# ENGG1811 Assignment 1: Pattern detection

---

- **Due date:** 9am, Tuesday of Week-08 .
- **Late Penalty:** Late submissions will be penalised at the rate of **10% per day** (including weekends). The penalty applies to the maximum available mark. For example, if you submit 2 days late, maximum available marks is 80% of the assignment marks.
- Submissions will **not** be accepted after 9am Saturday of Week-08.

---

## The pattern detection problem

In this assignment, your goal is to write a python program to determine whether a given pattern appears in a data series, and if so, where it is located in the data series. This type of problems is very common in many disciplines, including computer science, engineering, medicine and science. There are many different types of pattern detection problems, the setting of this assignment is similar to that used in radars. A radar transmits a pulse of a specific shape and waits for a pulse of similar shape to return, in order to determine the position of an object. The method described below is known as matched filtering and is widely used in communication systems. This means your mobile phones perform the same type of calculations that you will be programming below!

## Learning objectives

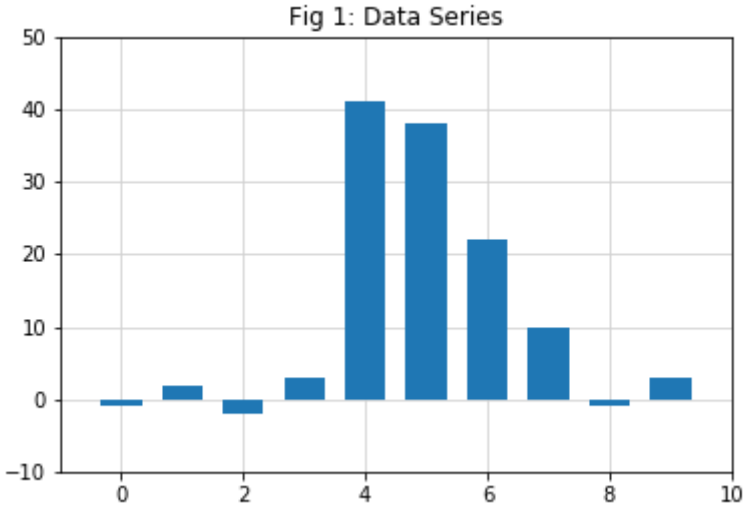
By completing this assignment, you will learn:

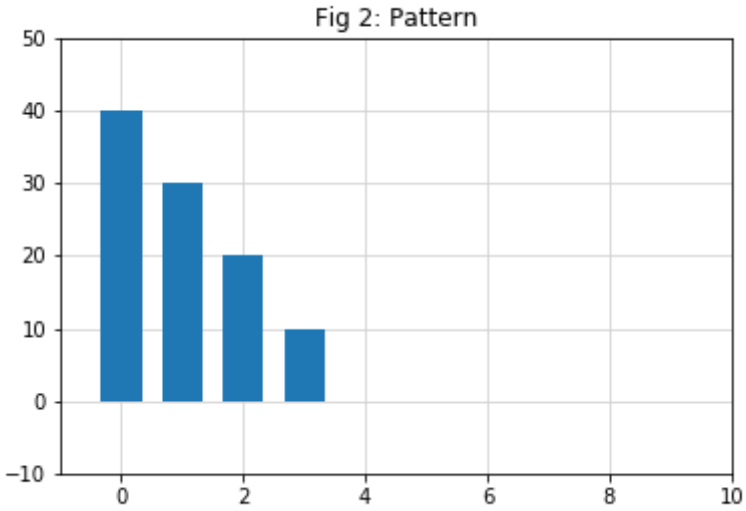
1. To apply programming concepts of variable declaration, constant declaration, assignment, selection and iteration (for loop).
2. To translate an algorithm described in a natural language to a computer language.
3. To organize programs into smaller modules by using functions.
4. To use good program style including comments, meaning variable names and others.
5. To get a practice on software development, which includes incremental development, testing and debugging.

