Assignment 2 Report

Lawrence Chung, chungl1

March 3, 2018

This report covers information about testing of the original and partner's code. The report also reveals any ambiguous points of the assignment that would need fixing if ever used in the industry.

1 Testing of the Original Program

Testing involved two categories: Tests that would throw exceptions and tests that would not. The tests that did not throw exceptions were judged as correct if the expected output occurred and inaccurate if expected output did not occur. For exceptions, Pytest classified those test cases as errors. One problem uncovered through testing was that the specification was not clear about formatting floating point values. This problem will be further discussed in the report.

All testing followed the same criteria. Failed tests are tests that gave unexpected output or threw exceptions. Successful test are tests that output the expected values.

2 Results of Testing Partner's Code

Most of the results of my partner's code was expected. That is, exceptions were thrown when expected, but some methods behaved differently in comparison to the original code. There were a total of 13 failed tests, and 11 passed tests.

3 Discussion of Test Results

The results of the original code yielded 10 failed tests and 14 passed tests. Most of the testing resulted in expected output with the exception of a few. The unexpected output will be further discussed later in the report.

3.1 Problems with Original Code

The unexpected output stemmed from, as mentioned earlier, CurveT.eval, CurveT.dfdx and Curve.df2dx2. The reason for this unexpected output was the lack of clarity in the specification about formatting of floating point values. For example, if an expected value was 4, the output would be 3.99999999. The value isn't necessarily incorrect, especially in the context of this assignment, but because there was no specification about formatting, this result is unexpected and was deemed as a failed test.

3.2 Problems with Partner's Code

Some of my partner's unexpected output stemmed from the same issue of formatting. However, there were some other cases of unexpected output. There were some attribute errors that were unexpected that prevented Data.eval from running. Another error was from SeqServices.index, where the output was not expected and led to an assertion error. The last error that was different from the original code was from a test for non-Ascending sequence for SeqServices.index. However, my partner code was still correct because there was no mention about exception handling for that method.

3.3 Problems with Assignment Specification

The main issue with the Assignment specification is the formatting of the floating point values. In comparison to assignment 1, the specification is more formal, but the Load.py and Plot.py would require a more formal specification. It was explained in words, so the potential for ambiguity still exists.

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?

   $out := (\neg isAscending \Rightarrow IndepVarNotAscending | X_0 \leq x \leq X_{|X|-1})$

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

   In the constructor, change the exception to only occur if order is greater than 3.

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.

   The module interface is essential.

   It is consistent to a high degree, with a few exceptions. The method signatures are the same, but performance is not on par across different code.

   The interface is general, in that it cannot be predicted how it will be used. In this assignment, it is used to plot graphs, but it has other characteristics that allow it to be used for another purpose.

   The interface is minimal. Each method in the ADT performs one task for the user.

   The interface is not as opaque as it should be. It does not mention anything specific about which variables should not be accessible to the user.

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.

   In terns of consistency, essentiality, generality, minimality and opaqueness, Data and CurveADT are similar. Main difference is that Data.py is more general than

CurveADT. Data provides many methods purely on data manipulation, but data manipulation can be used in many contexts.

