Assignment 2 Report

Shengchen Zhou–zhous20

March 3, 2018

This report contains following contents:
1. Description of Test Cases
2. Testing results of my code and partner's code.
3. A further discussion on my code and partner's code.
4. Problems about The Specification.
5. Specification Comparison against A1.
6. Answers to Questions

1 Testing of the Original Program

Brief discription of test cases:

SeqSerivces.py:

1. isAscending(X)
   Corner case: test scenario of $X_{(i+1)} = X_i$
   Exception: n/a

2.isInBounds(X, x)
   Corner case: test scenario of $X_{(i+1)} = X_i$
   Exception: n/a

3. interpLin(x1, y1, x2, y2, x)
   Corner case: test scenario of $x_1 = x_2$, which will give divide-by-zero error (ZeroDivisionError to be specific)
   Exception: n/a

4. interpQuad(x1, y1, x2, y2, x)
   Corner case: test scenario of $x_0 = x_2$ or $x_1 = x_2$, which will give divide-by-zero error (ZeroDivisionError to be specific)

Exception: n/a

5.index(X, x)
Corner case: test scenario of $X_{(i+1)} = X_i$, which is ignored as it will possibly lead to no index returned and there are no relevant assumptions specified. (ignored means the test function will bypass the testing.)
Exception: n/a

CurveADT.py:

1. new CurveT(X, Y, i)
   Corner case: n/a
   Exception: IndepVarNotAscending / SeqSizeMismatch / InvalidInterpOrder

2. minD()
   Corner case: n/a
   Exception: n/a

3. maxD()
   Corner case: n/a
   Exception: n/a

4. order()
   Corner case: n/a
   Exception: n/a

5. eval(x)
   Corner case: as for the wrapped local function interp(X, Y, o, v), if i = 0, then $X_{(i-1)}$ will not exist, which implies it should be automatically incremented by one in this case, which is i = 1.
   Exception: OutOfDomain

6. dfdx(x)
   Corner case: n/a
   Exception: OutOfDomain

7. d2fdx2(x)
   Corner case: n/a
   Exception: OutOfDomain

Data.py:

1. Data_init()
   Corner case: n/a
   Exception: n/a

2. Data_add()
   Corner case: n/a
   Exception: IndepVarNotAscending

3. Data_getC()
   Corner case: n/a
   Exception: InvalidIndex

4. Data_eval()
   Corner case: n/a
   Exception: OutOfDomain

5. Data_slice()
   Corner case: n/a
   Exception: n/a

Output in ubuntu terminal:

```
src/test_All.py .............................................. [100%]
——— coverage: platform linux, python 3.5.2-final-0 ———
Name                      Stmts    Miss    Cover
------------------------------------------------

src/A2Examples.py            22      22       0%
src/CurveADT.py              71       0     100%
src/Data.py                  41       0     100%
src/Exceptions.py            12       0     100%
src/Load.py                  40      40       0%
src/Plot.py                  23      23       0%
src/SeqServices.py           38       1      97%
src/test_All.py             217       4      98%
------------------------------------------------
TOTAL                       464      90      81%
========== 50 passed in 1.69 seconds ==========
```

Summary of results:

The results look good. (further discussion see below)

2 Results of Testing Partner's Code

```
partner/test_All.py ................................................ [100%]
——— coverage: platform linux, python 3.5.2-final-0 ———
Name                          Stmts    Miss    Cover
----------------------------------------------------
src/A2Examples.py               22      22      0%
src/CurveADT.py                 40       2     95%
src/Data.py                     30       1     97%
src/Exceptions.py               12       0    100%
src/Load.py                     40      40      0%
src/Plot.py                     23      23      0%
src/SeqServices.py              15       0    100%
src/test_All.py                217      17     92%
-----------------------------------------------------
TOTAL                          399     105     74%
================= FAILURES ==============
(detailed outputs omitted here)
=========12 failed, 38 passed in 2.11 seconds=========
```

3 Discussion of Test Results

3.1 Problems with Original Code

The results look good, dont see a single failed case, and the uncovered lines (3% for SeqServices.py) are not relevant for the code's functionality.
All corner cases and exceptions have been thoroughly tested according to the content of last section: Brief description of test cases, please check it for details.