## Algorithm for locating a pattern in a data series

The goal is to detect whether a certain pattern appears in the data series. In the following example, the pattern to be detected is a sequence of 4 numbers (see Fig 2); and the data series contains 10 data points (see Fig 1).

```
data_series = [-1, 2, -2, 3, 41, 38, 22, 10, -1, 3]
pattern = [40, 30, 20, 10]
```




If you compare these two figures, you can see that this pattern appears between 5th (at index 4) to 8th (at index 7) data points of the data series. Note that it is not an exact match, but a fairly close one. Now you can spot the pattern using your eyes, let us see how an algorithm can do it.

Since the given pattern has 4 data points, we will take a segment of 4 consecutive data points from the data series at a time and compute a similarity measure. The similarity measure should have the property that if a segment of the data series is similar to the given pattern, then the similarity measure of that segment is large, and vice versa. Hence, if we can compute the similarity measures of all possible segments, then we can identify the segment that is most similar to the given pattern. We will now provide more details.

The algorithm begins with computing the similarity measure between the given pattern and the *first segment*, which is formed by the first 4 data points of the data series. In terms of the two lists above, the similarity measure for the first segment is:

```
data_series[0]*pattern[0] + data_series[1]*pattern[1] +
data_series[2]*pattern[2] + data_series[3]*pattern[3]
```

After this, we compute the similarity measure between the given pattern and the *second segment*, which is formed by the second to fifth data points of the data series. For the above example, the similarity measure for this segment is:

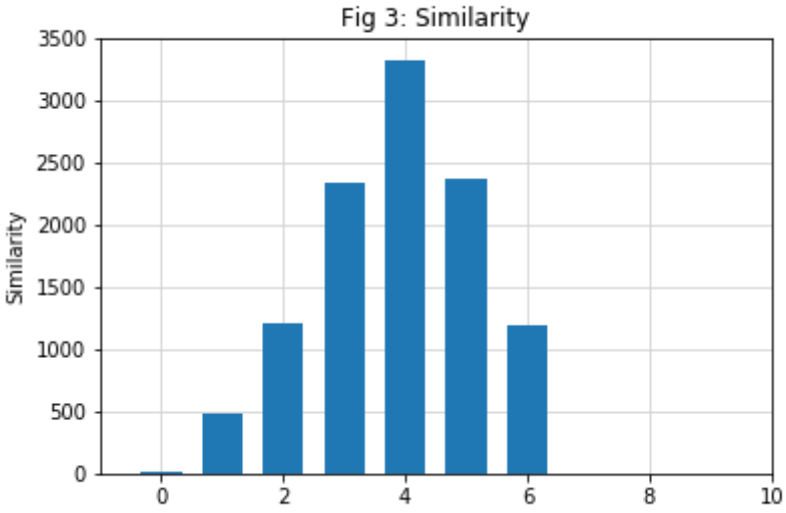
```
data_series[1]*pattern[0] + data_series[2]*pattern[1] +
data_series[3]*pattern[2] + data_series[4]*pattern[3]
```

We then do the same with the segment formed by the third to sixth data points (the third segment), giving a similarity measure equals to:

```
data_series[2]*pattern[0] + data_series[3]*pattern[1] +
data_series[4]*pattern[2] + data_series[5]*pattern[3]
```

We repeat this until we have computed the similarity measure for the last possible segment, which is formed by the 7th to 10th data points. The following is the similarity list for the above example and a plot:

```
similarity = [10, 490, 1210, 2330, 3320, 2370, 1190]
```




We need to consider the following cases.

*Case 1*: It is possible that the given data series is shorter than the given pattern, in this case, we return "Insufficient data".

*Case 2:* All the similarity measures are (strictly) less than the given *threshold value*. This means none of the segments is similar to the given pattern. In this case, we say the pattern is not present in the data series and return "Not detected". This is not the case for this example.

