ENGG1811 Computing for Engineers

Week 5B:

Numpy: motivation, arrays, indexing, slicing, axes. Nested for loop.

Numpy functions: statistical, Boolean, logic, reshape, where

Motivating example

- The manager of a frozen food company approaches you with the following problem:
  - The company has installed 1,000 thermometers to monitor the temperature in its warehouses
  - They have collected 100,000 data points from each thermometer
  - The manager wants to know:
    - Has the temperature in any thermometer ever exceeded a threshold?
    - Which thermometers had readings exceeded the given threshold? How often did it happen?
    - Which thermometer had the highest average temperature?

- Typical modern day data processing problems:
  - Many sequences; each sequence has many data points

Illustrative example: the data

- We assume:
  - There are 5 thermometers
  - Each thermometer has 10 temperature readings
- We store the data in a list of lists
  - The variable temp_list (see below) is a list of 5 lists
  - Each entry of temp_list is a list with 10 elements
- In general, you store
  - A data sequence in a list,
  - Multiple data sequences in a list of lists

```
temp_list = [[ 0.3,  0.4,  0.5,  0.5,  0.1,  0.8,  0.8,  0.5,  0.0,  0.7],
             [ 0.2,  0.4,  0.8,  0.4,  0.8,  1.8,  0.9,  0.1,  1.4,  1.7],
             [ 1.1,  0.1,  0.8,  0.9,  0.5,  0.3,  0.2,  0.2,  1.1,  0.4],
             [ 0.4,  0.7,  0.6,  0.6,  0.4,  0.4,  0.5,  0.0,  0.1,  0.2],
             [ 0.2,  0.1,  0.9,  0.9,  0.3,  0.5,  0.4,  0.7,  0.2,  0.7]]
```

Thermometer 0

Thermometer 2

Illustrative example: the question

- Assuming that the threshold = 1
- For each thermometer, how many times have the readings exceeded the threshold?

```
temp_list = [[ 0.3,  0.4,  0.5,  0.5,  0.1,  0.8,  0.8,  0.5,  0.0,  0.7],
             [ 0.2,  0.4,  0.8,  0.4,  0.8,  1.8,  0.9,  0.1,  1.4,  1.7],
             [ 1.1,  0.1,  0.8,  0.9,  0.5,  0.3,  0.2,  0.2,  1.1,  0.4],
             [ 0.4,  0.7,  0.6,  0.6,  0.4,  0.4,  0.5,  0.0,  0.1,  0.2],
             [ 0.2,  0.1,  0.9,  0.9,  0.3,  0.5,  0.4,  0.7,  0.2,  0.7]]
```

Two different solutions

- Classical programming solution
  - Nested for-loops
- Python numpy library
  - Num is short for numerical

Nested for-loops

- A nested for-loop means a for-loop within a for-loop
  - You have seen nested if's before

- Example problem:
  - See illustration on the next slide for making a nested for-loop
  - Code in nested_for_prelim.py

```
temp_list = [[ 0.3,  0.4,  0.5,  0.5,  0.1,  0.8,  0.8,  0.5,  0.0,  0.7],
             [ 0.2,  0.4,  0.8,  0.4,  0.8,  1.8,  0.9,  0.1,  1.4,  1.7],
             [ 1.1,  0.1,  0.8,  0.9,  0.5,  0.3,  0.2,  0.2,  1.1,  0.4],
             [ 0.4,  0.7,  0.6,  0.6,  0.4,  0.4,  0.5,  0.0,  0.1,  0.2],
             [ 0.2,  0.1,  0.9,  0.9,  0.3,  0.5,  0.4,  0.7,  0.2,  0.7]]
```

threshold = 1

Expected answer = [0, 3, 2, 0, 0]

Nested for-loops

For each thermometer

For a thermometer: # times the readings > threshold

This task is a for-loop

For each reading in thermometer

Increment count if reading > threshold

Combine the code to get nested for-loop:

```
For each thermometer
    For each reading in thermometer
        Increment count if reading > threshold
```

Nested-for loop code

```
# %% Development step 2:
# Use a list within temp_list to cou
# exceeded
thermometer_readings = temp_list[2]

count = 0     # number of times that
for temp in thermometer_readings:
    if temp > threshold:
        count += 1
```

Code in nested_for_prelim.py

```
# %%  Development step 3:
count_exceeding = []

for thermometer_readings in temp_list:

    count = 0
    for temp in thermometer_readings:
        if temp > threshold:
            count += 1

    count_exceeding.append(count)
```

Why numpy?

