# Assignment 2

COMP SCI 2ME3 and SFWR ENG 2AA4

February 7, 2018

## 1 Dates and Deadlines

**Assigned:** February 1, 2018

**Part 1:** February 12, 2018

**Receive Partner Files:** February 18, 2018

**Part 2:** March 2, 2018

**Last Revised:** February 7, 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 an ADT, an Abstract Object and libraries related to processing and plotting curves and sequences of curves. As for the previous assignment, you will use doxygen, make, LaTeX and Python (version 3). In addition, this assignment will use PyTest for unit testing. This assignment also takes advantage of functional programming in Python.

An example of a sequence of curves is given in Figure 1. This set of curves is used for predicting whether a given plate of glass will break under a specified blast load.

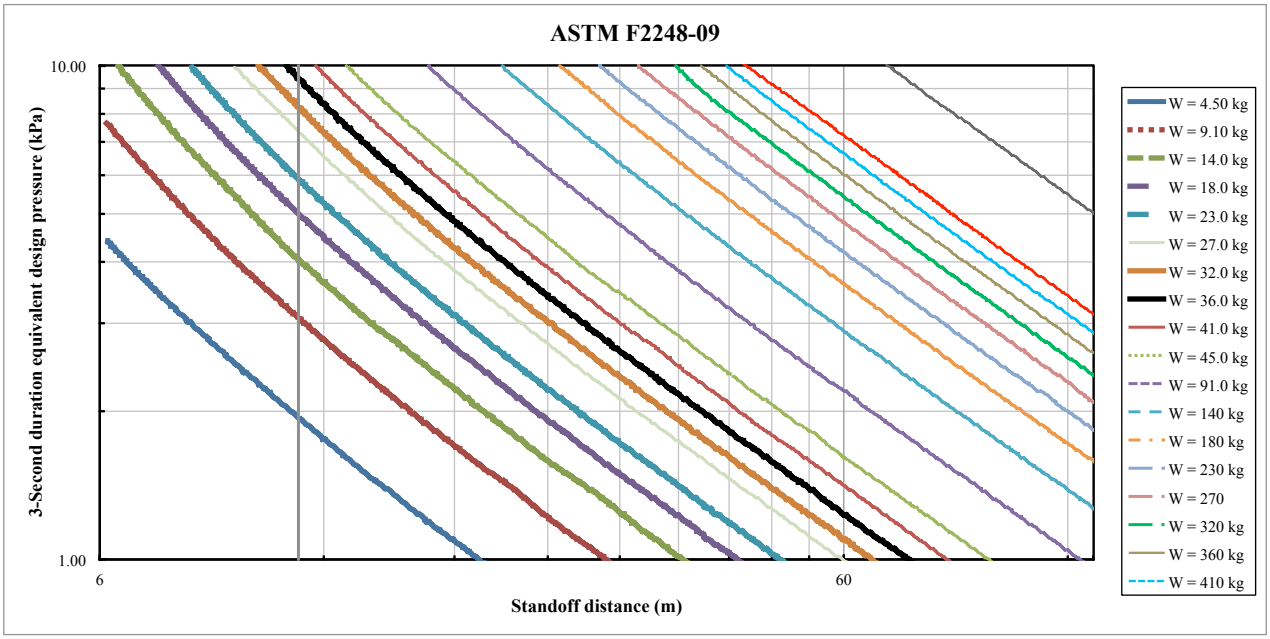


Figure 1: 3 second equivalent pressure ($q$) versus Stand off distance (SD) versus charge weight ($w$)

All of your code, except for the testing files, which are optional, should be documented using doxygen. Your report should be written using LaTeX. Your code should follow the given specification exactly. In particular, you should not add public methods or procedures that are not specified and you should not change the number or order of parameters for methods or procedures. If you need private methods or procedures, please use the Python convention of naming the files with the double underscore (`__methodName__`) (dunders). **Please follow specified naming EXACTLY. You do not want to lose marks for a simple mistake.**

For the purpose of understandability, the specification provided at the end of the assignment uses notation that is slightly different from the Hoffman and Strooper notation. Specifically the types for real and natural numbers are represented by $\mathbb{R}$ and $\mathbb{N}$, respectively. (In this specification, the natural numbers are assumed to include 0.) Also, subscripts are used for indexing a sequence. For instance, $x_i$ means the same thing as $x[i]$.