*Case 3*: At least one similarity measure is greater than or equal to the given *threshold value*. This is the case for this example. The similarity measure plot above shows that the fifth similarity measure is the largest. You can work out that the fifth similarity measure is computed by using the segment formed by the fifth (index 4) to eighth (index 7) data points of the data series. This procedure is therefore telling us that the segment consisting of fifth to eighth data points is most similar to the given pattern. Indeed this is what we had found by using visual inspection earlier. We will identify the location of the pattern by using the first index of the segment that has the highest similarity measure.

- function `pattern_search_max` (described below): we return the index of the highest similarity measure that is also greater than or equal to the given *threshold value*. In Fig 3, the index of the highest similarity measure is 4.

- function `pattern_search_multiple` (described below): Consider the following definition of *overlapping* indices,

  > We say two indices are *overlapping* if the distance between them is less than the width of the pattern (4 in the above example).

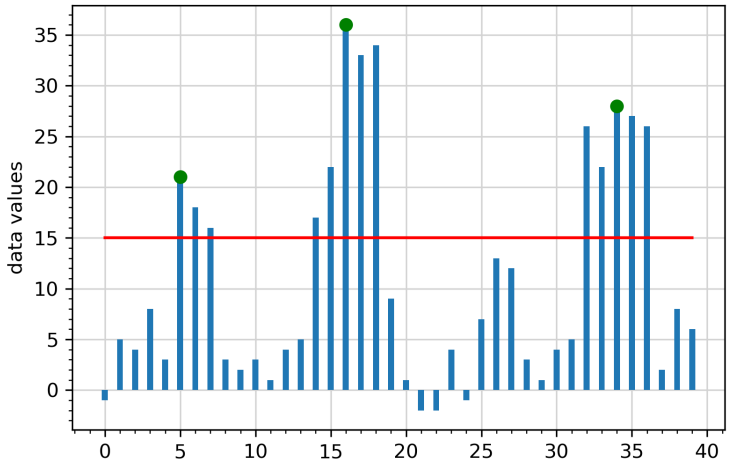
  The function 'pattern_search_multiple' returns a list of *non overlapping indices* that are greater than or equal to the given *threshold value*, **and** that satisfy the following criteria:

  - an index is **not** selected if the value at the index is less than a value at one of it's overlapping indices.
  - an index is **not** selected if it is overlapping with first or last index.

In the following example, selected indices are marked with green circles. Here, index 6 is not selected because the value at index 5 (distance of 1) is greater than the value at index 6. Similarly index 32 is not selected because the value at index 34 (distance 2) is higher. The example in Fig 4 will return the list [5, 16, 34].

There are many different approaches to calculate similarity measures between data segments and a pattern, above we have discussed just one of them! Therefore, a given data series representing similarity measures may or may **not** have clearly defined 'pyramid' patterns as shown in Fig 3. For example, there are no clearly defined 'pyramid' patterns in Fig 4.

Fig 4: pattern_search_multiple
(threshold = 15, pattern_width = 4)




This completes the description of the tasks

## Implementation requirements

You need to implement the following four functions, each in a separate file (provided). You need to submit these four files, each containing one function you implement.

1. `def calculate_similarity(data_segment, pattern):`

   - The aim of this function is to calculate the similarity measure between **one** data segment and the pattern.
   - The first parameter 'data_segment' is a list of (float) values, and the second parameter 'pattern' is also a list of (float) values. The function calculates the similarity measure between the given data segment and pattern and returns the calculated similarity value (float), as described earlier in this assigment. The function returns "Error" if 'data_segment' and 'pattern' have different lengths.
   - This function can be tested using the file '`test_calculate_similarity.py`'.

2. `def calculate_similarity_list(data_series, pattern):`

   - The aim of this function is to calculate the similarity measures between **all** possible data segments and the pattern.
   - The first parameter 'data_series' is a list of (float) values, and the second parameter 'pattern' is also a list of (float) values. The function calculates the similarity measures, using the above function 'calculate_similarity', of all possible data segments in a sequence, and returns the list of calculated similarity values.
   - This function can be tested using the file '`test_calculate_similarity_list.py`'.

