**Version and change log**

**Due Date : Friday 29th October 5pm AEST**

**1 Introduction**

Automatic detection of faults can be found in many engineering systems. There are systems to automatically diagnose faults in engines, chemical plants, photovoltaic plants, manufacturing processes and on on.

This assignment is *inspired*by a fault detection system in a water treatment plant [[1](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#XWTW)]. There are two main types of water treatment plants. One is to produce potable water (colloquially known as drinking water) and the other is to treat wastewater before discharging it into the natural environment. In both cases, water quality is an important issue. Therefore, there are instruments in the plants to constantly measure water quality and there are computers to process the collected data to check whether the plant is working correctly. If a fault has been developed, then the plant technicians should be alerted automatically so that they can fix the faults as soon as possible.

In this assignment, you will write Python programs to perform fault detection. The aims of your program is to process data sequences to determine whether there are faults, the type of faults and when the fault is detected.

Note that we chose the word inspired earlier because we have adapted the fault detection problem in [[1](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#XWTW)] as a programming assignment by simplifying and liberally changing a few aspects of the original problem. In particular, we have made changes so that, in this assignment, you will have to use the various Python constructs that you have learnt. This means a few details of this assignment may not be realistic in engineering terms, but on the whole, you will still get a taste on how programming can be used to perform automatic fault detection.

**1.1 Learning objectives**

By completing this assignment, you will learn:

* To apply basic programming concepts of variable declaration, assignment, conditional, functions, loops and import.
* To use the Python data types: list, float, int and Boolean
* To translate an algorithm described in a natural language to a computer language.
* To organize programs into modules by using functions
* To use good program style including comments and documentation
* To get a practice on software development, which includes incremental development, testing and debugging.

**1.2 Prohibition**

For this assignment, the only external Python library that you are allowed to use is the math library. Any other libraries, such as numpy , scipy etc., are not allowed.

This is an individual assignment, so no group work.

**2 Intuition behind the fault detection algorithm**

The reference [[1](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#XWTW)] uses a fault detection algorithm called CUSUM and you will be programming this algorithm in the assignment. We want to give you some intuition behind CUSUM. The name CUSUM is short for *cumulative sum*because the algorithm does the action of accumulating, as you will see in a moment.

Figures 1a and 1b plot two signals (or data sequences) that you will be using in your testing. These two signals may look the same but they are actually slightly different. In fact, some of the data in one signal have slightly higher values because of a fault. Since the difference is small, it is not possible to use only one data point to decide whether there is a fault. However, each data point is important. Although a data point may only give us a tiny amount of evidence on whether there is a fault or not, what we can do is to *accumulate*the evidence from many data points. If in the end, the cumulative evidence that there is a fault is large, then we can be confident that a fault has in fact been developed. The blue curves in Figures 1c and 1d show the cumulative evidence of whether there is a fault in the signals in, respectively, Figures 1a and 1b. The red horizontal dashed lines can be thought of as the threshold of evidence that we need to decide whether there is a fault. If the cumulative evidence goes above the threshold, as in Figure 1c, then we say that a fault has occurred. If the cumulative evidence stays below the threshold, as in Figure 1d, then a fault has not been detected. The computer programs that you will write in this assignment will enable you to calculate the cumulative evidence.

|  |  |
| --- | --- |
| Chart, line chart  Description automatically generated **Fig 1a**. The signal used in the test case 0 of the test file test\_fault\_detection\_main\_1.py | Chart, line chart  Description automatically generated **Fig 1b**. The signal used in the test case 1 of the test file test\_fault\_detection\_main\_1.py |
| Chart, line chart  Description automatically generated **Fig 1c**. The CUSUM of the signal in Fig. 1a. CUSUM can be thought of as cumulative evidence that a fault has occurred. | Chart  Description automatically generated **Fig 1d**. The CUSUM of the signal in Fig. 1b. CUSUM can be thought of as cumulative evidence that a fault has occurred. |

**3 Requirements for fault detection**

This section describes the requirements on the fault detection algorithm that you will be programming in this assignment. You should be able to implement these requirements by using only the Python skills that you have learnt in the first four weeks' of the lectures in this course.

