ENGG1811 22T3 Assignment 1: Processing vibration signals

> Due date: 5pm, Friday 28 October (week 7). Submissions will generally not be accepted after 5pm, Wednesday 2 November, 2022. Late submissions will be penalised, see Sec. 8.3.

Version and change log

This version: v1.04 on 12 Oct 2022.

Updates:

- (30/09/22) Please note that any updates or corrections will be summarized here.
- (30/09/22) I briefly talked about the Assignment in the lecture on Friday of Week 3. That segment starts at 1:33:18 in the recording.
- (02/10/22) The formula for damping ratio ξ had a square-root missing on the right-hand side in v1.00. The square-root, which appears in red, has now been added to the formula for ξ in Sec. 3.4
- (04/10/22) The earlier versions stated that the last possible time to submit the assignment would be 5pm on 2 October. That should have been 2 Novemeber.
- (09/10/22) Corrected a typo in Sec. 4.
- (12/10/22) An assumption on the `resp_list` has been added in Sec 3.1 so that you do not consider possible flat troughs. The addition is typeset in red.

1 Introduction

Vibration is a everyday pheonomenon, e.g. vehicles driving over a bridge can cause the bridge to vibrate. Therefore, many engineering disciplines pay attention to how materials or structures respond to vibration. Sometimes engineers may conduct an experiment to understand the vibration response. Figure 1 shows an experimental setup to study the vibration of a beam due to an impact hammer. The response to a hammer is typically an oscillatory signal that decays over time, see Figure 2 for an example.

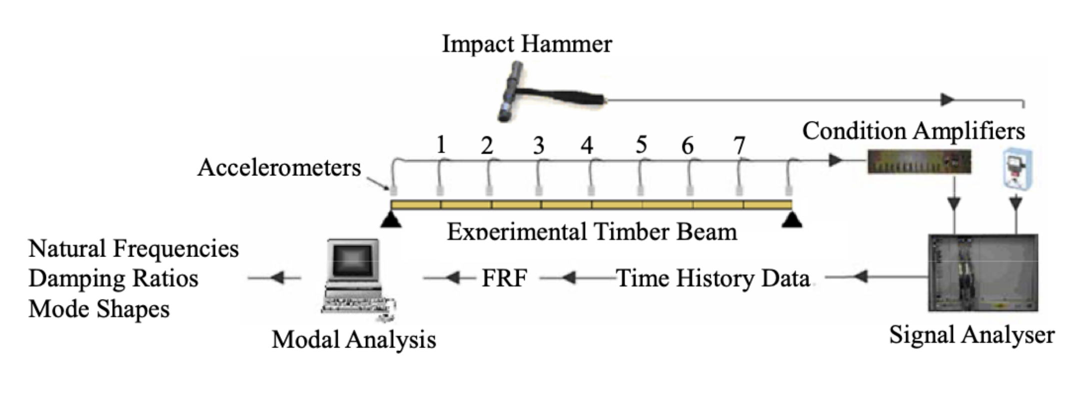

Figure 1. Illustrating the setup of vibration testing [1].

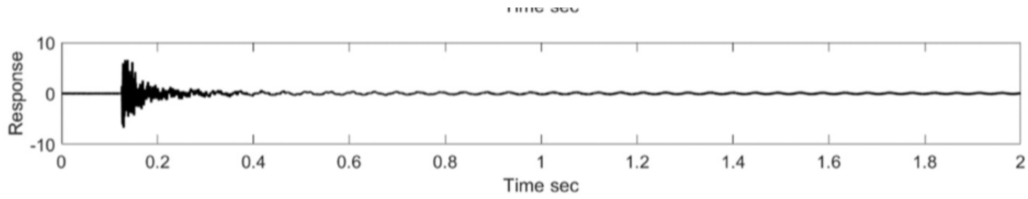

Figure 2. A sample response to an impact hammer [2].

This assignment is *inspired* by the impact hammer experiment [1] [2]. In this assignment, you will write Python programs to process data sequences to determine the amount of damping and how well the experiment had been conducted.

Note that we chose the word inspired earlier because we have adapted the the impact hammer experiment [1] [2] by simplifying and liberally changing many 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 in engineering in general.