- numpy has a lot of functions that can save you time in writing code
- If we want to solve the same problem of determining the number of readings in a thermometer exceeding a threshold in numpy, the code is in count_exceed.py
- Line 35 uses the array of temperature and threshold to compute the answer that you want. You may not understand how to use numpy yet but we'd like you to appreciate that you can get the work done with merely 1 line of code

```
26 import numpy as np
27
28 # set the threshold
29 threshold = 1
30
31 # convert the list to an numpy array
32 temp_array = np.array(temp_list)
33
34 # count the number of data points exceeding the threshold per thermometer
35 count_exceeding = np.sum(temp_array > threshold, axis = 1)
```

The numpy library

- The numpy library is a collection of functions which are very useful for data analysis, efficient computation

- Very often, a numpy function can replace many lines of code. Less lines of code means:
  - Shorter development time
  - less debugging

- You combine numpy functions with your coding skills to solve bigger problems

- Why don’t we teach you numpy from the beginning?
  - The same reason why we don’t give calculators to the primary school children

numpy basic concepts

- Basic concepts:
  - Creation, slicing, assignments
  - Shape and indexing
  - Axes

numpy arrays: creation, slicing, assignment

- Basic techniques
  - There are many ways to create numpy **arrays**
    - One way is to enter the arrays directly or convert them from a list
  - Accessing and modifying elements, slicing
    - Information on indexing 2-d arrays are on the next slide
- Code in numpy_elements.py

2-D array: shape and indexing

```
# a 2-dimensional array and
array2 = np.array([ [-1.2,  2, -3.1, 4.5],
                    [   4, -5,  3.5, 7.1],
                    [ 2.7,  9,  1.7, 3.4]])

# You can access individual elements
array2[0,3]  # 4.5
array2[1,2]  # 3.5
```

- The indexing for 2-D array is **similar to matrices** except the indices begin with zero
- We say the shape of this array is (3,4). If it were a matrix, you'd say a 3-by-4 matrix.

↓
Indices

[0,0]	[0,1]	[0,2]	[0,3]
[1,0]	[1,1]	[1,2]	[1,3]
[2,0]	[2,1]	[2,2]	[2,3]

Array axes

- A 2-D numpy array has 2-axes
  - Axis 0
  - Axis 1

Axis 1

Axis 0

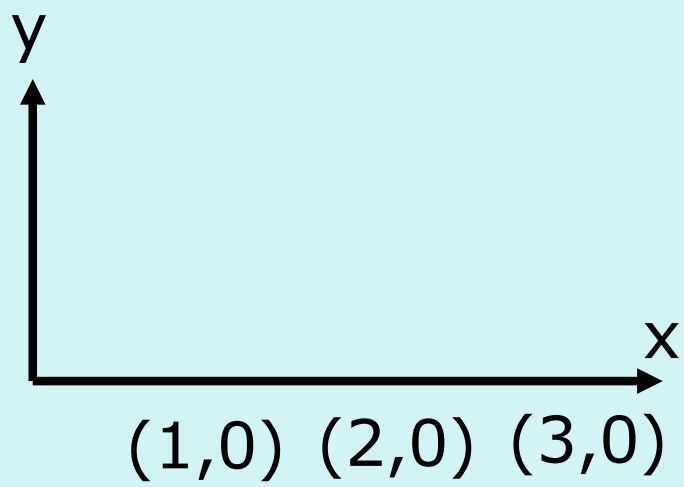
[0,0]	[0,1]	[0,2]	[0,3]
[1,0]	[1,1]	[1,2]	[1,3]
[2,0]	[2,1]	[2,2]	[2,3]

This index is the axis 0 coordinate

This index is the axis 1 coordinate

- Along Axis 0, only Axis 0 coordinates change
- Mnemonic: *Downright*

Cartesian co-ordinates



Along or parallel to x-axis, only the x-coordinates change

Rotate the x-y plane clockwise by 90°

Recap: numpy (1)