E Code for CurveADT.py

```
## @file CurveADT.py
# @title CurveADT
# @author Lawrence Chung
# @brief ADT for a Curve

from scipy.interpolate import interp1d
from Exceptions import *
from SeqServices import *

MAX_ORDER = 2
DX = 1*(10**(-3))

class CurveT:

        ## @brief Constructor for CurveT
        # @param X Sequence of x values
        # @param Y Sequence of Y values
        # @param i the order of the curve
        def __init__(self, X, Y, i):

                if not isAscending(X):
                                raise IndepVarNotAscending("X values not Ascending")
                if(len(X) != len(Y)):
                        raise SeqSizeMismatch("Sizes of arrays do not match")
                if(i > MAX_ORDER):
                        raise InvalidInterpOrder("Order of function too great")

                self.x = X
                self.y = Y
                self.i = i
                self.f = interp1d(X, Y, i)

                self.minx = self.x[0]
                self.maxx = self.x[len(self.x)-1]

        ## @brief determines minimum value in sequence
        # @return the first element in the Sequence
        def minD(self):
                return self.minx

        ## @brief determines maximum value in sequence
        # @return the last element in the Sequence
        def maxD(self):
                return self.maxx

        ## @brief determines the order of the Curve
        # @return the third argument of the Constructor
        def order(self):
                return self.i

        def interp(self, X, Y, i, v):
                index = index(X, v)
                if(i == 1):
                        return interpLin(X[index], Y[index], X[index+1], Y[index+1], v)
                if(i == 2):
                        return interpQuad(X[index-1], Y[index-1], X[index], Y[index], X[index+1],
                                Y[index+1], v)

        ## @brief Evaluates curve at an x value
        # @details Uses interp1d from scipy module to evaluate
        # @param x the x value at which the Curve will be evaluated
        def eval(self, x):
                if(x < self.minD() or x > self.maxD()):
                        raise OutOfDomain("The x value is out of domain")

                return self.f(x)

        ## @brief Approximates first derivative
        # @details Uses formula of Forwarded Divided Difference to approximate
        # @param x The x value at which the curve will be evaluated
        def dfdx(self, x):
                if(x < self.minD() or x > self.maxD()):
                        raise OutOfDomain("The x value is out of domain")

                return ((self.f(x + DX) - self.f(x))/DX)
```

```
## @brief Approximates second derivative
# @details Uses formula of Forwarded Divided Difference to apprximate
# @param x The x value at which the curve will be evaluated
def d2fdx2(self, x):
        if(x < self.minD() or x > self.maxD()):
                raise OutOfDomain("The x value is out of domain")

        return ((self.f(x + 2*DX) - 2*self.f(x + DX) + self.f(x))/DX**2)
```

F Code for Data.py

```
## @file Data.py
# @title Data
# @author Lawrence Chung
# @brief Manipulate data

from Exceptions import *
from CurveADT import CurveT
from SeqServices import interpLin
from SeqServices import index
from SeqServices import isInBounds

MAX_SIZE = 10

class Data:

        ## @brief Initializes abstract object
        # @details Initiializes two empty sequences, the independent and dependent sequences
        @staticmethod
        def init():
                Data.S = []
                Data.Z = []

        ##@brief adds element to sequenes
        # @details Adds a sequences of CurveT to S and a real value z to Z
        # @param s element to add to sequence S
        # @param z element to add to sequence Z
        @staticmethod
        def add(s, z):

                if(len(Data.S) == MAX_SIZE):
                        raise Full("Sequence is full")
                if(len(Data.Z) > 0 and z < Data.Z[len(Data.Z)-1]):
                        raise IndepVarNotAscending("Z values not ascending")

                Data.S.append(s)
                Data.Z.append(z)

        ## @brief Method to obtain a value in the sequence
        # @param i Indexing value
        # @return Corresponding value in the sequence at the provided index
        @staticmethod
        def getC(i):

                if(i < 0 or i >= len(Data.S)):
                        raise InvalidIndex("Index out of range")

                return Data.S[i]

        ## @brief Method to evaluate curve
        # @details Uses linear interpolation from SeqServices.py to determine the corresponding y value
        # @param x independent variable x
        # @param z x value from Z sequence
        @staticmethod
        def eval(x, z):

                if not (isInBounds(Data.Z, z)):
                        raise OutOfDomain("Value is not in domain")

                j = index(Data.Z, z)

                return interpLin(Data.Z[j], Data.S[j][1], Data.Z[j+1], Data.S[j+1][1], z)

        ## @brief Slices a curve
        # @param x the value at which the method will slice
        # @param i order of curve
        @staticmethod
        def slice(x, i):

                Y = []
                j = 0
                while(j < len(Data.S)):
                        Y.append(int(Data.S[j][1]))
                        j += 1

                return CurveT(Data.Z, Y, i)
```

