Assignment 1

COMP SCI 2ME3 and SFWR ENG 2AA4

1 Dates and Deadlines

Assigned: January 4, 2018

Part 1: January 22, 2018

Receive Partner Files: January 28, 2018

Part 2: January 31, 2018

Last Revised: February 6, 2018

All submissions are made through git, using your own repo located at:

`https://gitlab.cas.mcmaster.ca/se2aa4_cs2me3_assignments_2018/[macid].git`

where `[macid]` should be replaced with your actual macid. The time for all deadlines is 11:59 pm. If you notice problems in your Part 1 `*.py` files after the deadline, you should fix the problems and discuss them in your Part 2 report. However, the code files submitted for the Part 1 deadline will be the ones graded.

2 Introduction

The purpose of this software design exercise is to write a Python program that creates, uses, and tests a simple Abstract Data Type (ADT) that stores a sequence of values. The Sequence ADT will allow a programmer to create instances of the datatype `SeqT`. In this assignment, the SeqT type will be used to create a curve in Cartesian space and the values between the curve's data points will be determined using both interpolation and regression. The program will consist of two modules and a test driver program.

All of your code (all files) should be documented using doxygen. All of your reports should be written using LaTeX. Your code should follow the given specification as closely as possible. In particular, you should not add public methods that are not specified and you should not change the number or order of parameters for methods. If you need private methods, please use the Python convention of "dunder" names with double underscores (`__methodName__`).

Part 1

Step 1

Write a first module that creates the Sequence ADT. It should consist of Python code in a file named `SeqADT.py`. The module should define a class SeqT, which contains the following class methods that define the external interface:

- A constructor (`SeqT()`) that takes no arguments and creates an object whose state consists of an empty sequence.

- A method named `add(i, v)` that takes the following inputs: i and v, where i (an integer) is the index where the value v (a real) should be added to the sequence. Values can only be added within the existing sequence, or immediately after the last entry in the existing sequence.

- A method named `rm(i)` that takes one argument: i (an integer). A call to this method modifies the sequence so that the entry at index i is removed. The length of the list will decrease by 1.

- A method named `set(i, v)` that takes two inputs: i (an integer) and v (a real). `set` is used to modify the entry in the sequence at index i to have the value v.

- A method named `get(i)` that takes on input: i (an integer). This method returns the value of the sequence at index i.

- A method named `size()` that takes no arguments and returns the current size (length) of the sequence (an integer).

- A method named `indexInSeq(v)` that takes a real number v as input. For the sequence object `s`, `indexInSeq` returns the index i such that `s.get`(i) $\leq v \leq$ `s.get`(i+1).

As in Python, the code should use 0-based indexing. That is, the first entry in the sequence is at index 0. For a list of length n, the last entry is at index $n-1$.

Step 2

Write a second module that uses the first module to create another ADT. The second ADT represents a curve in the Cartesian x-y plane. The curve object has two state variables: a sequence of x values and a corresponding sequence of y values. For this assignment we are assuming that the values of x are increasing; that is, $x_i < x_{i+1}$ for all i values. Write your module in the Python file `CurveADT.py`. The module should define the class `CurveT`. The methods for `CurveT` are defined below. Where specifically indicated, please use numpy (`http://www.numpy.org/`) in your implementation.

- A constructor named `CurveT(s)` that takes one argument: s a string, where s represents the name for a text file that consists of two columns of data, with each data entry in a row separated by a comma and a space, and each row separated by a newline. The first column is the x values and the second is the corresponding y values. When this method is called the x and y sequence state variables for `CurveT` are set to the values of the first and second column, respectively, of the input file.

- A method named `linVal(x)` that takes one argument: x (a real) and returns one value y (a real). The value of y is found using linear interpolation between the data values on either side of x. If we assume the point on the curve to the left of x is (x_1, y_1) and the point to the right of x is (x_2, y_2), then the equation of y is:

$$y = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x - x_1) + y_1$$

- A method named `quadVal(x)` that behaves analogously to `linVal`, but with quadratic interpolation. We now have three points of interest, with corresponding subscripts of 0, 1 and 2. Assuming that the data is equally spaced, the equation for y is:

$$y = y_1 + \frac{y_2 - y_0}{x_2 - x_0}(x - x_1) + \frac{y_2 - 2y_1 + y_0}{2(x_2 - x_1)^2}(x - x_1)^2$$

- A method named `npolyVal(n, x)` that takes two inputs: an integer n and a real number x. This method returns an approximation of the value of y at x, like the previous two methods. However, in this case, regression (best fitting) is used instead of interpolation. In the case of interpolation, the interpolating polynomial (linear or quadratic in the above two cases) passes **exactly** through the data points. In the case of regression, the goal is instead to find a polynomial that **minimizes** the square of the error (difference) between the data and the fitted polynomial. The idea is to generalize the familiar notion of a best fit line, to be a best fit polynomial. The parameter n determines the degree of the polynomial. If n is 1, the best fit is for a line, if n is 2, the best fit is quadratic etc. You should use the `polyfit` function from `numpy` to find the best fit polynomial. You should then evaluate the resulting polynomial at x and return this value.