3.2 Problems with Partner's Code

The results look problematic, there are a few failed cases, not to mention the code coverage. The spotted problems are listed as follows:

SeqSerivces.py

for index(X, x)
The formula: $X_i \leq x < X_(i + 1)$ has been incorrectly implemented as $X_i \leq x \leq X_{(i+1)}$, in this case, index can be return only when there the condition:
$X_i = x = X_{(i+1)}$ get satisfied.
Because as a basic helper function, the index(X, x) is wrongly implemented, so unfortunately most of the functions which are dependent on it will not be able to be tested until it get fixed.

CurveADT.py

for new CurveT(X, Y, i) / dfdx(x) / d2fdx2(x)
Incorrect way to implement the Exported Constants.
When define a module level constant, there is no need to declare it as global.
Unfortunately, same as above, the constant is such a fundamental thing, any methods which are using them will be able to be tested.
for eval(x)
Just as mentioned in previous section: Brief description of test cases, when i = 0, the local function: interp(X, Y, o, v) should increment it to 1 in order to run quadratic interpolation algorithm.
The partners function failed to do so.
Even though this is not specified in instruction mandatorily, but I think its very obvious that it should be taken care of.

Data.py

for Data_add(s, z)
The exception FULL is not found in the Data module, which thus lead to a NameError. Which is proved to be a typo, this exception should be called Full instead of its original all uppercase form.
In order to inspect further about partners code, apparently the wrongly defined SeqSerivces.index(X, x), the inappropriately defined constants, and the miscalled exception type prevent Pytest checking other parts, I tried to fix these three exposed issues for partner and then re-test it.

Here is the testing result after a simple fix:

```
partner/test_All.py ............................................... [100%]
——— coverage: platform linux, python 3.5.2-final-0 ———
Name                              Stmts     Miss    Cover
------------------------------------------------------------
src/A2Examples.py                    22       22       0%
src/CurveADT.py                      42        0     100%
src/Data.py                          30        0     100%
src/Exceptions.py                    12        0     100%
src/Load.py                          40       40       0%
src/Plot.py                          23       23       0%
src/SeqServices.py                   15        0     100%
src/test_All.py                     217        5      98%
-------------------------------------------------------------
TOTAL                               401       90      78%
================= FAILURES ==============
(detailed outputs omitted here)
=========1 failed, 49 passed in 2.11 seconds=========
```

3.3 Problems with Assignment Specification

SeqSerivces.py:

index(X, x)
The assumptions of isAscending(X) is True as well as isInBounds(X, x) is True cannot avoid the situation of no index get returned when there exists at least one pair of two adjacent value: $X_{(i+1)} = X_i$, as there could be no x to satisfy $X_(i + 1) \leq x < X_i$.

interpLin(x1, y1, x2, y2, x)
There is one missed assumption: $x_1 x_2$, given that two identical x values would lead to a divide-by-zero error.

interpQuad(x1, y1, x2, y2, x)
Same as interpLin(x1, y1, x2, y2, x), there are two missed assumptions: $x_0 x_2$ and $x_1 x_2$, the divide-by-zero error would be raised.

CurveADT.py:
interp(X, Y, o, v) (used in eval(x))
When i = 0, there is no $X_{(i-1)}$, hence a supplement logic should be implemented to let i be automatically incremented by one in this case, so there will be sufficient three points for a quadratic interpolation.
Furthermore, in interp(X, Y, o, v) or eval(), there is a need to check the number of points in sequence, for o = 1, at least two points are required; for o = 2, at least three points are required. This requirements could become assumptions or we could prepare the function to raise a relevant exception.