We begin with describing the data that the algorithm will operate on. We will use the following Python code as an example. In the following, we will refer to the following code as the sample code. Note that the data and parameter values in the sample code are for illustration only; your code should work with any valid input data and parameter values.

*# %% Test data and parameters*  
  
*# The conc signal*  
conc\_signal = [6.55, 6.40, 6.76, 6.45, 6.87, 6.92, 6.45, 7.36, 7.41, 6.67]  
  
*# Parameters used to detect whether there is a fault in the conc\_signal*  
conc\_range = [6.1, 6.9]  
conc\_var = 0.5  
conc\_cusum\_limit = 1.55  
  
*# The flow\_signal*  
flow\_signal = [22.94, 23.07, 23.97, 23.9, 24.05, 23.65, 23.78, 23.78, 23.94, 23.91]  
  
*# The parameters used to detect whether there is a fault in the flow\_signal*  
flow\_range = [22.5, 23.5]  
flow\_control\_window = 3  
  
*# %% Call the fault detection function*  
import detect\_fault\_main as df  
fault\_type\_your = df.detect\_fault\_main(conc\_signal,conc\_range,conc\_var,  
conc\_cusum\_limit,flow\_signal,flow\_range,flow\_control\_window)

In the sample code, there are two data series which contain, respectively, concentration and flow measurements. Both series are Python lists, and their variable names are conc\_signal and flow\_signal .

In addition to the two data series, your programs will make use of five algorithmic parameters with the names conc\_range , conc\_var , conc\_cusum\_limit , flow\_range , and flow\_control\_window . We will introduce these parameters when we describe the algorithm.

We break the algorithm down into a number of steps. The first step is to compute what we called â€œa tiny amount of evidenceâ€ in Section [2](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-50002). Formally speaking, â€œa tiny amount of evidenceâ€ is referring to log-probability ratio, which we will shorten to log-prob ratio. You do not need to know what log-prob ratio to complete this assignment but an explanation is in Section [12](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-2400012) which you can read at your own time later on.

#### 3.1 Computing the log-prob ratio for one concentration measurement

Let c be a concentration measurement. We calculate the log-prob ratio of c using the formula:

Graphical user interface, text, application, email

Description automatically generated

Where Text

Description automatically generated, Graphical user interface, text, application, email

Description automatically generated,Graphical user interface, text, application, email

Description automatically generatedWe will use an example to illustrate how you can use this formula. This example uses values from the sample code. Let c be the first entry in conc\_signal which means c = 6.55. The Graphical user interface, text, application, email

Description automatically generated and  Text

Description automatically generatedvalues come from conc\_range , we have Graphical user interface, text, application, email

Description automatically generated=6.1 and Text

Description automatically generated= 6.9. The value Graphical user interface, text, application, email

Description automatically generatedof  is 0.5 which comes from conc\_var . Substituting these values into ([1](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-7001r1)), the log-prob ratio is .

You may wonder why we use the term log-prob ratio but you do not see any logarithm in the formula in ([1](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-7001r1)), this is because the logarithm has been cancelled out when ([1](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-7001r1)) is derived.

**3.2 Computing the log-prob ratio for conc\_signal**

Section [3.1](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-70003.1) explains how the log-prob ratio for one measurement in conc\_signal can be computed. The same procedure can be applied to each measurement in conc\_signal to compute the log-prob ratios for all the entries in conc\_signal . We will use the Python variable conc\_lpr to refer to this result. For the sample code, the conc\_lpr is expected to be:

[0.08, -0.16, 0.416, -0.08, 0.592, 0.672, -0.08, 1.376, 1.456, 0.272]

#### .3 Computing the CUSUM

The CUSUM is computed after conc\_lpr has been calculated. The CUSUM is a sequence of numbers (or list) which has the same number of entries as conc\_lpr . We will use conc\_cusum to denote this list. The first entry of conc\_cusum is related to the first entry of conc\_lpr by the following relationship:

Graphical user interface, text, application, email

Description automatically generated

For entries in conc\_cusum with indices k greater than or equal to 1, they are calculated using:

Graphical user interface, text, application, email

Description automatically generated

For the sample code, we have calculated conc\_lpr in Section [3.2](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-80003.2). By using that conc\_lpr , we can calculate conc\_cusum should be:

[0.08, 0, 0.416, 0.336, 0.928, 1.6, 1.52, 2.896, 4.352, 4.624]

Note that the definition in ([2](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-9006r2)) implies that if conc\_lpr[0] is negative, then conc\_cusum[0] will be zero.