Step 3

Write a third module that tests the first and second modules together. It should be a Python file named `testSeqs.py`. Your initial git repo contains a `Makefile` (provided for you) with a rule `test` that runs your `testSeqs` source with the Python interpreter. Each procedure should have at least one test case. Please note for yourself the rationale for test case selection and the results of testing. You will need this information when writing your report in Step 7. The requirements for testing are deliberately vague; at this time, we are most interested in your ideas and intuition for how to build and execute your test suite.

Step 4

Test the supplied `Makefile` rule for `doc`. This rule should compile your documentation into an html and LaTeX version. Your documentation should be generated to the `A1` folder. Along with the supplied `Makefile`, a doxygen configuration file (`seqDoc`) is also given in your initial repo.

Step 5

Submit (add, commit and push) the files `SeqADT.py`, `CurveADT.py`, and `testSeqs.py` using git. Please **do not change** the names and locations for the files already given in your git project repo. You should also push any input data files you created for testing purposes. For Part 1, the only files that you should modify are the Python files and the only "new" files you should create are the input data files. Changing other files could result in a serious grading penalty, since the TAs might not be able to run your code and documentation generation. You should NOT submit your generated documentation (html and latex folders). In general, files that can be regenerated are not put under version control.

After the deadline for submitting your solution has passed, your partner files, `SeqADT.py` and `CurveADT.py`, will be pushed to your repo. **Including your name in your partner code files is optional.**

Part 2

Step 6

After you have received your partner's files, replace your corresponding files with your partner's. Do not initially make any modifications to any of the code. Run your test module and record the results. Your evaluation for this step does not depend on the quality of your partner's code, but only on your discussion of the testing results. If the tests fail, for the purposes of understanding what happened, you are allowed to modify your partner's code.

Step 7

Write a report using LaTeX (`report.tex`) following the template given in your repo. The elements that you need to fill in include the following:

1. Your name and macid.
2. Your `SeqADT.py`, `CurveADT.py` and `testSeqs.py` files.
3. Your partner's `SeqADT.py` and `CurveADT.py` files.
4. The results of testing your files (along with the rational for test case selection).
5. The results of testing your files combined with your partner's files.
6. A discussion of the test results and what you learned doing the exercise. List any problems you found with (a) your program, (b) your partner's module, and (c) the specification of the modules (as given in this assignment description).
7. Answers to the following questions
    - For each of the methods in each of the classes, please classify it as a constructor, accessor or mutator.
    - What are the advantages and disadvantages of using an external library like `numpy`
    - The `SeqT` class overlaps with the functionality provided by Python's in-built list type. What are the differences between `SeqT` and Python's list type? What benefits does Python's list type provide over the `SeqT` class?

- What complications would be added to your code if the assumption that $x_i < x_{i+1}$ no longer applied?
- Will `linVal(x)` equal `npolyVal(n, x)` for the same `x` value? Why or why not?

Commit and push `report.tex` and `report.pdf`. Although the pdf file is a generated file, for the purpose of helping the TAs, we'll make an exception to the general rule of avoiding version control for generated files. If you have made any changes to your Python files, you should also push those changes.

Including code in your report is made easier by the `listings` package: https://en.wikibooks.org/wiki/LaTeX/Source_Code_Listings.

Linking to the original code in the repo is also helpful via the hyperref package: https://www.sharelatex.com/learn/Hyperlinks.

Notes

1. Your git repo will be organizes with the following directories at the top level: `A1`, `A2`, `A3`, and `A4`. Inside the `A1` folder you will start with initial stubs of the files and folders that you need to use. Please do not change the names or locations of any of these files or folders.

2. Please use the following doxygen components at the start of all Python files `@file`, `@author`, `@brief` and `@date`.

3. Your program must work in the ITB labs on mills when compiled with its versions of Python (version 3), LaTeX, doxygen and make. Python is called via python3 on mills.

4. If completing the assignment requires making any assumptions, or adding exceptions, please document this. Exceptions are documented with `@throws`.

5. The specification is for the external interface of the objects. That is, the specification is how other programs would access the services of these modules, not how the modules will be implemented.

6. Any changes to the assignment specification will be announced in class. It is your responsibility to be aware of these changes.