3.4 Specification Comparison against A1

In Assignment 1, the concept of Sequence is defined as a standalone ADT class, which has its own add, rm, set, get, size and indexInSeq methods. However, in Assignment 2, we have to make use of the raw List type in python, and a Sequence Services module to provide utility functions, i.e. isAscending, isInBounds, interpLin, interQuad, and index to operate on it. In Assignment 1 we may need to implement our own way to do isAscending and isInBounds. In addition, the interpLin and interQuad works well when it comes to implementing the CurveT.
In Assignment 1, the CurveT instance is initialized by a datafile, and only interpolation/evaluation methods are provided. In contrast, the Assignment 2 strips off the data reading functionality, the CurveT instance now can be explicitly initialized by two sequences and an order instead.
The CurveT class in Assignment 1 doesnt include a curves order, and it doesnt give any methods to access the domain information of the independent variable. But in Assignment 2, we have all of it.

4 Answers

1. What is the mathematical specification of the `SeqServices` access program isInBounds(X, x) if the assumption that X is ascending is removed?

   If the assumption of X is ascending is removed, then we cannot ensure that the first value of X sequence is the smallest one, and the last value of X sequence is the greatest one, thus, we will have difficulty checking if the given x is in the bounds. The only way to solve this is iterating all values to find the min and the max ones to clarify the bounds.

2. How would you modify `CurveADT.py` to support cubic interpolation?

   Since I use scipy.interpolate.interp1d() to help define the interpLin(x1, y1, x2, y2, x) and the interpQuad(x0, y0, x1, y1, x2, y2, x) functions in SeqServices, I could easily adopt the same approach to implement a third interpCubic(x0, y0, x1, y1, x2, y2, x3, y3, x) one. Apparently I need to provide three points instead of three or two, to let it performs the cubic interpolation. The core line will be: interpolate.interp1d(X, Y, kind='cubic'). After finishing implementing this basic function, I could use it in CurveTs local function interp(X, Y, o, v), just make sure that it's called only when the given order is 3 and the input sequence also has enough points. Finally I simply wrap the interp(X, Y, o, v) in CurveT.eval(x), the only additional thing to do there is to check exceptions, such as OutOfDomain (existed) and NotEnoughPoints (possible new exception for having no enough points).

3. 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.

   From the minimal aspect, we might separate the derivative calculation from the encapsulation, i.e. dfdx(x) and d2fdx2(x). The DX constant doesn't need to be included in the class and a delta x value for computing derivatives is not a mandatory part of a curve.
   From other perspectives, it looks good.

4. 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.

   Its easy to get the domain of x from a CurveT interface, but we cannot tell the domain of Z as it is a private field of Data. Therefore it might be necessary to provide two accessors, e.g. Data_minZ() and Data_maxZ(), so that we could be able to tell if a given z is invalid before calling Data_add(s, z) and Data_eval(x, z).
   Everything else seems appropriate.