1.1 Learning objectives

By completing this assignment, you will learn:

- To apply basic programming concepts of variable, assignment, data types, conditional, functions, loops and import.
- To use the Python data types: list, float, int, string 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 choice of variable names, comments and documentation etc.
- To get a practice on software development, which includes incremental development, testing and debugging.

1.2 Prohibition and academic honesty

You are allowed to use the math library for this assignment. However, you are **not** allowed to use any other libraries that are not written by you, e.g. numpy, scipy etc.

This is an individual assignment (i.e., no group work) and **must be your own work**. You should read the **assignment conditions** section carefully.

2 Some intuition behind vibration signal analysis

In this section, we will give you some intuition behind vibration analysis so that you can understand why we ask you to program certain methods.

Figures 3A and 3B show two different responses to a hammer hit. The signal in Figure 3B dies down faster as the structure has a higher amount of damping. We have marked the troughs of the signals with red crosses in both figures. You can see the signal in Figure 3B has fewer troughs and the amplitudes of the troughs decrease faster than those in Figure 3A. Therefore, in this assignment, you will program a method to identify troughs and to determine how fast the amplitudes of these troughs are decreasing.

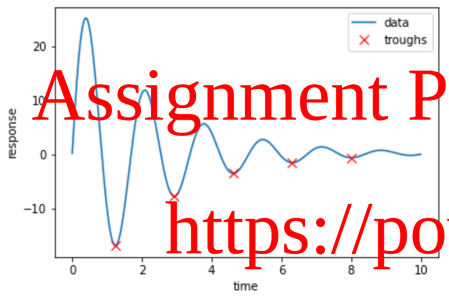

Figure 3A: Lower damping.

Figure 3B: Higher damping.

The analysis method that we are using assumes that there is only one hit from the impact hammer. However, sometimes the experiment does not go according to plan which results in two hammer hits. Figure 4 shows the response when there are two hits. Unfortunately, we cannot use such data with our analysis method. Therefore, you will be programming a method to identify whether the data are produced from two hammer hits. You will be using the information on the troughs to help you to do that. You can see that the troughs are roughly equally spaced in time in Figure 3A when there is only one hammer hit, but this is no longer the case in Figure 4. You will be using this observation in the computer program to help you to identify whether there is a second hammer hit.

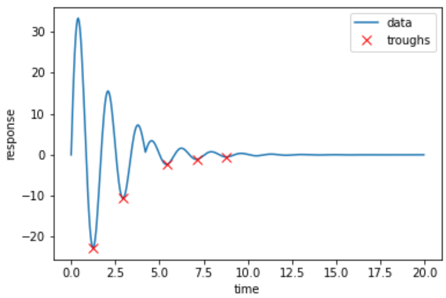

Figure 4. Double hammer hit.

Two remarks before we move on. First, you may notice that there is a trough in Figure 4 which looks like a kink but is not marked with a red cross. This is because we only consider certain troughs and this will be explained in Sec. 3.1. Second, you may ask why we choose to use troughs rather than peaks. You can certainly use peaks but we have discussed peaks in the lecture, so we choose to use troughs just to give you a tiny bit more work to do.

3 Requirements for processing vibration signals

This section describes the requirements on the algorithm on vibration signal analysis 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 weeksâ€™ 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.

```
# The response to an impact hammer
# time_list is a list of sampling time instants
# resp_list is the response at the sampling time instants
time_list = [0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5,
             5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5,
             10.0, 10.5, 11.0, 11.5, 12.0, 12.5, 13.0, 13.5, 14.0, 14.5,
             15.0, 15.5, 16.0, 16.5, 17.0, 17.5, 18.0, 18.5, 19.0, 19.5,
             20.0]
resp_list = [0.0, 0.83, 1.14, 0.79, 0.0, -0.75, -1.03, -0.71, -0.0, 0.68,
             0.93, 0.64, 0.0, -0.61, -0.85, -0.58, -0.0, 0.55, 0.77, 0.53,
             0.0, -0.5, -0.69, -0.48, -0.0, 0.45, 0.63, 0.43, 0.0, -0.41,
             -0.57, -0.39, -0.0, 0.37, 0.51, 0.35, 0.0, -0.34, -0.46, -0.32
             -0.0]

# A parameter used by the algorithm
trough_amp_upper_bound = -0.5

# Call the function to process the vibration signal
import process_vibration_signal as process
your_answer = process.processs_vibration_signal(time_list, resp_list, troug
```


In the sample code, the Python lists `time_list` and `resp_list` contain, respectively, the sampling time instants and the response to an impact hammer at the sampling time instants.

