FIT3139: Assignment 1

(Due by 11:59pm, Sunday, 10 Sept 2017)

[Weight: $10 = 5 + 5$ marks.]

This assignment has two parts. The objective of the first part is to motivate the effect of computer arithmetic on numerical calculations which are common to problems in computational science; this part is worth 5 of 10 marks. The second part of the assignment will motivate an exciting real-world scientific problem/exercise; this part is worth the remaining 5 of 10 marks. Follow the instructions given under the respective parts and prepare answers/solutions to the assignment.

Follow these procedures to submit this assignment

The assignment must be submitted *online* via Moodle, and should follow the following procedure:

- Accept the Electronic Plagiarism Statement for this Assignment. All your scripts/program will be scanned using MOSS (a plagiarism detection software). Read Monash Student Academic Integrity policy for consequences of plagiarism.
- All your scripts and reports **MUST** contain your name and student ID.
  - **You are free to program the assignment in either MATLAB or Python.**
  - Your submitted archive must extract to a directory named as your student ID.
  - This directory should contain a subdirectory for each of the two questions in the assignment, named as `q1/`, and `q2/`.
  - Within each of those subdirectories the contents include `MATLAB`/`Python` scripts and corresponding `PDF` reports or plain text output depending on what is asked of you for that question.
  - When submitting scripts and reports, choose file names that identify the sub-question. (Eg. `q1a_script.py`, or `q1b_report.pdf`, or `q2_script_driver.m`, or `q2_output.txt`)
- Submit your zipped file electronically via Moodle.

Supporting Material

For part 2 of this assignment, use the supporting material uploaded on Moodle under `My Assignments` tab.

Part 1

(a) A common computation in many applied mathematical problems involves the evaluation of the function $\log(1+x)$,* which is often represented as `log1p`(x). Indeed most math libraries (including MATLAB and Python) define an **in-built function** `log1p`(x) for this purpose, over and beyond using the standard **in-built logarithm function** `log`$(1+x)$.

One way to compute this function is by using the following Taylor series expansion over infinite terms:

$$\texttt{log1p}(x) = \log(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots + \frac{(-1)^{n+1}x^n}{n} + \cdots \tag{1}$$

- Write your own function `my_log1p`(x) that accepts a value x as input and evaluates the above series expansion until EITHER 1,000,000 terms in the expansion have been computed OR when the approximation to $\log(1+x)$ has **changed** by less than 10^{-6}.
- Your function should output the value it evaluated, the number of terms in the expansion it computed before it returned, the relative error compared to the the in-built `log1p`(x).
- Test and plot the accuracy of your function (in terms of the relative error) for varying values of x, including the values of x **less than** the `machine epsilon` of your floating-point representation.

What should you submit for this question?

Your submission should include your script that drives your function `my_log1p`(x) for various values of x and plots the relative errors.

Your submission must also include a report. Your report should **briefly** comment on the values of x that lead to the **most inaccurate** results in terms of relative error, and identify/describe the advantages/limitations of using the above expansion. It should also specify the values of x for which you would use `my_log1p`(x) over the in-built **logarithm function** `log`$(1+x)$, and *vice versa*. Note, this report should be separate from the script, and submitted in a PDF format, should be succinct and comprehensibly written.

(b) Consider **another** way of evaluating `log1p`(x). For any given x, one can **choose another value** y such that

$$\frac{1+y}{1-y} = 1 + x.$$

- Work out the expansion of $\log(1+x)$ as $\log\left(\frac{1+y}{1-y}\right)$.
- Write another function `my_another_log1p`(x) that evaluates `log1p`(x) using this new expansion.†
- Test and plot the accuracy of this new function (in terms of relative error wrt the in-built `log1p`(x)).

*Here log() is the natural logarithm function to the base e.

†The termination of this series expansion is similar to what you have done in part 1(a).

What should you submit for this question?

Your submission should contain a script that drives `my_another_log1p`(x) for varying values of x and plots the relative errors. Your report should contain a fully worked out expansion for $\log(1+x)$ (with intermediate steps that facilitated the expansion). Compare and contrast the results with those analysed in part 1(a). Your report should comment which of the two approaches lead to greater accuracy and why? Suggest any further observations you might have about the comparison between the these approaches.

Part 2

This part of the assignment motivates an exciting real-world computational science problem. To address this exercise, refer to supporting material on Moodle.

Background:

A common task in many areas of science is to ask if two shapes (each described as a set of points/vectors/coordinates in some space) are similar or not. In the three-dimensional (3D) world we live in, such a task is notably common and extremely useful and has routine applications in the fields of Robotics, Chemistry, Engineering, Structural Biology amongst many others.

A way to recognize shape similarity is to spatially rotate and translate one shape with respect to another so that they can be *superimposed* on top of each other. One useful and practical measure of similarity (*after superposition*) is the *root-mean-squared deviation* (RMSD) of corresponding points in the two shapes. Here the assumption is that the two shapes have the same number of points, and that there is a *one-to-one correspondence* between the points describing the shapes being compared. Once this measure of similarity is defined, the goal is to find the **optimal superposition** that minimizes the RMSD between two shapes.