**3.4 Determining whether the concentration signal indicates a fault**

We will use conc\_cusum (which can be thought of as the cumulative evidence that there is a fault) and the parameter conc\_cusum\_limit to determine whether there is a fault. We say the concentration measurements indicate that there is a fault if at least an entry in conc\_cusum is greater than or equal to conc\_cusum\_limit . For the sample code, there is a fault.

We are interested to determine two pieces of information from conc\_cusum and conc\_cusum\_limit . First, we want to determine the time at which the fault is detected. We use the sample code to illustrate how this time is to be determines. In this example, the entries at indices 5, 7, 8 and 9 of conc\_cusum are at or above conc\_cusum\_limit . The smallest of all these indices is 5 and we say that the fault is detected at time 5. Second, we want to determine the number of entries in conc\_cusum that are at or above conc\_cusum\_limit . For the sample code, the answer is 4.

The above example is for the case where there is a fault in conc\_cusum . In the case where a fault is absent, we will assign math.inf (i.e. infinity in the math library) to the time at which fault is detected. The number of entries in conc\_cusum that are at or above conc\_cusum\_limit should be zero.

Note that Figures 1c and 1d in Section [2](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-50002) plot the CUSUM. The red dashed lines in these figures indicate the level of conc\_cusum\_limit .

**3.5 Classifying the type of fault**

If the concentration measurements indicate that there is a fault, we want to determine whether the fault is limited to the concentration only or whether there is also a fault in the flow. We will describe the procedure to determine whether there is a fault in the flow in two steps.

In the first step, we compute the average of a number of consecutive entries in flow\_signal . The number of entries to be used is specified by the parameter flow\_control\_window . Another requirement is that the last of these consecutive entries is at the index at which the concentration fault is detected. Let us use the sample code as an illustration. Since the value flow\_control\_window is 3, this means we will use 3 consecutive entries in flow\_signal to compute the average. We know from Section [3.4](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-100003.4) that the concentration fault is detected at index 5, so the three consecutive entries have indices 3, 4 and 5. Hence, we compute the average of 23.9, 24.05 and 23.65, which is 23.87.

In the second step, we determine whether the average flow computed is within an acceptable range. We assume that the two numbers in flow\_range form a closed interval, i.e. inclusive of the ends. If the average flow is inside the interval, then there are no faults in the flow; otherwise there is. In the sample code, the closed interval is *[22.5,23.5]*and the computed average of 23.87 is not inside of this interval, so there is a fault in the flow.

Note that the above procedure requires that the time at which the concentration fault is detected is greater than or equal to the value in flow\_control\_window . You can assume that all the tests that we use for testing your code will meet this requirement.

**3.6 Validity checks**

The description above explains how the data ( conc\_signal and flow\_signal ) and algorithmic parameters ( conc\_range , conc\_var , conc\_cusum\_limit , flow\_range , and flow\_control\_window ) are used to determine whether there are faults. Note that the algorithmic parameters must be valid so that the computation can be carried out. We require that your code performs a number of validity checks before determining if there are any faults. For example, we assume that the algorithmic parameter flow\_control\_window is required to be a strictly positive integer. The following table states the requirements for the algorithmic parameters to be valid and what assumptions you can make when testing.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Algorithmic parameters | Requirements for the parameter to be valid | Assumptions you can make when testing or further explanation |
|  |  |  |
| conc\_range , flow\_range | Must have exactly 2 entries.  The value of the second entry in the list must be strictly greater than that of the first. | You can assume that the given conc\_range and flow\_range are of Python list type, their entries are numbers (int or float) which are strictly positive.  Examples of invalid inputs: [2.5,3.1,3.5] , [7.2,6.7] |
|  |  |  |
| conc\_var , conc\_cusum\_limit | Must be strictly positive. | You can assume that the given conc\_var and conc\_cusum\_limit are numbers (int or float).  Examples of invalid inputs: -1.5, 0 |
|  |  |  |
| flow\_control\_window | Data type must be int and its value is strictly positive | You can assume that the given flow\_control\_window is always a number. Examples of invalid inputs: -5, -5.2, 5.7. |
|  |  |  |
|  |  |  |

You can assume that we will only use valid data sequences conc\_signal and flow\_signal for testing. You can assume that these variables are Python lists whose entries are strictly positive numbers. You can assume that, for a given test case, these data sequences have the same length.