In addition to the two lists, your code will make use of an algorithmic parameter `trough_amp_upper_bound` (see the sample code) for the computation. The algorithm will also make use of three constants and we will introduce them in Sec. 3.2 and Sec 3.5. We will introduce the parameter and constants when we describe the algorithm.

We break the algorithm down into a number of steps in Secs. 3.1-3.5.

3.1 Find the troughs

The aim of this step is to determine the times and amplitudes of the troughs. Figure 5 plots the response `resp_list` in the sample code against `time_list`. You can see that there are 5 troughs in the data and 4 of them are marked by red crosses.

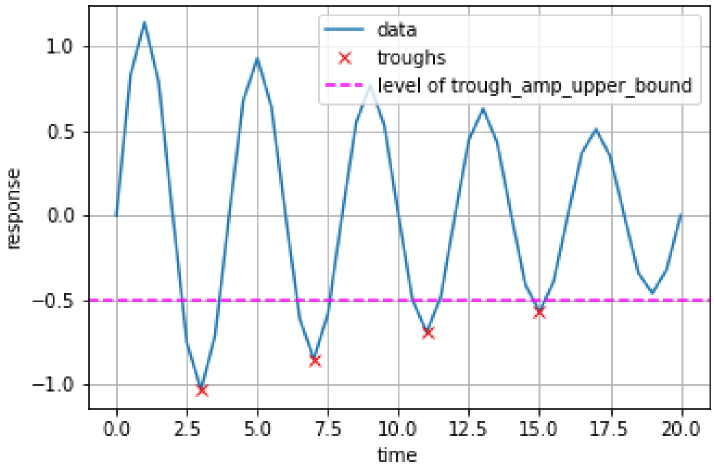

Figure 5. Response against time for the data in the sample code.

We will not use all the troughs. We will only use those troughs whose amplitude is less than the level specified by the parameter `trough_amp_upper_bound`. The magenta dashed lines in Figure 5 is at the level of the parameter `trough_amp_upper_bound` which equals to -0.5 for the sample code. The amplitudes of the first 4 troughs are less than `trough_amp_upper_bound` so they are accepted. The fifth trough which has an amplitude -0.46 is *not* less than `trough_amp_upper_bound`, so it is not accepted. The rationale for using `trough_amp_upper_bound` is to eliminate troughs with small amplitudes as they can be heavily influenced by noise.

For the sample code, the acceptable troughs appear at times 3.0, 7.0, 11.0 and 15.0; and the corresponding amplitudes are -1.03, -0.85, -0.69 and -0.57. We will use the Python variables `trough_time_list` and `trough_amp_list` to refer to these results. Specifically, for the sample code, `trough_time_list` and `trough_amp_list` are, respectively:

```
[3.0, 7.0, 11.0, 15.0]

[-1.03, -0.85, -0.69, -0.57]
```

(Added 12/10/2022) You can assume that when we test your work, any two consecutive entries in `resp_list` will have different values. This is so that you do not have to consider flat troughs.

3.2 Checking whether the data is usable

The aim of this section is to check whether `trough_time_list` and `trough_amp_list` will be usable for computing the amount of damping later on. We will conduct two checks to ensure that: (1) There are enough acceptable troughs. (2) There is not a second hammer hit. If both checks are passed, then the algorithm will proceed to compute the amount of damping; otherwise, the algorithm should terminate. We now explain these checks.

The first check is to ensure that there is a minimum number of acceptable troughs. We require that there are 4 or more acceptable troughs in the given data. This check can be done using either `trough_time_list` or `trough_amp_list`. The data in the sample code has 4 acceptable troughs so they pass this check.

If the results in `trough_time_list` and `trough_amp_list` pass the first check, we will proceed to check whether there is a second hammer hit in the data. Sec. 2 points out that we can determine the existence of a second hammer hit by checking whether the troughs are regularly spaced in time or not. You will perform this check using `trough_time_list`.