E Code for CurveADT.py

```
## @file CurveADT.py
#  @author Shengchen Zhou
#  @brief The curve ADT module
#  @date 15 Feb 2018


from Exceptions import IndepVarNotAscending, SeqSizeMismatch, InvalidInterpOrder, OutOfDomain
from SeqServices import isAscending, isInBounds, interpLin, interpQuad, index
from copy import copy


## @brief CurveT ADT class
#  @details It's assumed that the user will not request function
#   evaluations that would cause the interpolation to select index
#   values outside of where the function is defined
class CurveT:
    MAX_ORDER = 2  # the maximum order constant
    DX = 1e-3  # the delta x constant

    ## @brief Constructor of the CurveT type
    #  @details Construct a CurveT instance
    #  @param X The input x sequence
    #  @param Y The input y sequence
    #  @param i The input order
    #  @exception IndepVarNotAscending if X is not in ascending order
    #  @exception SeqSizeMismatch if X and Y don't have the same size
    #  @exception InvalidInterpOrder if i is an invalid interpolation order
    def __init__(self, X, Y, i):
        if isAscending(X) is False:
            raise IndepVarNotAscending()
        elif len(X) != len(Y):
            raise SeqSizeMismatch()
        elif i <= 0:
            raise InvalidInterpOrder()
        elif i > CurveT.MAX_ORDER:
            raise InvalidInterpOrder()

        self.minx = X[0]
        self.maxx = X[-1]
        self.o = i

        X = copy(X)
        Y = copy(Y)

        def f(x):
            y = interp(X, Y, i, x)
            return y
        self.f = f

    ## @brief Obtain the smallest x
    #  @return The smallest x
    def minD(self):
        return self.minx

    ## @brief Obtain the greatest x
    #  @return The greatest x
    def maxD(self):
        return self.maxx

    ## @brief Obtain the order
    #  @return The order
    def order(self):
        return self.o

    ## @brief Calculate the y at the input x
    #  @param x The input x
    #  @exception OutOfDomain if x is not in the domain of X
    #  @return The y
    def eval(self, x):
        if x < self.minx:
            raise OutOfDomain()
        if x > self.maxx:
            raise OutOfDomain()
        y = self.f(x)
        return y
    ## @brief Calculate the first derivative at the input x
    #  @param x The input x
    #  @exception OutOfDomain if x is not in the domain of X
    #  @return The 1st derivative at the input x
    def dfdx(self, x):
        if x < self.minx:
            raise OutOfDomain()
        if x > self.maxx:
            raise OutOfDomain()
        numerator = self.f(x + CurveT.DX) - self.f(x)
        denominator = CurveT.DX
        rc = numerator / denominator
        return rc

    ## @brief Calculate the second derivative at the input x
    #  @param x The input x
    #  @exception OutOfDomain if x is not in the domain of X
    #  @return The 2nd derivative at the input x
    def d2fdx2(self, x):
        if x < self.minx:
            raise OutOfDomain()
        if x > self.maxx:
            raise OutOfDomain()
        numerator = (self.f(x + CurveT.DX * 2) - self.f(x + CurveT.DX) +
                     self.f(x) - self.f(x + CurveT.DX))
        denominator = CurveT.DX ** 2
        rc = numerator / denominator
        return rc


## @brief Conduct a linear interpolation
#  @param X The input x sequence
#  @param Y The input y sequence
#  @param o The input order
#  @param v The input v
#  @return The y as the interpolated result with regards to the input v
def interp(X, Y, o, v):
    i = index(X, v)

    xi = X[i]
    yi = Y[i]
    xi1 = X[i + 1]
    yi1 = Y[i + 1]

    if o == 1:
        y = interpLin(xi, yi, xi1, yi1, v)
    elif i > 0:
        xi_1 = X[i - 1]
        yi_1 = Y[i - 1]
        y = interpQuad(xi_1, yi_1, xi, yi, xi1, yi1, v)
    else:
        xi2 = X[i + 2]
        yi2 = Y[i + 2]
        y = interpQuad(xi, yi, xi1, yi1, xi2, yi2, v)

    return y
```

F Code for Data.py

```
## @file Data.py
#  @author Shengchen Zhou
#  @brief The data module
#  @date 15 Feb 2018


from Exceptions import Full, IndepVarNotAscending, InvalidIndex, OutOfDomain
from SeqServices import isInBounds, interpLin, index
from CurveADT import CurveT


## @brief Data class
#  @details It's assumed that Data.init() is called before any other access program
class Data:
    MAX_SIZE = 10  # the maximum size constant

    ## @brief Initializer of the Data type
    @staticmethod
    def init():
        Data.S = []
        Data.Z = []

    ## @brief Append a curve and a scalar value
    #  @param s The input curve
    #  @param z The input value
    #  @exception Full if the Data reaches its capacity
    #  @exception IndepVarNotAscending if Data.Z cannot be in ascending order
    #   after appended with the input z
    @staticmethod
    def add(s, z):
        if len(Data.S) == Data.MAX_SIZE:
            raise Full()
        if len(Data.Z) > 0:
            if Data.Z[-1] >= z:
                raise IndepVarNotAscending()
        Data.S.append(s)
        Data.Z.append(z)

    ## @brief Pick a curve
    #  @param i The index
    #  @exception InvalidIndex if i isn't a valid index of Data.S
    #  @return The Curve
    @staticmethod
    def getC(i):
        if i < 0:
            raise InvalidIndex()
        if i >= len(Data.S):
            raise InvalidIndex()
        s = Data.S[i]
        return s

    ## @brief Calculate the y at the input x and input z
    #  @param x The input x
    #  @param z The input z
    #  @exception OutOfDomain if input z is not in the domain of Data.Z
    #  @return The y
    @staticmethod
    def eval(x, z):
        if isInBounds(Data.Z, z) is False:
            raise OutOfDomain()
        j = index(Data.Z, z)
        xj = Data.Z[j]
        yj = Data.S[j].eval(x)
        xj1 = Data.Z[j + 1]
        yj1 = Data.S[j + 1].eval(x)
        y = interpLin(xj, yj, xj1, yj1, z)
        return y

    ## @brief Slice data
    #  @param x The input x
    #  @param i The order
    #  @return The curve
    @staticmethod
    def slice(x, i):
        Y = []
        for j in range(len(Data.S)):
```

```
        s = Data.S[j]
        y = s.eval(x)
        Y.append(y)
s = CurveT(Data.Z, Y, i)
return s
```