**4 Implementation requirements**

You need to implement the following six functions. These six functions work together to implement the the fault detection algorithm.

**The requirement is that you implement each function in a separate file.**This is so that we can test them independently, see Section [6](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-160006) on testing. We have provided template files, see Section [5](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-150005) on Getting Started.

In order to facilitate testing, you need to make sure that within each submitted file, you only have the code required for that function. Do not include test code in your submitted file.

1. def calc\_log\_prob\_ratio(measurement,conc\_range,conc\_var):
   * This function computes the log-prob ratio for one concentration measurement as described in Section [3.1](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-70003.1).
   * Note in the input measurement is referring to one concentration measurement.
   * This function should return one output which is the value of the log-prob ratio calculated.
   * You can test this function using the file test\_calc\_log\_prob\_ratio.py
2. def calc\_signal\_log\_prob\_ratio(conc\_signal,conc\_range,conc\_var):
   * This function computes the log-prob ratio for conc\_signal as described in Section [3.2](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-80003.2).
   * This function should return one output which is a list of log-prob ratios.
   * You can test this function using the file test\_calc\_signal\_log\_prob\_ratio.py
   * This function requires the function calc\_log\_prob\_ratio() . An import line has been included in the template file for you. Please do not change it.
3. def cusum(conc\_lpr):
   * This function computes the CUSUM for conc\_lpr as described in Section [3.3](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-90003.3).
   * This function should return one output which is a Python list.
   * You can test this function using the file test\_cusum.py
4. def detect\_fault\_in\_conc(conc\_cusum,conc\_cusum\_limit):
   * This function determines whether conc\_cusum indicates that there is a fault. The procedure has been described in Section [3.4](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-100003.4).
   * This function should return two (2) outputs:
     + The first output is the time at which the fault in conc\_cusum is detected.
     + The second output is number of entries in conc\_cusum that are at or above conc\_cusum\_limit
   * Note that the first input is either a non-negative integer or math.inf .
   * You can test this function using the file test\_detect\_fault\_in\_conc.py
5. def classify\_fault(conc\_signal,conc\_range,conc\_var,conc\_cusum\_limit,flow\_signal,flow\_range,flow\_control\_window):
   * This function should return one output which should be a python string. There are three possible outputs for this function.
     + If conc\_signal does not indicate a fault, the function should return the string 'no faults' . Note that if conc\_signal does not indicate a fault, you do not need to check whether there is a fault in flow\_signal .
     + If there is fault in conc\_signal but not in flow\_signal , the returned string should be 'conc fault only'
     + If faults are found in both conc\_signal and flow\_signal , the returned string should be 'flow and conc faults'
   * The procedure for determining the fault type has been described in Section [3.5](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-110003.5).
   * You can test this function using the file test\_classify\_fault.py
   * This function requires the functions calc\_signal\_log\_prob\_ratio() , cusum() , detect\_fault\_in\_conc() . Three import lines have been included in the template file for you. Please do not change them.
   * Note that this function will only require one of the two values returned by the function detect\_fault\_in\_conc() . You may want to use the method described in Section [4.1](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-140004.1) to handle this.
6. def detect\_fault\_main(conc\_signal,conc\_range,conc\_var,conc\_cusum\_limit,flow\_signal,flow\_range,flow\_control\_window):
   * The expected steps within the function detect\_fault\_main() are:
     + The function should first check whether all algorithmic parameters are valid. If any algorithmic parameter is invalid, the function should return the string 'invalid parameters' . It should not proceed to execute the next step. See Section [3.6](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-120003.6) on validity checks for the requirements on the algorithmic parameters.
     + If all algorithmic parameters are valid, then the function should proceed to classify the fault type and return a string indicating the fault type.
   * The function should return one output which is a string. The four possible outputs are: 'invalid parameters' , 'no faults' , 'conc fault only' , and 'flow and conc faults' .
   * This function can be tested by the following test files: test\_detect\_fault\_main\_0.py, test\_detect\_fault\_main\_1.py and test\_detect\_fault\_main\_2.py.
     + The test file test\_detect\_fault\_main\_0.py is based on the sample code.
     + The tests in test\_detect\_fault\_main\_1.py have valid parameters. The signals in these test cases are a lot longer than the signals in other tests. Test case 0 in the file correspond to Figures 1a and 1c. Test case 1 in the file correspond to Figures 1b and 1d.
     + The test file test\_detect\_fault\_main\_2.py contains a number of test cases where the algorithmic parameters are invalid. However, we have not included all the possibilities and you are asked to ensure that you code checks for all the requirements given in Section [3.6](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assignment1.htm#x1-120003.6).
   * This function requires the function classify\_fault() . An import line has been included in the template file for you. Please do not change it.