We will use the data for Fig. 4 for this illustration because those data contain a second hammer hit. We assume that we have already determined the `trough_time_list` for these data and it is give by:

```
[1.25, 2.95, 5.4, 7.1, 8.8]
```

The algorithm to determine whether there is a second hit is:

Steps of the algorithm	**Expected outcome if `trough_time_list` is `[1.25, 2.95, 5.4, 7.1, 8.8]`**
(1) Compute the difference of consecutive entries of `trough_time_list`	`[1.7 , 2.45, 1.7 , 1.7 ]` Note: 2.95 - 1.25 = 1.7, and 5.4 - 2.95 = 2.45, etc.
(2) Determine the maximum and minimum value in the list obtained in the last step	Maximum is 2.45. Minimum is 1.7.
(3) Compute the ratio of the maximum to the minimum.	$\frac{2.45}{1.7} = 1.44$
(4) If the ratio computed in the last step is larger than a given threshold, then we say that the troughs are irregularly spaced and consequently there is a second hammer hit	Assuming that the threshold is 1.1, then the outcome is that there is a second hammer hit.

If you apply the above algorithm to the `trough_time_list` from the sample code, the ratio that you will obtain at Step (3) is 1. If we assume that the threshold in Step (4) is 1.1, then the conclusion is that there is no a second hit.

3.3 Compute the ratio of successive trough amplitudes

This computation assumes that the data have passed both checks in Sec 3.2. Our goal is to determine how quickly the amplitudes of the troughs are decreasing, so we will compute the ratio of the successive pairs of trough amplitudes. We will use the sample code to describe the calculations that you need to do.

We start from the `trough_amp_list` from the sample code, which is `[-1.03, -0.85, -0.69, -0.57]`, you need to compute:

$$\frac{-0.85}{-1.03}, \frac{-0.69}{-0.85}, \frac{-0.57}{-0.69}$$

For this description, we assume that these results are stored in a Python list with the name `trough_amp_ratio_list`. For the sample code, `trough_amp_ratio_list` is:

```
[0.8252, 0.8118, 0.8261]
```

where the entries are displayed to 4 decimal places.

3.4 Computing geometric mean and damping ratio

This computation assumes that the `trough_amp_ratio_list` has already been calculated. The goal of this computation is to calculate the geometric mean of the numbers in `trough_amp_ratio_list`. As a reminder, the geometric mean of n numbers $a_1, a_2, \ldots, a_n$ is given by

$$(a_1 \, a_2 \, \ldots \, a_n)^{\frac{1}{n}}$$

For the sample code, the expected value of the geometric mean of the numbers in `trough_amp_ratio_list` is 0.8210 to 4 decimal places. Let us use the mathematical symbol g to denote the calculated geometric mean. We will then use g to calculate the damping ratio ξ in two steps:

$$r = \frac{\log(g)}{2\pi}$$

$$\xi = \sqrt{\frac{r^2}{1 + r^2}}$$

[Correction: The red square-root sign was missing in version 1.00. It has been added in v1.01.]

Note that the above logarithm is to the base e. Intuitively, a large damping ratio means the oscillation will die down faster. You do not need to understand how the above equations above are derived.

3.5 Validity check

It is a standard programming practice to check the validity of the inputs supplied by the user. There are at least two reasons for that. First, invalid inputs can cause your program to crash, which is not desirable. Second, you have specific requirements for the inputs but the users may overlook these requirements when they enter the inputs, so the best practice is to check that the requirements are met.

For this assignment, you need to check the validity of `trough_amp_upper_bound`, `time_list` and `resp_list`.

The input `trough_amp_upper_bound` is valid if the following two conditions are satisfied: (1) It must be of either `float` or `int` type; and (2) Its value must be less than zero. Some examples of valid input are: `-1.2`, `-2` etc.; and some examples of invalid inputs are: `0.5`, `[-0.5]`, `False`.

For the inputs `time_list` and `resp_list` to be valid, they must satisfy these two conditions: (1) Each list must have 9 or more entries; and (2) The two lists must have the same number of entries. An example of invalid inputs is:

```
time_list = [0, 0.1, 0.2, 0.3, 0.4]
resp_list = [0, 0.7, 1.2, 1.7, 1.8]
```

because the number of entries in `time_list` is less than 9 and the same problem with `resp_list`. Another example of invalid inputs is:


```
# Note: time_list has 10 entries. resp_list has 11 entries
time_list = [0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45]
resp_list = [0.2, -0.5, -1.3, -0.7, 0.15, -0.4, -1.0, -0.5, 0.1, -0.2, -0.4
```


where the two lists have different number of entries. An example of valid inputs are:

```
# Note: time_list has 10 entries. resp_list has 11 entries
time_list = [0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4]
resp_list = [0.2, -0.5, -1.3, -0.7, 0.15, -0.4, -1.0, -0.5, 0.1]
```

where each list has least 9 entries, and both lists have the same number of entries.