- Creating arrays, indexing, slicing, shape

```
# %% import numpy
import numpy as np
```

Column index

0 1 2 3

Row index

← 0

← 1

← 2

```
# Creating a 2-dimensional array and
array2 = np.array([ [-1.2,  2, -3.1, 4.5],
                    [   4, -5,  3.5, 7.1],
                    [ 2.7,  9,  1.7, 3.4]])

# Indexing, slicing, shape
array2[0,3]  # 4.5
array2[1,:]  # array([ 4. , -5. ,  3.5,  7.1])
array2.shape # (3, 4)
```

↓

Indices

[0,0]	[0,1]	[0,2]	[0,3]
[1,0]	[1,1]	[1,2]	[1,3]
[2,0]	[2,1]	[2,2]	[2,3]

Array axes (Recap)

- A 2-D numpy array has 2-axes
  - Axis 0
  - Axis 1

Axis 1

Axis 0

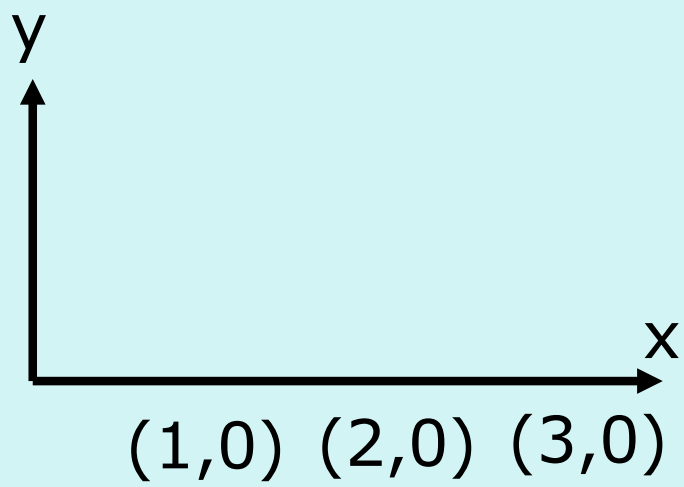
[0,0]	[0,1]	[0,2]	[0,3]
[1,0]	[1,1]	[1,2]	[1,3]
[2,0]	[2,1]	[2,2]	[2,3]

This index is the axis 0 coordinate

This index is the axis 1 coordinate

- Along Axis 0, only Axis 0 coordinates change
- Mnemonic: *Downright*

Cartesian co-ordinates



Along or parallel to x-axis, only the x-coordinates change

Rotate the x-y plane clockwise by 90°

numpy topics

- Functions for basic computation & statistics
- Boolean
- diff, dtype
- reshape(), ravel(), where()

numpy: basic computation and statistics

- Need to prepend with numpy. or the short form that you use
- These functions can be applied to the whole array, or a particular axis

numpy function	What it does
sum()	sum
mean(), median()	Mean, median
average()	Weighted average
std()	Standard deviation
max(), min()	Maximum, minimum
argmax(), argsort()	Argument that maximises / minimises
cumsum(), cumprod()	Cumulative sum and cumulative product
sort(), argsort()	sort

numpy.sum() function

- File is numpy_sum.py

- array1 is:

```
[ [-3.2,  0,  0.5, 5.8],
  [   6, -4,  6.2, 7.1],
  [ 3.8,  5,  2.7, 3.7]]
```

```
[  3.1   15.3   15.2]
```

```
[  6.6    1.     9.4   16.6]
```

```
print(np.sum(array1)) # default parameter value
                      # sum all values in the array

print(np.sum(array1, axis = 0)) # along axis 0

print(np.sum(array1, axis = 1)) # along axis 1
```

Exercise: max(), argmax()

- Go through the file numpy_max.py
- See whether you can figure out what max() and argmax() do
- Hint: argmax() has something to do with where the maximum is located
- Note: arg is short for argument

max() and argmax()

```
ay([ [-3.2,  0,   6.2, 5.8],
     [  6, -4,   0.5, 7.1],
     [ 3.8,  5,   2.7, 3.7]])
```