G Code for SeqServices.py

```
## @file SeqServices.py
#  @author Shengchen Zhou
#  @brief The sequence services module
#  @date 15 Feb 2018


from scipy import interpolate


## @brief Verify whether or not a input sequence is in ascending order
#  @param X The input sequence
#  @return True if the input sequence is in ascending order;
#    False if it isn't
def isAscending(X):
    rc = True  # the return code, defaults to True
    for i in range(len(X) - 1):
        xi, xi1 = X[i: i + 2]
        if xi1 < xi:  # non-ascending pair found
            rc = False
    return rc


## @brief Verify whether or not a input x is within the bounds of a input sequence
#  @details The input sequence is assumed to be in ascending order
#  @param X The input sequence
#  @param x The input x
#  @return True if the input x is within the bounds of the input sequence;
#    False if it isn't
def isInBounds(X, x):
    rc = True
    if x < X[0]:
        rc = False
    if x > X[-1]:
        rc = False
    return rc


## @brief Conduct a linear interpolation
#  @details Conduct a linear interpolation using two points.
#    The input two points are assumed to form a sequence
#    which should be in ascending order
#  @param x1 The x component of the input point #1
#  @param y1 The y component of the input point #1
#  @param x2 The x component of the input point #2
#  @param y2 The y component of the input point #2
#  @param x The x component of the to-be-determined point
#  @return The y component as the interpolated result with regards to the input x component
def interpLin(x1, y1, x2, y2, x):
    X = [x1, x2]
    Y = [y1, y2]
    f = interpolate.interp1d(X, Y, kind='linear')
    y = f(x)
    return y


## @brief Conduct a quadratic interpolation
#  @details Conduct a quadratic interpolation using three points.
#    The input three points are assumed to form a sequence
#    which should be in ascending order
#  @param x0 The x component of the input point #0
#  @param y0 The y component of the input point #0
#  @param x1 The x component of the input point #1
#  @param y1 The y component of the input point #1
#  @param x2 The x component of the input point #2
#  @param y2 The y component of the input point #2
#  @param x The x component of the to-be-determined point
#  @return The y component as the interpolated result with regards to the input x component
def interpQuad(x0, y0, x1, y1, x2, y2, x):
    X = [x0, x1, x2]
    Y = [y0, y1, y2]
    f = interpolate.interp1d(X, Y, kind='quadratic')
    y = f(x)
    return y


## @brief Get the index of a input x in a input sequence
```

```
#  @details An estimation approach is used if the input x is
#   not exactly one of the element inside the input sequence.
#   The input sequence is assumed to be in ascending order, and
#   the input x is assumed to be in bounds of the input sequence
#  @param X The input sequence
#  @param x The input x
#  @return The index of the input x
def index(X, x):
    rc = None
    for i in range(len(X) - 1):
        xi, xi1 = X[i: i + 2]
        if xi > x:
            continue
        if x >= xi1:
            continue
        rc = i
        break
    return rc
```