A remarkable aspect about this seemingly hard computational problem is that it can be solved analytically using an extremely elegant and efficient method described in the article `rmsdpaper.pdf` included as a part of supplementary material supporting this assignment on Moodle. In fact, the time-complexity of this method/algorithm is $O(N)$, that is, linear in the number of points N within each shape being superposed.

The suggested article is understandably difficult to digest, so included in the supplementary material is a handwritten guide that should take you through the computation of RMSD *step-by-step* in easily understood terms. So, all you will need to complete this exercise will be to go through the handwritten guide provided: See the document `guideToRMSDComputation.pdf` in the supporting material directory on Moodle.

The supporting material includes **two** plain text files, `A.pdb` and `B.pdb`, which correspond to the three-dimensional coordinates of the experimentally determined protein Human *Haemoglobin*.[‡] In this exercise you will work with the information given in these files, especially using the information in the following range of column numbers (**one column in the file is one ASCII character wide.** The numbers below indicate the position of the column/character from the start of the line, counting from 1):

Columns 13 to 16 These columns give the atom names;

[‡]See https://pdb101.rcsb.org/motm/41. Note, Max Ferdinand Perutz won a Nobel Prize in Chemistry (in 1962) for the first X-ray crystal structure of Haemoglobin. Read his interesting Nobel lecture.

Columns 31 to 38 These columns give the x *component* of the (x, y, z) coordinate of the atom identified above.

Columns 39 to 46 These columns give the y *component* of the (x, y, z) coordinate of the same atom.

Columns 47 to 54 These columns give the z *component* of the (x, y, z) coordinates of the same atom.

Ignore the rest of the columns in each line as they are unnecessary for this specific exercise.

Note, these (x, y, z) coordinates are given in Angstrom (Å) units. 1 Å is 10^{-10} meters. You don't have to worry about this unit, but just be aware that RMSD (which is a descriptive statistic of error after superposition) is consequently measured in Angstrom (Å) units.

Your tasks on hand for this exercise

Below are the set of tasks you need to address to solve this exercise:

1. First, write a function that takes a (structure's) *file name* as an argument, and parse this file to extract all the coordinates in the rows/lines corresponding to atom names identified as ' `CA` '. That is, your script will read the (x, y, z) coordinates specified in the lines/rows of a given file, **only when the column range 13 to 16 contains the string ' `CA` '**. Upon parsing the given file, your function should return the extracted list of (x, y, z) coordinates as an $n \times 3$ matrix, where n is the number of `CA` atoms in the file. (You **cannot** use any in-built parser/reader for *.pdb format files.)

   Pass the two given structures (`A.pdb` and `B.pdb`) through your function, one at a time, and extract their (x, y, z) coordinates, storing them in two separate matrices of sizes $n_1 \times 3$ and $n_2 \times 3$, where n_1 and n_2 are the number of `CA` atoms in the respective files.

2. Then, write another function that accepts as arguments two (smaller) matrices of **equal size** $k \times 3$, where k is an arbitrary integer ≥ 6. Your function should then compute the RMSD using these two matrices. Each matrix passed into this function is a **continuous stretch** (of size k) of (x, y, z) coordinates coming from each of the two larger matrices extracted from `A.pdb` and `B.pdb` files in step 1 above. In this exercise, we will use the term '***fragment***' to indicate any continuous stretch of `CA` coordinates of a structure. Using these two smaller matrices (corresponding to the coordinates of a pair of equal-sized fragments, or ***fragment pair***, in short), compute the RMSD between this fragment pair. To be able to compute RMSD of a given fragment pair, use the guide to RMSD computation provided in the supporting material. (Refer guideToRMSDComputation.pdf on Moodle.) Your program must return the computed RMSD value.[§]

3. Your final task is to use the two functions developed above to produce the list of **well-superposable fragment pairs** between structures `A.pdb` and `B.pdb` of size $k = 9$. A fragment pair is said to be **well-superposable** if the RMSD between the pair of fragments is strictly less than 1.5 Å.

[§]To sanity check your output, the supporting material includes the file: `rmsdsanitycheck.txt`. You should be able to check if your RMSD function is working correctly based on the examples given in this file.

What should you submit for this question?

You will have to submit

- An entire working program along with its companion functions to parse the structure files and compute the RMSD.
- An output file (in plain ASCII text) generated by your program with the following information:
  - The first line of this output should be the **the total number of well-superposable fragment pairs** between the two structures. **Do not add any other text. This line should simply be one number (integer).**
  - The subsequent lines should list the identified well-superposable fragment pairs, each of them listed in a separate line, denoted using **three** numbers of the form

    $$i \quad j \quad rmsd$$

    where i and j are the start indexes (counting from 1) of two fragments in the structures `A.pdb` and `B.pdb` respectively, and $rmsd$ is the RMSD after superposition of this fragment pair. Do not add any other text. Each line should simply list the three numbers mentioned above, separated by a white space: the first two numbers will be integers, and the last is a real number stated to 3 places after the decimal point.

No further explanation or PDF report is required for this part of the assignment.

```
-=o0o=-
  END
-=o0o=-
```