3. `def pattern_search_max(data_series, pattern, threshold):`

   - The three possible outcomes are "Insufficient data", "Not detected" and the location of the pattern if the pattern is found in the data series. Here, the location

of the pattern refers to the index of the highest similarity measure that is also greater than or equal to the given threshold value.
- The first parameter 'data_series' is a list of (float) values, the second parameter 'pattern' is a list of (float) values, and the third parameter 'threshold' is a (float) value. In this function, you need to use the function 'calculate_similarity_list'.
- This function can be tested using the file '`test_pattern_search_max.py`'.

4. **`def pattern_search_multiple(data_series, pattern_width, threshold):`**

- The three possible outcomes are "Insufficient data", "Not detected" and a list of *non overlapping indices* that are greater than or equal to the given *threshold value*, **and** that satisfy the following criteria:
  - an index is **not** selected if the value at the index is less than a value at one of it's overlapping indices.
  - an index is **not** selected if it is overlapping with first or last index.

  We say two indices are *overlapping* if the distance between them is less than the width of the pattern.
- The first parameter 'data_series' is a list of (float) values, the second parameter 'pattern_width' is a (float) value, and the third parameter 'threshold' is a (float) value.
- This function can be tested using the file 'test_pattern_search_multiple.py'.

## Getting Started

1. Download the zip file **ass1_prelim.zip**, and unzip it. This will create the directory (folder) named 'ass1_prelim'.
2. Rename/move the directory (folder) you just created named 'ass1_prelim' to '**ass1**'. The name is different to avoid possibly overwriting your work if you were to download the 'ass1_prelim.zip' file again later.
3. First browse through all the files provided, and importantly read comments in the files.
4. Do not try to implement too much at once, just one function at a time and test that it is working before moving on.
5. Start implementing the first function, properly test it using the given testing file, and once you are happy, move on to the the second function, and so on.
6. Please do **not** use 'print' or 'input' statements. We won't be able to assess your program properly if you do. Remember, all the required values are part of the parameters, and your function needs to return the required answer. Do not 'print' your answers.

## Testing

Test your functions thoroughly before submission. You can use the provided python programs (files like 'test_calculate_similarity.py', 'test_pattern_search_max.py', 'test_pattern_search_multiple.py', etc.) to test your functions.

Please note that the tests provided in these files cover **only basic scenarios (cases)**, you need to think about other possible cases, modify the files accordingly and test your functions. For example, you need to **add cases** to test for "Insufficient data" and "Not detected" scenarios.

## Submission

You need to submit the following four files. Do not submit any other files. For example, you do not need to submit your modified test files.

- calculate_similarity.py
- calculate_similarity_list.py
- pattern_search_max.py
- pattern_search_multiple.py

Later, **instructions on how to submit your files will be posted here**.

## Assessment Criteria

We will test your program thoroughly and objectively. This assignment will be marked out of 25 where 20 marks are for correctness and 5 marks are for style.

### Correctness

The 20 marks for correctness are awarded according to these criteria.

| Criteria | Nominal marks |
|---|---|
| Function calculate_similarity | 4 |
| Function calculate_similarity_list | 4 |
| Function pattern_search_max | 3 |
| Function pattern_search_multiple | 5 |
| Case "Insufficient data" | 2 |
| Case "Not detected" | 2 |

### Style

Five (5) marks are awarded by your tutor for style and complexity of your solution. The style assessment includes the following, in no particular order:

- Use of meaningful variable names where applicable
- Use of sensible comments to explain what you're doing
- Use of docstring for documentation to identify purpose, author, date , data dictionary, parameters, return value(s) and program description at the top of the file

## Assignment Originality

You are reminded that work submitted for assessment must be your own. It's OK to discuss approaches to solutions with other students, and to get help from tutors and consultants, but you must write the python code yourself. Sophisticated software is used to identify submissions that are unreasonably similar, and marks will be reduced or removed in such cases.

## Further Information

- Additional **Help Sessions** will be available for this assignment during week-06 to week-08.
- Use the forum to ask general questions about the assignment, but take specific ones to **Help Sessions**.
- Keep an eye on the class webpage notice board for updates and responses.