H Code for Plot.py

```
## @file Load.py
#  @author Shengchen Zhou
#  @brief The plot module
#  @details For plotting the user select the numbers of
#   subdivisions to be small enough that there will no be
#   an interpolation problem with then end points
#  @date 15 Feb 2018


from Exceptions import SeqSizeMismatch
import matplotlib.pyplot as plt
import numpy as np


## @brief Plot sequence
#  @param X The input sequence
#  @param Y The input sequence
def PlotSeq(X, Y):
    if len(X) != len(Y):
        raise SeqSizeMismatch()
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(X, Y, linestyle='None', marker='o')
    plt.show()


## @brief Plot curve
#  @param c The input curve
#  @param n The input points' amount
def PlotCurve(c, n):
    X = list(np.linspace(c.minD(), c.maxD(), n))
    del X[-1]
    if c.order() == 2:
        del X[0]
    Y = []
    for i in range(len(X)):
        y = c.eval(X[i])
        Y.append(y)
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(X, Y, linestyle='solid', marker='None')
    plt.show()
```

I Code for Load.py

```
## @file Load.py
#  @author Shengchen Zhou
#  @brief The load module
#  @date 15 Feb 2018


from CurveADT import CurveT
from Data import Data


## @brief Load data
#  @details It's assumed that the input file will match the given specification
#  @param s The input file's filename
def Load(s):
    Data.init()
    with open(s, 'r') as infile:
        i = 0
        for line in infile:
            if i == 0:
                Data_Z = line.split(',')
                __toFloat__(Data_Z)
                Xs = []
                Ys = []
                for j in range(len(Data_Z)):
                    Xs.append([])
                    Ys.append([])
                i += 1
            elif i == 1:
                curve_o = line.split(',')
                __toFloat__(curve_o)
                i += 1
            else:
                values = line.split(',')
                __toFloat__(values)
                for j in range(len(values)):
                    value = values[j]
                    if value is not None:
                        k = j // 2
                        if j % 2 == 0:
                            Xs[k].append(value)
                        else:
                            Ys[k].append(value)
        for j in range(len(Data_Z)):
            X = Xs[j]
            Y = Ys[j]
            s = CurveT(X, Y, curve_o[j])
            z = Data_Z[j]
            Data.add(s, z)


## @brief Convert list of string to list of floats in place
#  @param L the input list
def __toFloat__(L):
    for i in range(len(L)):
        if L[i].strip():
            L[i] = float(L[i])
        else:
            L[i] = None
```