# Part 1

## Step 1

Write a module that creates a Curve ADT. It should consist of a Python code file named `CurveADT.py`. The specification for this module is given at the end of the assignment. The exceptions for `CurveADT.py` (and all other modules) should be in the file `Exceptions.py`. Hint: The implementation will be easier if you use the scipy function `interp1d`.

## Step 2

Write a module that implements an abstract object called Data. It should consist of a Python file named `Data.py`. The new module should follow the specification given at the end of the assignment. Although efficient use of computing resources is always a good goal, your implementation will be judged on correctness and not on performance. Remember, the exceptions should be defined in the file `Exceptions.py`.

## Step 3

Write a library module that provides services for processing sequences. It should consist of a Python code file named `SeqServices.py`. The specification for this module is given at the end of the assignment.

## Step 4

Write a library module for plotting curves. It should consist of a Python code file named `Plot.py`. The specification for this module is given at the end of the assignment. You should use matplotlib for your implementation. Remember, the exceptions should be defined in the file `Exceptions.py`.

## Step 5

Write a library module for loading curves into the abstract object Data. It should consist of a Python code file named `Load.py`. The specification for this module is given at the end of the assignment.

## Step 6

Write a module (named `test_All.py`), using PyUnit, that tests the following modules: `CurveADT.py`, `Data.py` and `SeqServices.py`. The given makefile `Makefile` will have the rule `test` for running your tests. Each procedure should have at least one test case. Record your rationale for test case selection and the results of using this module to test the procedures in your modules. (You will submit your rationale with your report in Step 10.) Please make an effort to test normal cases, boundary cases, and exception cases. Your test program should compare the calculated output to the expected output and provide a summary of the number of test case that have passed or failed.

For testing `CurveADT.py`, do not worry about test cases close to edges of the domain. The formulas for interpolation and differentiation will break down if they are looking for data outside of the bounds. We don't need to worry about this complication for this assignment.

## Step 7

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

## Step 8

Submit (add, commit and push) the files `CurveADT.py`, `Data.py`, `SeqServices.py`, `Plot.py`, `Load.py`, `Exceptions.py` and `test_All.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.

You should tag your final submission of Part 1 of the assignment with the label `A2Part1`.

# Part 2

Your `CurveADT.py`, `Data.py`, `SeqServices.py` files will automatically be pushed to your partner's repo and vice versa. **Including your name in your partner code files is optional.**

# Step 9

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 10

Write a report using LaTeX (`report.tex`) following the template given in your repo. The final submission should have the tag `A2Part2`. The report should include the following:

1. Your name and macid.
2. Your `CurveADT.py`, `Data.py`, `SeqServices.py`, `Plot.py`, `Load.py` and `test_All.py` files.
3. Your partner's `CurveADT.py`, `Data.py`, `SeqServices.py` files.
4. The results of testing your files (along with the rational for test case selection). The summary of the results should consist of the following: the number of passed and failed test cases, and brief details on any failed test cases.
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.
7. Answers to the following questions
    - What is the mathematical specification of the `SeqServices` access program isInBounds(X, x) if the assumption that X is ascending is removed?

- How would you modify `CurveADT.py` to support cubic interpolation?
- What is your critique of the CurveADT module's interface. In particular, comment on whether the exported access programs provide an interface that is consistent, essential, general, minimal and opaque.
- What is your critique of the Data abstract object's interface. In particular, comment on whether the exported access programs provide an interface that is consistent, essential, general, minimal and opaque.

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.

**Notes**

1. Your git repo will be organizes with the following directories at the top level: `A1`, `A2`, `A3`, and `A4`.

2. Inside the `A2` 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.

3. Please put your name and macid at the top of each of your source files, except for those that you share with a partner. Including your name and macid is optional for those files.

