Unit 2: Numpy

Computer Modelling

1 Aims

In this unit you will:

- learn how to use numpy arrays and vector operations.
- learn how to write functions that accept and return numpy arrays.
- write a gaussian likelihood function.

2 Preparation

Make a new file, `unit2.py`. You will write all your code in this file.

3 Background

3.1 Debugging and testing code

Most programming is debugging; this takes longer than initially writing the code. These notes may help you do this efficiently.

VSCode will highlight problems it can detect in your code by underlining them with a wiggly line. You can hover over the line to see what the problem is. You can jump to the next problem in your code by pressing `F8`. For more detail you can click the yellow icon that appears next to the line when you hover over it, and ask CoPilot to fix it.

Note that not all errors will be detected; in particular, VSCode cannot detect errors which produce valid code which is the wrong answer. For example, this code is perfectly valid but obviously wrong:

```
def add(a, b):
    return a - b
```

For this kind of thing it is useful to write tests, which are code that checks your

code is working correctly.

3.2 Arrays

You should carefully read through the lecture slides to gain an understanding of how to use numpy arrays. Here are some key points:

- Homogenous: numpy arrays, unlike python lists, can only contain one type of data, nearly always numerical.
- Fixed size: numpy arrays have a fixed size, which is set when they are created. You can't add or remove elements from a numpy array. Functions which look like they do this actually just make a whole new array.
- Fast: numpy arrays are much faster than python lists for numerical operations.
- Vectorised operations: numpy arrays can be used in vectorised operations, which means that you can perform operations on the whole array at once, rather than looping over each element. This is much faster and easier to read.
- Slicing: you can extract pieces or individual elements of a numpy array very easily and operate on them, for example single rows from matrices, or the last element of an array.

3.3 Data analysis

Next semester we will be analyzing samples drawn from a distribution. As preparation for that, we will try some general descriptive analysis (that is, intended to describe the properties of data rather than model it) in one of these tasks. When you are analyzing a new data set, the first things you should usually look at are: means and standard deviations, histograms of individual data points, and scatter plots of pairs of data points.

3.4 Function inputs and outputs

When we talk about the inputs and outputs to a function, we mean the arguments it accepts (inputs) and the values it returns (outputs). It's usually wrong for the functions you write in this course to ask the user to type something in as an input, or to use some fixed value.

On another note, the things you return from your functions should generally be useful to *the rest of your code*, not just for you when printed to the screen. So it's usually more useful to return a True or False value than text describing something.

4 Tasks

Add the tasks below to your `unit2.py` file. You should make good use of numpy throughout this work, and you should demonstrate the functions that you write from your `main` function.

Some of the tasks below ask you to include comments as part of the work. You will be graded on these. You can include additional comments as well as these if you think they are helpful, and if so they should be correct and relevant.

4.1 Creating numpy arrays

Task 1a: Complete the function called `task1a` so that it takes as input two integer values, m and n, and returns three numpy arrays:

1. a 1D array of all zeros of size m
2. a 1D array containing the numbers 1, 2, 3, ... n
3. a 2D array of uniform random numbers in the range [0, 1) of shape (m, n)

Task 1b: Write a docstring for your function `task1a`.

A *docstring* is a string (text) which must appear as the first line in a python function, and is designed for users of your code to read to help them understand the function. They should be triple-quoted strings, like this:

```
def my_function(x):
    """
    This is a docstring.

    It can be split over multiple lines, and should explain
    what the function does, and its inputs and outputs.
    """
```

Task 1c: Write a function called `task1c` that calls your function from task 1a, and returns the mean value of its second array output and the maximum value of its third array output.

Task 1d: Write a function called `task1d` that takes as input a numpy array `a`, and doesn't return anything, but does modify all the values inside the array `a` to be the square of their original values. Ask the AI to explain the difference between mutable and immutable objects, and write that explanation in a comment in your code.

4.2 Vector arithmetic

Task 2a: Write a function called `task2a` which computes $2t(\mathbf{a} + \mathbf{b})$ for vectors a, b and scalar t.

Task 2b: Write a function called `task2b` which takes as input two 3D position vectors $\mathbf{x}$ and $\mathbf{y}$, and returns the distance between the positions.

Hint: Numpy has a library function that can help. That's a better solution than doing it manually.

Task 3: Write three functions called `task3a`, `task3b`, `task3c` that each compute from their inputs two numpy vectors, representing the left- and right-hand sides of these three vector identities:

$$\boldsymbol{v}_1 \times \boldsymbol{v}_2 = -\boldsymbol{v}_2 \times \boldsymbol{v}_1 \qquad (a)$$

$$\boldsymbol{v}_1 \times (\boldsymbol{v}_2 + \boldsymbol{v}_3) = (\boldsymbol{v}_1 \times \boldsymbol{v}_2) + (\boldsymbol{v}_1 \times \boldsymbol{v}_3) \qquad (b)$$

$$\boldsymbol{v}_1 \times (\boldsymbol{v}_2 \times \boldsymbol{v}_3) = (\boldsymbol{v}_1 \cdot \boldsymbol{v}_3)\boldsymbol{v}_2 - (\boldsymbol{v}_1 \cdot \boldsymbol{v}_2)\boldsymbol{v}_3 \qquad (c)$$

4.3 2D Arrays

We can use a 2D array to represent a matrix.

Task 4a: Write a function called `task4a` which takes an integer `n` and makes a square 2D array `M` of shape $n \times n$, with elements filled in so that $\mathrm{M}_{ij} = i + 2j$ where i and j start from zero.

Task 4b: Write a function called `task4b` which takes an integer `n` as input, and computes a 1D array $y_i = \sum_{j=0}^{j=n-1} M_{ij}$. You should consider how you can make this function as simple as possible, given functions you already have, but you shouldn't use any analytic expressions for the result.

4.4 Likelihoods

Task 5 Write a function `task5` that computes a Gaussian log-likelihood from three vectors:

$$\log L = \frac{1}{2} \sum_i \left(\frac{d_i - \mu_i}{\sigma_i} \right)^2$$

4.5 Data analysis

Task 6 Look at the file `data.txt`. Write a function `task6` that does basic data analysis of the data, as described above. Label each output correctly. You can either save them to files or display them on screen.

4.6 Main function

Write a main function that shows off the use of all your other functions, demonstrating their use and ideally showing that they produce a correct answer. It should run only when the file is run, not when it is imported.

5 Submitting

Submit your single python file `unit2.py`.

5.1 Mark Scheme

This unit is marked out of 20, and counts in total for 10% of your final grade for this course.