J Code for Partner's CurveADT.py

```
## @file CurveADT.py
#  @author Guiye Wu
#  @brief Provides the CurveT ADT class for representing a curve
#  @date 20 Feb 2018

from SeqServices import *
from Exceptions import *

## @brief An ADT that represents a curve, the clas only provides the information
##  of the curve, not modifying.
class CurveT:
    global MAX_ORDER
    # maximum order is 2
    MAX_ORDER = 2

    global DX
    # let dx be 1e-3
    DX = 1*10**(-3)

    ## @brief CurveT constructor
    #  @details Initilizes a curve using a sequence of x values, a sequence of
    #           y values and its order (1 or 2). The minimum value of x, the
    #           maximum value of x, the order of the curve and the function f(x)
    #           are initilized into minx, maxx, o and f respectively, the f(x)
    #           use interp function to find out the y value that corresponding to
    #           x.
    #  @param X The sequence of x values
    #  @param Y The sequence of y values
    #  @param i The order of the curve
    #  @exception IndepVarNotAscending Raise an error when the input sequence
    #             of x value is not ascending
    #  @exception SeqSizeMismatch Raise an error when the size of X and Y are not
    #             the same.
    #  @exception InvalidInterpOrder Raise an error when i is not 1 or 2
    def __init__(self,X,Y,i):
        if (not isAscending(X)):
            raise IndepVarNotAscending("Independent Variables are not ascending")
        if (len(X) != len(Y)):
            raise SeqSizeMismatch("X and Y are not the same size")
        if (i < 1 or i > MAX_ORDER):
            raise InvalidInterpOrder("Order is out of range")
        self.minx = X[0]
        self.maxx = X[len(X)-1]
        self.o = i
        self.f = lambda v : interp(X,Y,self.o,v)

    ## @brief Gets the minimum value of x in the x sequence
    #  @return The minimum x value
    def minD(self):
        return self.minx

    ## @brief Gets the maximum value of x in the x sequence
    #  @return The maximum x value
    def maxD(self):
        return self.maxx

    ## @brief Gets the order of the curve
    #  @return The the order of the curve
    def order(self):
        return self.o

    ## @brief Calculates and return the corresponding y value by the input x value
    #  @details The output value is calculated by f(x) which is initialized in
    #           CurveT.
    #  @param x The input x value
    #  @return The y value that corresponding to the input x
    def eval(self,x):
        if not (self.minx <= x and x <= self.maxx):
            raise OutOfDomain("The input value is not in the range")
        return self.f(x)

    ## @brief Calculateds the differential value by input x with DX
    #  @details Uses the formula (f(x+DX)-f(x))/DX with DX = 1e-3
    #  @param x The input value x
    #  @return The output dfdx which is the differential of x
    #  @exception OutOfDomain Raise an error when input x is out of the range of
    #               sequence x.
    def dfdx(self,x):
        if not (self.minx <= x and x <= self.maxx):
            raise OutOfDomain("The input value is not in the range")
        return (self.f(x+DX)-self.f(x))/DX

    ## @brief Calculateds the second differential value by input x with DX
    #  @details Uses the formula (f(x+2DX)-2f(x+DX)+f(x))/(DX*DX) with DX = 1e-3
    #  @param x The input value x
    #  @return The output d2fdx2 which is the second differential of x
    #  @exception OutOfDomain Raise an error when input x is out of the range of
    #               sequence x.
    def d2fdx2(self,x):
        if not (self.minx <= x and x <= self.maxx):
            raise OutOfDomain("The input value is not in the range")
        return (self.f(x+2*DX)-2*self.f(x+DX)+self.f(x))/(DX*DX)

## @brief The function interp which uses linear interpolation mehtod or quadratic
#          interpolation method.
#  @details If the curve order is 1, uses the linear interpolation to find the
#            approximation y value with x values sequence, y values sequence and
#            the input v, if the curve order is 2, uses the quadratic interpolation.
#  @param X The sequence of x values
#  @param Y The sequence of y values
#  @param o The order of the curve
#  @param v The input value v
#  @return The result by using linear or quadratic interpolation
def interp(X,Y,o,v):
    i = index(X,v)
    if o == 1:
        return interpLin(X[i],Y[i],X[i+1],Y[i+1],v)
    elif o == 2:
        return interpQuad(X[i-1],Y[i-1],X[i],Y[i],X[i+1],Y[i+1],v)
```