Row index

← 0

← 1

← 2

```
In [7]: np.max(array1, axis = 0)
Out[7]: array([6. , 5. , 6.2, 7.1])

In [8]: np.argmax(array1, axis = 0)
Out[8]: array([1, 2, 0, 1])
```

Exercise

- The file is numpy_max_prelim.py
- Temperature readings are stored in temp_array
  - Each row corresponds to a thermometer, labelled as 0,1,….,4
  - Each thermometer has 10 readings
- Your tasks
  - Find the maximum temperature in each thermometer
  - Determine which thermometer has the highest temperature at this time?

```
temp_array = np.array(
            [[ 0.3,  0.4,  0.5,  0.5,  0.1,  0.8,  0.8,  0.5,  0.0,  0.7],
             [ 0.2,  0.4,  0.8,  0.4,  0.8,  1.8,  0.9,  0.1,  1.4,  1.7],
             [ 1.1,  0.1,  0.8,  0.9,  0.5,  0.3,  0.2,  0.2,  1.1,  0.4],
             [ 0.4,  0.7,  0.6,  0.6,  0.4,  0.4,  0.5,  0.0,  0.1,  0.2],
             [ 0.2,  0.1,  0.9,  0.9,  0.3,  0.5,  0.4,  0.7,  0.2,  0.7]]
            )
```

Time

Some commonly used functions

numpy function	What it does
sum()	sum
mean(), median()	Mean, median
average()	Weighted average
std()	Standard deviation
max(), min()	Maximum, minimum
argmax(), argsort()	Argument that maximises / minimises
cumsum(), cumprod()	Cumulative sum and cumulative product
sort(), argsort()	sort

numpy topics

- Functions for basic computation & statistics
- Boolean
- diff, dtype
- reshape(), ravel(), where()

Boolean numpy arrays

- You can do **elementwise** comparison using ==, !=, >, <, >=, <=
  - Try it out in numpy_boolean_1.py
  - The result is a numpy arrays of Boolean type

```
array1 = np.array([ [-3.2,  0,  0.5, 5.8],
                    [   6, -4,  6.2, 7.1],
                    [ 3.8,  5,  2.7, 3.7]])
```

```
In [8]: array1_cp0 = array1 > 2

In [9]: array1_cp0
Out[9]:
array([[False, False, False,  True],
       [ True, False,  True,  True],
       [ True,  True,  True,  True]])
```

Counting the number of True's

- You can use numpy.sum() to count the number of True's
  - numpy_boolean_1.py

- At the beginning of the last lecture, we mentioned the problem of determining the number of readings in a thermometer exceeding a threshold
  - The code is in count_exceed.py

```
import numpy as np

# set the threshold
threshold = 1

# convert the list to an numpy array
temp_array = np.array(temp_list)

# count the number of data points exceeding the threshold per thermometer
count_exceeding = np.sum(temp_array > threshold, axis = 1)
```

Boolean operators

- There are three numpy Boolean operators:
  - numpy.logical_and()
  - numpy.logical_or()
  - numpy.logical_not()

- Example (see next slide for a fuller explanation) and a quiz in numpy_boolean_2_prelim.py

- Forum exercise:
  - You know how to count the number of True's. How do you count the number of False's?

Illustrating elementwise and

```
array1 = np.array([ [ 3.2,  0.5, 5.8],
                    [   6, -1.2, 7.1]])

array1_la = np.logical_and(array1 > 3,array1 < 7)
```

□array1 > 3

□array1 < 7

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

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

logical_and()
(= apply logical and to corresponding elements in the 2 arrays)

□array1_la

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

Some useful logic functions

- The full list of logic functions are at:
  - https://docs.scipy.org/doc/numpy-1.17.0/reference/routines.logic.html

numpy function	What it does
all()	True if all elements along an axis is True
any()	True if at least one element along an axis is True
allclose(), isclose()	Are elements in two arrays within a tolerance?
array_equal()	Same shape and equal elements for two arrays

Example and quiz are in numpy_logic_prelim.py

numpy topics

- Functions for basic computation & statistics
- Boolean
- diff, dtype
- reshape(), ravel(), where()

numpy.diff()