**4.1 An implementation note**

Let us say the function foo() returns two output values. You know that you can assign these output values to two variables using the syntax:

a, b = foo()

Let us say that you only need the variable a but not the variable b , there is no point in creating the variable b . You can use the following syntax:

a, \_ = foo()

so that only the variable a is created.

**5 Getting Started**

1. Download the zip file [assign1\_prelim.zip](https://www.cse.unsw.edu.au/~en1811/21T3/assign1/assign1_prelim.zip) and unzip it. This will create the directory (folder) named 'assign1\_prelim'.
2. Rename/move the directory (folder) you just created named 'assign1\_prelim' to 'assign1'. The name is different to avoid possibly overwriting your work if you were to download the 'assign1\_prelim.zip' file again later.
3. The zip file that we have provided contains 6 template files, 8 test files and 9 data files. We ask you to first browse through all the files provided including the test files. Note that the 9 data files are used by the test file test\_detect\_fault\_main\_1.py where you will be using longer conc\_signal and flow\_signal for testing. So, instead of cluttering the test file with a large trunk of numbers, we have put these numbers into files.
4. (Incremental development) 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 will not 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.

**6 Testing**

Test your functions thoroughly before submission.

You can use the provided test programs (files like test\_cusum.py etc.) to test your functions. Please note that each file covers a limited number of test cases. We have purposely not included all the cases because we want you to think about how you should be testing your code. You are welcome to use the forum to discuss additional tests that you should use to test your code.

We will test each of your files independently. Let us give you an example. Let us assume we are testing three files: prog\_a.py, prog\_b.py and prog\_c.py. These files contain one function each and they are: prog\_a(), prog\_b() and prog\_c(). Let us say prog\_b() calls prog\_a(); and prog\_c() calls both prog\_b() and prog\_a(). We will test your files as follows:

* We will first test your prog\_a().
* When we test your prog\_b(), we will test your prog\_b() together with our working version of prog\_a(). In this way, if your prog\_a() does not work for some reason, there is a chance that your prog\_b() may work and you may still receive marks for prog\_b().
* When we test your prog\_c(), we will test your prog\_c() together with our working version of prog\_a() and prog\_b().

**7 Submission**

You need to submit the following six files. Do not submit any other files. For example, you do not need to submit your modified test files. To submit this assignment, go to the Assignment 1 page and click the *Make Submission*tab.

* calc\_log\_prob\_ratio.py
* calc\_signal\_log\_prob\_ratio.py
* cusum.py
* detect\_fault\_in\_conc.py
* classify\_fault.py
* detect\_fault\_main.py,

**8 Assessment Criteria**

We will test your program thoroughly and objectively. This assignment will be marked out of 27 where 21 marks are for correctness and 6 marks are for style.