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

5. Python specifics:

   - The exceptions in the specification should be implemented via Python exceptions. Your exceptions should have exactly the same name, including case, as given in the specification. Your exceptions should inherit from the Exception class and they should only be used with one argument, a string explaining what problem has occurred.
   - For the Python implementation of the abstract module, your access programs should be called via, `Data.accessProg`, not `Data_accessProg`, as shown in the specification. The call `Data.Data_accessProg` is also incorrect. Some sample calls include the following: `Data.init()`, `Data.add(s, z)`, etc.

- Since the specification is silent on this point, for methods that return an object, or use objects in their state, you can decide to either use references or construct new objects. The implementation will be easier if you just work in terms of references to objects.

6. **Your grade will be based to a significant extent on the ability of your code to compile and its correctness. If your code does not compile, then your grade will be significantly reduced.**

7. **Any changes to the assignment specification will be announced in class. It is your responsibility to be aware of these changes. Please monitor all pushes to the course git repo.**

# Curve ADT Module

## Template Module

CurveADT

## Uses

SeqServices

## Syntax

### Exported Constants

MAX_ORDER = 2
DX = $1 \times 10^{-3}$

### Exported Types

CurveT = ?

### Exported Access Programs

| Routine name | In | Out | Exceptions |
|---|---|---|---|
| new CurveT | $X : \mathbb{R}^n$, $Y : \mathbb{R}^n$, $i : \mathbb{N}$ | CurveT | IndepVarNotAscending, SeqSizeMismatch, InvalidInterpOrder |
| minD | | $\mathbb{R}$ | |
| maxD | | $\mathbb{R}$ | |
| order | | $\mathbb{N}$ | |
| eval | $x : \mathbb{R}$ | $\mathbb{R}$ | OutOfDomain |
| dfdx | $x : \mathbb{R}$ | $\mathbb{R}$ | OutOfDomain |
| d2fdx2 | $x : \mathbb{R}$ | $\mathbb{R}$ | OutOfDomain |

## Semantics

### State Variables

minx: $\mathbb{R}$
maxx: $\mathbb{R}$

o: $\mathbb{N}$
f: $\mathbb{R} \to \mathbb{R}$

**State Invariant**

None

**Assumptions**

The user will not request function evaluations that would cause the interpolation to select index values outside of where the function is defined.

**Access Routine Semantics**

new CurveT($X, Y, i$):

- transition: minx, maxx, $o, f := X_0, X_{|X|-1}, i, (\lambda v : \text{interp}(X, Y, o, v))$
- output: *out* := self
- exception: ($\neg$isAscending($X$) $\Rightarrow$ IndepVarNotAscending$||X| \neq |Y| \Rightarrow$ SeqSizeMismatch$|i \notin$ [1..MAX_ORDER] $\Rightarrow$ InvalidInterpOrder)

minD():

- output: *out* := minx
- exception: None

maxD():

- output: *out* := maxx
- exception: None

order():

- output: *out* := o
- exception: None

eval($x$):

- output: *out* := $f(x)$

- exception: $(\neg(\text{minx} \leq x \leq \text{maxx}) \Rightarrow \text{OutOfDomain}))$

dfdx($x$): *# approximate first derivative using forward divided difference*

- output: $out := \frac{f(x+DX)-f(x)}{DX}$
- exception: $(\neg(\text{minx} \leq x \leq \text{maxx}) \Rightarrow \text{OutOfDomain}))$

d2fdx2($x$): *# approximate second derivative using forward divided difference*

- output: $out := \frac{f(x+2DX)-2f(x+DX)+f(x)}{DX^2}$
- exception: $(\neg(\text{minx} \leq x \leq \text{maxx}) \Rightarrow \text{OutOfDomain}))$

## Local Functions

interp: $\mathbb{R}^n \times \mathbb{R}^n \times \mathbb{N} \times \mathbb{R} \to \mathbb{R}$

interp($X$, $Y$, $o$, $v$)