- The code is in numpy_diff.py

```
np.array([ [-3,  0,  0,  5],
           [ 6, -4,  6,  7],
           [ 3,  5,  2,  3]])
```

Along axis 1

```
[[  3   0   5]
 [-10  10   1]
 [  2  -3   1]]
```

Along axis 0

```
[[ 9 -4  6  2]
 [-3  9 -4 -4]]
```

dtype

- All the elements in a numpy array must have the same dtype. Possible dtype's are:
  - float
  - int
  - bool

- We will explore the implication of this in the file numpy_dtype.py

numpy topics

- Functions for basic computation & statistics
- Boolean
- diff, dtype
- reshape(), ravel(), where()

reshape(), ravel()

- You can reshape the arrays using
  - numpy.reshape()
  - numpy.ravel()
- See examples in numpy_reshape.py
- Explanation on the next slide
- You will use reshape() to do something useful in the lab next week

```
b = np.array([[  3,   9,    5,   1],
              [ 14,   51, 16,   7],
              [ 12,   39, 47, 11]])

b_flat = np.ravel(b)
```

The function numpy.ravel() takes a ”row” (default) at a time and concatenate them

```
In [65]: b_flat
Out[65]: array([ 3,  9,  5,  1, 14, 51, 16,  7, 12, 39, 47, 11])
```

```
In [9]: b_4by3 = np.reshape(b,(4,3)); print(b_4by3)
[[ 3  9  5]
 [ 1 14 51]
 [16  7 12]
 [39 47 11]]
```

numpy indexing

- We go back to the file numpy_max.py

```
# index_max = np.argmax(array1)
# indices = np.unravel_index(index_max, array1.shape)
```

- Uncomment these lines and run the file. You will find index_max is 7
- What is this number 7?
  - This is the "ravel" index
  - See next slide for explanation

```
array1 = np.array([ [-3.2,  0,  6.2, 5.8],
                    [   6, -4,  0.5, 7.1],
                    [ 3.8,  5,  2.7, 3.7]])
```

↓ ravel()

```
In [69]: print(np.ravel(array1))
[-3.2  0.   0.5  5.8  6.  -4.   6.2  7.1  3.8  5.   2.7  3.7]
```

↑ index of 7.1 is 7

How do we get the original indices back? Let us go back to the previous slide.

numpy.where()

- The code is at numpy_where.py

↓ picture of array1 in numpy_where.py

-3	0	0	5
6	-4	6	7
3	5	2	3

array1[1,3] is 7

```
print(np.where(array1 == 7))
```

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

numpy.where()

- The code is at numpy_where.py

↓ picture of array1

-3	0	0	5
6	-4	6	7
3	5	2	3

```
print(np.where(array1 >= 5))
```

The indices of the elements which are >= 5 are:

[0 , 3]
[1 , 0]
[1 , 2]
[1 , 3]
[2 , 1]

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

numpy has many functions

- Routines
  - Array creation routines
  - Array manipulation routines
  - Binary operations
  - String operations
  - C-Types Foreign Function Interface (`numpy.ctypeslib`)
  - Datetime Support Functions
  - Data type routines
  - Optionally Scipy-accelerated routines (`numpy.dual`)
  - Mathematical functions with automatic domain (`numpy.emath`)
  - Floating point error handling
  - Discrete Fourier Transform (`numpy.fft`)
  - Financial functions
  - Functional programming
  - NumPy-specific help functions
  - Indexing routines
  - Input and output
  - Linear algebra (`numpy.linalg`)
  - Logic functions
  - Masked array operations
  - Mathematical functions
  - Matrix library (`numpy.matlib`)
  - Miscellaneous routines
  - Padding Arrays
  - Polynomials
  - Random sampling (`numpy.random`)
  - Set routines
  - Sorting, searching, and counting
  - Statistics
  - Test Support (`numpy.testing`)
  - Window functions

- The categories of functions that numpy has are listed here
  - https://docs.scipy.org/doc/numpy/reference/

Summary

- Data as two-dimension numpy array
- Nested for-loops
- numpy basic concepts
  - Creation of arrays, indexing, slicing, axes
- numpy is ideal for data analysis
  - No need to use loops
  - Elementwise operation
  - Operation along an axis
- A vast collection of functions that can speed up your data analysis