K Code for Partner's Data.py

```
## @file Data.py
#  @author Guiye Wu
#  @brief Provides a data class to store or get the curve
#  @date 20 Feb 2018

from CurveADT import *
from SeqServices import *
from Exceptions import *

## @brief A class Data which is used to store or get the information of curves
#  @details Uses a sequence S to store the curveT, and Z to store the values z
class Data:

    # maximum size is 10
    MAX_SIZE = 10

    # empty sequences
    S = []
    Z = []

    ## @brief Initializes the sequence S and Z to be empty
    def init():
        Data.S = []
        Data.Z = []

    ## @brief Adds a curveT to S sequence and a z value to Z sequence
    #  @param s The curveT
    #  @param z The z value
    #  @exception FULL Raises error when the size of S sequence is full
    #  @exception IndepVarNotAscending Raise error when the input z is less than
    #             the previous value
    def add(s,z):
        if len(Data.S) == Data.MAX_SIZE:
            raise FULL("Sequence of CurveT is full")
        if len(Data.Z) > 0:
            if z <= Data.Z[len(Data.Z)-1]:
                raise IndepVarNotAscending("Input value is too small")
        Data.S.append(s)
        Data.Z.append(z)

    ## @brief Gets the curveT in the index i of S sequence
    #  @param i The input index
    #  @return The curveT which in the index i of S sequence
    #  @exception InvalidIndex Raise error when the index is out of the range
    def getC(i):
        if not (0 <= i and i <= len(Data.S)-1):
            raise InvalidIndex("Index is not in the range")
        return Data.S[i]

    ## @brief Evaluate the value x with the index of z in Z and S sequence
    #  @details Uses linear interpolation to find the result, z is used to find
    #           the index of a curveT and a z values to do the calculation
    #  @param x The input value x
    #  @param z The input z value that in order to find the index in Z
    #  @return The result by using linear interpolation
    #  @exception OutOfDomain Raise error when input z is not in the range of
    #             Z sequence
    def eval(x,z):
        if not isInBounds(Data.Z,z):
            raise OutOfDomain("Input is out of domain")
        j = index(Data.Z,z)
        return interpLin(Data.Z[j],Data.S[j].eval(x),Data.Z[j+1],Data.S[j+1].eval(x),z)

    ## @brief Create a curveT
    #  @details The curveT is created with Z sequence to be x values and evaluation
    #           of input x of each curveT in S sequence to be y values and i to be
    #           the order
    #  @param x The input x value
    #  @param i The order of the curve
    #  @return A curveT
    def slice(x,i):
        Y = [Data.S[j].eval(x) for j in range(len(Data.Z))]
        return CurveT(Data.Z,Y,i)
```

L Code for Partner's SeqServices.py

```
## @file SeqServices.py
#  @author Guiye Wu
#  @brief Provides the some functions to support CurveADT
#  @date 20 Feb 2018

## @brief Checks if the sequence is ascending
#  @details By comparing each value to the next value in the sequence to check
#           if it is ascending
#  @param X The input sequence
#  @return Return false if the sequence is not ascending, otherwise return true
def isAscending(X):
    for i in range(len(X)-1):
        if (X[i+1] < X[i]):
            return False
    return True

## @brief Checks if a value in the range of a sequence
#  @param X The input sequence
#  @param x The input value
#  @return Return true if the value is in the range, false otherwise.
def isInBounds(X,x):
    return ((X[0] <= x) and (x <= X[len(X)-1]))

## @brief Linear interpolation
#  @details Uses formula (y2-y1)/(x2-x1)*(x-x1)+y1 to find the approximation of
#           y value that is corresponding to the input x value
#  @param x1 The input value x1
#  @param y1 The input value y1
#  @param x2 The input value x2
#  @param y2 The input value y2
#  @param x The input value x
#  @return The result y value
def interpLin(x1,y1,x2,y2,x):
    return (y2-y1)/(x2-x1)*(x-x1)+y1

## @brief Quadratic interpolation
#  @details Uses formula y1+(y2-y0)/(x2-x0)(x-x1)+(y2-2y1+y0)/(2(x2-x1)^2)(x-x1)^2
#           to find the approximation of y value that is corresponding to the
#           input x value
#  @param x0 The input value x0
#  @param y0 The input value y0
#  @param x1 The input value x1
#  @param y1 The input value y1
#  @param x2 The input value x2
#  @param y2 The input value y2
#  @param x The input value x
#  @return The result y value
def interpQuad(x0,y0,x1,y1,x2,y2,x):
    return y1+(y2-y0)/(x2-x0)*(x-x1)+(y2-2*y1+y0)/(2*(x2-x1)**2)*(x-x1)**2

## @brief Finds the index of x in sequence X
#  @param X The input sequence X
#  @param x The input value x
#  @return The index of x in X
def index(X,x):
    for i in range(len(X)-1):
        if ((X[i] <= x) and (x <= X[i+1])):
            return i
```

M Makefile

```
PY = pytest
PYFLAGS = --cov
DOXY = doxygen
DOXYCFG = doxConfig

RMDIR = rm -rf

.PHONY: test doc clean

test:
	$(PY) $(PYFLAGS) src

doc:
	$(DOXY) $(DOXYCFG)
	cd latex && $(MAKE)

clean:
	@- $(RMDIR) html
	@- $(RMDIR) latex
```