$\equiv (o = 1 \Rightarrow \text{interpLin}(X_i, Y_i, X_{i+1}, Y_{i+1}, v) | o = 2 \Rightarrow \text{interpQuad}(X_{i-1}, Y_{i-1}, X_i, Y_i, X_{i+1}, Y_{i+1}, v))$

where $i = \text{index}(X, v)$

# Data Module

## Module

Data

## Uses

CurveADT for CurveT, SeqServices for interpLin and index

## Syntax

### Exported Constants

MAX_SIZE = 10

### Exported Access Programs

| Routine name | In | Out | Exceptions |
|---|---|---|---|
| Data_init | | | |
| Data_add | s: CurveT, $z : \mathbb{R}$ | | Full, IndepVarNotAscending |
| Data_getC | $i : \mathbb{N}$ | | InvalidIndex |
| Data_eval | $x : \mathbb{R}, z : \mathbb{R}$ | | OutOfDomain |
| Data_slice | $x : \mathbb{R}, i : \mathbb{N}$ | CurveT | |

## Semantics

### State Variables

$S$: sequence of CurveT
$Z$: sequence of $\mathbb{R}$

### State Invariant

$|S| \leq$ MAX_SIZE
$|Z| \leq$ MAX_SIZE

### Assumptions

Data_init() is called before any other access program.

**Access Routine Semantics**

Data_init():

- transition: $S, Z := <>, <>$
- exception: none

Data_add($s, z$):

- transition: $S, Z := S||< s >, Z||< z >$
- exception: $exc := (|S| = \text{MAX\_SIZE} \Rightarrow \text{Full}|z \leq Z_{|Z|-1} \Rightarrow \text{IndepVarNotAscending})$

Data_getC($i$):

- output: $out := S[i]$
- exception: $exc := (\neg(0 \leq i < |S|) \Rightarrow \text{InvalidIndex})$

Data_eval($x, z$):

- output: $out := \text{interpLin}(Z_j, S_j.\text{eval}(x), Z_{j+1}, S_{j+1}.\text{eval}(x), z)$, where $j = \text{index}(Z, z)$
- exception: $exc := (\neg\text{isInBounds}(Z, z) \Rightarrow \text{OutOfDomain})$

Data_slice($x, i$):

- output: $out := \text{CurveT}(Z, Y, i)$, where $Y = ||(i : \mathbb{N}|i \in [0..|Z| - 1] : \langle S_i.\text{eval}(x)\rangle)$ or $Y =$ map eval($x$) $S$ # *in both cases there is an abuse of notation. The idea is simply to convey that the independent variable is the sequence $Z$ and the dependent variable is the sequence where the $i$th entry is found by evaluating the curve corresponding to $Z_i$ at $x$.*
- exception: None

# Sequence Services Module

## Module

SeqServices

## Uses

None

## Syntax

### Exported Constants

None

### Exported Access Programs

| Routine name | In | Out | Exceptions |
|---|---|---|---|
| isAscending | $X : \mathbb{R}^n$ | $\mathbb{B}$ | |
| isInBounds | $X : \mathbb{R}^n, x : \mathbb{R}$ | $\mathbb{B}$ | |
| interpLin | $x_1 : \mathbb{R}, y_1 : \mathbb{R}, x_2 : \mathbb{R}, y_2 : \mathbb{R}, x : \mathbb{R}$ | $\mathbb{R}$ | |
| interpQuad | $x_0 : \mathbb{R}, y_0 : \mathbb{R}, x_1 : \mathbb{R}, y_1 : \mathbb{R}, x_2 : \mathbb{R}, y_2 : \mathbb{R}, x : \mathbb{R}$ | $\mathbb{R}$ | |
| index | $X : \mathbb{R}^n, x : \mathbb{R}$ | $\mathbb{N}$ | |

## Semantics

### State Variables

None

### State Invariant

None

### Assumptions

None, unless noted with a particular access program

**Access Routine Semantics**

isAscending($X$)