You can assume that when we test your code, the given `time_list` and `resp_list` are of Python list type, their entries are numbers (i.e., int or float).

Remark: You may ask why we ask for at least 9 entries in `time_list` and `resp_list`. This is because this is the minimum number of data points for the data to have 4 troughs.

4 Implementation requirements

You need to implement the following six functions. These six functions work together to implement the vibration signal processing algorithm.

The requirement is that you implement each function in a separate file. This is so that we can test them independently, see Sec. 6 on testing. We have provided template files, see Sec. 5 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 find_trough_time_amp(time_list, resp_list, trough_amp_upper_bound):`
   - This function determines the `trough_time_list` and `trough_amp_list` from the three function inputs.
   - The algorithm for this function has been described in Sec. 3.1.
   - This function should return two outputs in this order: `trough_time_list` , `trough_amp_list` .
     - Note that this order is important for us to test your code.
   - You can test this function by using the file `test_find_trough_time_amp.py`
2. `def calc_trough_amp_ratio(trough_amp_list):`
   - This function computes the ratio of the amplitudes of the consecutive troughs as described in Sec. 3.3
   - This function should return one output which is a list of ratio. An example of the output `trough_amp_ratio_list` is in Sec. 3.3
   - You can test this function by using the file `test_calc_trough_amp_ratio.py`

3. `def calc_geometric_mean(a_list):`
   - This function has one input which is a list of numbers
   - It computes the geometric mean of the numbers in the given input `a_list`
   - It returns one output which is a `float` whose value is the computed geometric mean
   - You can test this function by using the file `test_calc_geometric_mean.py`
4. `def calc_damping_ratio(trough_amp_list):`
   - This function has one input which is a list that plays the role of `trough_amp_list`
   - It returns one output which is a `float` whose value is the damping ratio
   - You can find an example of this computation in Sec. 3.3 and Sec 3.4
   - Note that the computation steps to obtain the damping ratio from `trough_amp_list` will require you to compute the ratio of consecutive amplitudes and geometric ratio. We require that you **must** make use of the functions `calc_trough_amp_ratio()` and `calc_geometric_mean()` in your implementation.

- The template file has two import statements to load `calc_trough_amp_ratio` and `calc_geometric_mean`. Do not delete those two import lines.
- You can test this function by using the file `test_calc_damping_ratio.py`

5. `def exist_second_hammer_hit(trough_time_list):`

   - This function has one input which is a list that plays the role of `trough_time_list`
   - The aim of the function is to determine whether the trough times given in the funciton input indicate that there is a second hammer hit. The algorithm has been described in Sec. 3.2
   - This function returns one output which is of datatype `bool`. The output should be `True` if there is a second hit, otherwise it should return `False`.
   - This function requires a threshold, see Step (4) in the algorithm in Sec. 3.2. You can assume that the value of threshold is 1.1. Since this threshold is a constant, we expect that you use good programming style to specify it.
   - You can test this function by using the file `test_exist_second_hammer_hit.py`

6. `def process_vibration_signal(time_list, meso_list, trough_amp_upper_bound):`

   - The expected steps within this function are:
     - The function should first check whether all the three inputs are valid as explained in Sec. 3.5. If any of the inputs is invalid, the function should return the string `'invalid input'` and it should not proceed to execute the next step.
     - If all the three inputs are valid, the function should proceed to determine `trough_time_list` and `trough_amp_list`
     - The function should check whether there are sufficient number of troughs as explained in Sec 3.2. If the number of troughs is insufficient, then the function should return the string `'too few troughs'` and it should not proceed to the next step.
     - If there are sufficient number of troughs, the function should determine whether there is a second hit as explained in Sec 3.2. If there is a second hit, the function should return the string `'second hammer hit'` and it should not proceed to execute the next step.
     - If there is not a second hammer hit, the function should compute the damping ratio and return its value as the output.

