elif condition 2 is met:

    do B

else:

    do C

elif stands for 'else if' and you can have as many ***elif*** statements as you need to. Note that Python uses indentation. Anything indented is treated as a block that will be executed if the condition evaluates to True. You need to type the following code into your jupyter notebook:

```
userInput = input( 'Enter 1, 2 or 3:')

if (userInput == "1") | (userInput == "2") | (userInput == "3"):
    print ("Hello")
else:
    print ("YOu did not enter a valid number")
```

Task5:

1) Run the program four times, enter 1, 2, 3 and 4 respectably for each run.
2) Modify the above code:
    a. Remove **userInput == "3"** from the if statement
    b. Add one elif statement into the code as follows:

```
elif userInput == "3"
    print ("I love Python")
```

c. Run the program three times, enter 2, 3 and 4 respectably for each run.

6.3 Inline If:

This is used when you only need to perform a simple task.

The syntax is:

do Task A if condition is True else do Task B

For example:

```
userInput = input('Enter 1: ')

print("Hello") if userInput== "1" else print ("You did not enter a valid number")
```

6.4 For loops

The ***for*** loop executes a block of code repeatedly until the condition in the ***for*** statement is no longer valid. The syntax for looping is shown as follows:

for i in iterable:
    print(i)

where an iterable refers to anything that can be looped over such as a string, list, dictionary, and so on. For example,

```
student_names = ['David', 'Mary', 'Joe', 'Anne', 'Mark']

for onename in student_names:
    print(onename)
```

Task6:

Write a ***for*** loop to show each letter in the message 'Hello World'

6.5 While loop

A ***while*** loop repeatedly executes instructions inside the loop while a certain condition remains valid. The structure of a ***while*** statement is shown as follows:

while condition is true:

    do A

For example:

```
num = 4

while num > 0:
    print("num = ", num)
    num = num -1
```

Task7:

Write a program to print integer numbers from 1 to 10 using a ***while*** loop.

References

1. J. VanderPlas: Python Data Science Handbook, 2016, O'Reilly Media, Inc.
2. Jupyter, python, image compression and SVD – an interactive exploration, by Ramesh Putalapattu, 2017
   https://medium.com/@rameshputalapattu/jupyter-python-image-compression-and-svd-an-interactive-exploration-703c953e44f6 by