- output: $out := \neg\exists(i|i \in [0..|X|-2] : X_{i+1} \leq X_i)$
- exception: none

isInBounds($X, x$) # *assuming isAscending is True*

- output: $out := X_0 \leq x \leq X_{|X|-1}$
- exception: none

interpLin($x_1, y_1, x_2, y_2, x$) # *assuming isAscending is True*

- output: $out := \frac{(y_2-y_1)}{(x_2-x_1)}(x - x_1) + y_1$
- exception: none

interpQuad($x_0, y_0, x_1, y_1, x_2, y_2, x$) # *assuming isAscending is True*

- output: $out := y_1 + \frac{y_2-y_0}{x_2-x_0}(x - x_1) + \frac{y_2-2y_1+y_0}{2(x_2-x_1)^2}(x - x_1)^2$
- exception: none

index($X, x$) # *assuming isAscending is True and isInBounds is True*

- output: $out := i$ such that $X_i \leq x < X_{i+1}$
- exception: none

# Plot Module

## Module

Plot

## Uses

CurveT

## Syntax

### Exported Constants

None

### Exported Access Programs

| Routine name | In | Out | Exceptions |
|---|---|---|---|
| PlotSeq | $X : \mathbb{R}^n, Y : \mathbb{R}^n$ | | SeqSizeMismatch |
| PlotCurve | $c$ : CurveT, $n : \mathbb{N}$ | | |

## Semantics

### Environment Variables

win: two dimensional sequence of coloured pixels

### State Variables

None

### State Invariant

None

### Assumptions

For plotting the user will select the number of subdivisions to be small enough that there will not be an interpolation problem with the end points.

**Access Routine Semantics**

PlotSeq($X, Y$)

- transition: modify win so that it displays an x-y graph of the data points showing X and the corresponding Y values. X is the independent variable and Y as the dependent variable.
- exception: ($|X| \neq |Y| \Rightarrow$ SeqSizeMismatch)

PlotCurve($c, n$)

- transition: modify win so that it displays an x-y graph of the curve c between c.minD to c.maxD for evaluating $c$ at $n$ equally spaced points in between. For quadratic interpolation the points c.minD and c.maxD do not have to be plotted, since c.eval() is problematic for these points. For linear interpolation c.maxD does not have to be plotted.
- exception: none

# Load Module

## Module

Load

## Uses

CurveADT, Data

## Syntax

### Exported Constants

None

### Exported Access Programs

| Routine name | In | Out | Exceptions |
|---|---|---|---|
| Load | $s$ : string | | |

## Semantics

### Environment Variables

infile: two dimensional sequence of text characters

### State Variables

None

### State Invariant

None

### Assumptions

The input file will match the given specification.

**Access Routine Semantics**

Load($s$)

- transition: read data from the file infile associated with the string s. Use this data to update the state of the Data module. The text file consists of a text file with the following format, where $z_i$ is the value for the $i$th curve, $o_i$ is the order of the $i$th curve, $x_j^i$ is the $x$ value for the $j$th point of the $i$th curve, $y_j^i$ is the $y$ value for the $j$th point of the $i$th curve, $m$ is the number of curves and $n_i$ is the number of data points in the $i$th curve. All data values in a row are separated by commas. Rows are separated by a new line. Please note, there is no requirement that there is an equal number of data points for each of the curves. If there is no data past a certain row, the data entry should be empty, but there will still need to be a comma to mark the empty spot.

$$\begin{array}{ccccc} z_1, & z_2, & z_3, & ..., & z_m \\ o_1, & o_2, & o_3, & ..., & o_m \\ x_1^1, & y_1^1, & ..., & x_1^m, & y_1^m \\ x_2^1, & y_2^1, & ..., & x_2^m, & y_2^m \\ x_3^1, & y_3^1, & ..., & x_3^m, & y_3^m \\ ..., & ..., & ..., & ..., & ... \\ x_{n_1-3}^1, & y_{n_1-3}^1, & ..., & x_{n_m-1}^m, & y_{n_m-1}^m \\ x_{n_1-2}^1, & y_{n_1-2}^1, & ..., & x_{n_m}^m, & y_{n_m}^m \\ x_{n_1-1}^1, & y_{n_1-1}^1, & ..., & , & \\ x_{n_1}^1, & y_{n_1}^1, & ..., & , & \end{array} \tag{1}$$

- exception: none