**8.1 Correctness**

The 21 marks for correctness are awarded according to these criteria.

|  |  |
| --- | --- |
|  |  |
| Criteria | Nominal marks |
|  |  |
| Function calc\_log\_prob\_ratio | 2 |
|  |  |
| Function calc\_signal\_log\_prob\_ratio | 3 |
|  |  |
| Function cusum | 4 |
|  |  |
| Function detect\_fault\_in\_conc | 4 |
|  |  |
| Function classify\_fault | 4 |
|  |  |
| Function detect\_fault\_main Case 1: There are invalid algorithmic parameters. | 3 |
|  |  |
| Function detect\_fault\_main Case 2: All algorithmic parameters are valid. | 1 |
|  |  |
|  |  |

**8.2 Style**

Six (6) 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 are 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.

**9 Assignment Originality**

You are reminded that work submitted for assessment must be your own. Sophisticated software is used to identify submissions that are unreasonably similar, and marks will be reduced or removed in such cases.

**10 Further Information**

* We will run Help Sessions for this assignment during Weeks 4-7. These consultations allow you to get one-on-one help on a first-come-first-serve basis. The timetable for the Help Sessions can be found on the course website.
* Use the forum to ask general questions about the assignment, but take specific ones to Help Sessions.
* Keep an eye on the course webpage notice board for updates and responses.

**11 Assignment conditions**

* Joint work is not permitted on this assignment.
  + This is an individual assignment. The work you submit must be entirely your own work: submission of work even partly written by any other person is not permitted.
  + Do not request help from anyone other than the teaching staff of ENGG1811 - for example, in the course forum, or in help sessions.
  + Do not post your assignment code to the course forum.
  + Assignment submissions are routinely examined both automatically and manually for work written by others.

*Rationale*: this assignment is designed to develop the individual skills needed to produce an entire working program. Using code written by, or taken from, other people will stop you learning these skills. Some other of your UNSW courses focus on skills needed for working in a team.

* The use of code-synthesis tools, such as GitHub Copilot, is not permitted on this assignment.

*Rationale*: this assignment is designed to develop your understanding of basic concepts. Using synthesis tools will stop you learning these fundamental concepts, which will significantly impact your ability to complete future courses.

* Sharing, publishing, or distributing your assignment work is not permitted.
  + Do not provide or show your assignment work to any other person, other than the teaching staff of ENGG1811. For example, do not message your work to friends.
  + Do not publish your assignment code via the Internet. For example, do not place your assignment in a public GitHub repository.

*Rationale*: by publishing or sharing your work, you are facilitating other students using your work. If other students find your assignment work and submit part or all of it as their own work, you may become involved in an academic integrity investigation.

* Sharing, publishing, or distributing your assignment work after the completion of ENGG1811 is not permitted. For example, do not place your assignment in a public GitHub repository after this offering of ENGG1811 is over.

*Rationale*: ENGG1811 may reuse assignment themes covering similar concepts and content. If students in future terms find your assignment work and submit part or all of it as their own work, you may become involved in an academic integrity investigation.

Violation of any of the above conditions may result in an academic integrity investigation, with possible penalties up to and including a mark of 0 in ENGG1811, and exclusion from future studies at UNSW. For more information, read the UNSW Student Code, or contact the course account.

**12 Some remarks on fault detection**

Roughly speaking, the probability ratio that you are calculating is:

The log-prob ratio is the logarithm of the above probability ratio. If the probability that there is a fault is higher, then the probability ratio is a number greater than 1 and the log-prob ratio will be positive. On the other hand, if the probability that there is not a fault is higher, then the log-prob ratio will be negative. This explains the role played by log-prob ratio in fault detection.

You can use one data point to calculate the above probability ratio but it may not be a good idea to decide whether there is a fault or not based on one data point alone. This is because the chance of getting a false alarm can be high. So, instead of using one sample, CUSUM makes use of the log-prob ratio from multiple data points to accumulate the evidence before making a decision. This will help to reduce the number of false alarms. However, you may wonder whether using multiple data points can be bad because it will take a longer time to detect fault. Yes, it will. Fault detection is ultimately a subtle problem that requires you to balance between false alarms, not missing the fault and detecting the fault quickly.

**References**

[1] G. Riss et al., Detection of Water Quality Failure Events at Treatment Works Using a Hybrid Two-stage Method with CUSUM and Random Forest Algorithms. Water Supply, 2021. <https://iwaponline.com/ws/article/21/6/3011/80771/Detection-of-water-quality-failure-events-at>