   - This function should return one output. This output can be a string or a float. If the output is a string, it can be either `'invalid input'`, `'too few troughs'` or `'second hammer hit'`. If the output is a float, then its interpretation is a damping ratio.
   - You can test this function by using the files `test_process_vibration_signal_0.py` and `test_process_vibration_signal_0.py`

        - The tests in `test_process_vibration_signal_0.py` are expected to return a string.
        - The tests in `test_process_vibration_signal_1.py` are expected to return a float.

    - This function **must** make use the functions `find_trough_time_amp()`, `exist_second_hammer_hit()` and `calc_damping_ratio()`. Three import lines have been included in the template for this purpose, do not delete them.
    - This function requires two constants and we expect that you use good programming style to specify them.
        - The first constant is the number 9 which is used to determine the minimum required length of `time_list` and `resp_list`, see Sec. 3.5.
        - The second constant is the number 4 which is used to determine the minimum number of acceptable troughs, see Sec 3.2.

Notes:

1. Please note that each test 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. In particular, we want to point out the tests in `test_process_vibration_signal_0.py` certainly do **not** cover all cases. You are welcome to use the forum to discuss additional tests that you should use to test your code.
2. If your function returns a float or a list of floats, do *not* round their values. In addition, your program should do all calculations using the full precision, do *not* round the numbers in any intermediate step. In Section 3, we round the numbers to make the text easier to read.

5 Getting Started

1. Download the zip file 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, 7 test files and 6 data files. We ask you to first browse through all the files provided including the test files. Note that the 6 data files are used by the test files `test_find_trough_time_amp.py`, `test_process_vibration_signal_0.py` and `test_process_vibration_signal_1.py` where you will be using longer `time_list` and `resp_list` 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 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. Do not `print` your answers.

6 Testing

Test your functions thoroughly before submission. You can use the provided test files to test your functions.

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.

- `find_trough_time_amp.py`
- `calc_trough_amp_ratio.py`
- `calc_geometric_mean.py`
- `calc_damping_ratio.py`
- `exist_second_hammer_hit.py`
- `process_vibration_signal.py`

8 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.

8.1 Correctness

The 20 marks for correctness are awarded according to:

Functions	Nominal marks
`find_trough_time_amp`	4
`calc_trough_amp_ratio`	3
`calc_geometric_mean`	2
`calc_damping_ratio`	3
`exist_second_hammer_hit`	3
`process_vibration_signal` Case 1: Invalid input	2
`process_vibration_signal` Case 2: Not enough troughs	1
`process_vibration_signal` Case 3: Second hammer hit	1
`process_vibration_signal` Case 4: Return damping coefficient	1

8.2 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:

- Code layout: The first section of the code should be the docstring, followed by import statements (if any), definition of constants (if any) and finally your code. Refer to style guide for ENGG1811 for details.
- Use of docstring for documentation to identify purpose, author, date, method and data dictionary
- Use of sensible comments to explain what you are doing
- Use of meaningful variable names where applicable
- Code style to specify constant values
- Ensure that code does not have long lines to enhance readability

We have prepared a style guide for ENGG1811 to help you to meet the style requirements. Note that we have divided the guide into two sections: mandatory and recommended requirements. The mandatory requirements (which have been listed above) will be used when marking the assignment. However, we would like to see that you use the recommended requirements as they will make your code more readable by you, your marker and others.

8.3 Late penalty

Late submissions will be penalised at 0.2% per hour. For example, if your submission is 24 hours and 20 minutes late, then it will be considered to be 25 hours late and will receive a penalty of $25 \times 0.2\% = 5\%$. If your submission is judged to be worth 20 marks, then the actual mark that you will receive is $20 \times (100 - 5)\% = 19$ marks.

9 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. If you really need to show your code for your forum question, you should mark your question **private**.
- Keep an eye on the course forum for updates and responses.

10 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.

References

1. B. Samali, J. Li, U. Dackermann and F. C. Choi. Vibration-based Damage Detection for Timber Structures in Australia. in *Structural Health Monitoring in Australia*, pp. 81 - 108
2. A. Chiniforush, M. Makki Alamdari, U. Dackermann, H.R. Valipour and A. Akbarnezhad. 'Vibration behaviour of steel-timber composite floors, part (1): Experimental & numerical investigation'. *Journal of Constructional Steel Research*, vol. 161, pp. 244 - 257, 2019.