G Code for SeqServices.py

```
## @file SeqServices.py
# @title SeqServices
# @author Lawrence Chung
# @brief Sequence operations


## @brief determines whether sequence is ascending
# @param X a sequence in the form of a list
# @return boolean, true if ascending, false if not
def isAscending(X):
        j = 0
        i = 0
        while(j < len(X)-1):
                if(X[j+1] >= X[j]):
                        i += 1
                j += 1
        if(i == len(X)-1):
                return True
        else:
                return False

## @brief determines if input value is in sequence
# @param X a Sequence of values
# @param x value to be checked
# return boolean, True if in bounds, False if not
def isInBounds(X, x):
        if(x < X[0] or x > X[len(X)-1]):
                return False
        else:
                return True

## @brief linear interpolation
# @details Determines the corresponding y using formula
# @param x1 independent variable of first set of poitns
# @param y1 dependent variable of first set of points
# @param x2 independent variable of second set of points
# @param y2 dependent variable of second set of points
# @param x the x value that will be interpolated
# @return the y value determined through interpolation
def interpLin(x1, y1, x2, y2, x):
        tempVal = (y2 - y1)/(x2 - x1)
        val = tempVal*(x - x1) + y1
        return val

## @brief Quadratic interpolation
# @details Determines the corresponding y using formula
# @param x0 independent variable of first set of poitns
# @param y0 dependent variable of first set of points
# @param x1 independent variable of second set of points
# @param y1 dependent variable of second set of points
# @param x2 independent variable of third set of points
# @param y2 dependent variable of third set of points
# @param x the x value that will be interpolated
# @return the y value determined through interpolation
def interpQuad(x0, y0, x1, y1, x2, y2, x):
        tempVal = ((y2 - y0)/(x2- x0))*(x - x1)
        tempVal2 = ((y2 - 2*y1 + y0)/(2*(x2-x1)**2))*(x-x1)**2
        val = y1 + tempVal + tempVal2
        return val

## @brief Determines index of value in sequence
# @param X A sequence of x values
# @param x a value in Sequence X
# @return the index of the x value
def index(X, x):
        if(isInBounds(X,x) and isAscending(X)):
                i = 0
                while(i < len(X)):
                        if(X[i] == x):
                                return i
                        i += 1
```

H Code for Plot.py

```
## @file Plot.py
# @title Plot
# @author Lawrence Chung
# @brief Module to plot graph

from CurveADT import CurveT
from Exceptions import *
import matplotlib.pyplot as plt

## @brief Plots Sequences of curve
# @details uses maplotlib to plot graph
# @param X Sequence of x values
# @param Y Sequence of y values
def PlotSeq(X, Y):

        if(len(X) != len(Y)):
                raise SeqSizeMismatch("X and Y are different lengths")

        plt.plot(X, Y, 'b')
        plt.show()

## @brief Plot Sequences of curve based on size of n
# @details uses matplotlib to plot graph
# @param c A curve object
# @param n Value determining amount of spaced points
def PlotCurve(c, n):

        X = []
        Y = []
        interval = (c.maxD() - c.minD())/n

        if(c.order() == 1):
                j = 0
                i = c.minD()
                while(i <= n-2):
                        X.append(i)
                        i += interval
                for j in X:
                        Y.append(c.eval(j))

        elif(c.order() == 2):
                i = c.minD() + interval
                j = 0
                while(i <= n-2):
                        X.append(i)
                        i += interval
                for j in X:
                        Y.append(c.eval(j))

        plt.plot(X, Y, 'b')
        plt.show()
```

I Code for Load.py

```
## @file Load.py
# @title Load
# @author Lawrence Chung

from CurveADT import *
from Data import *

## @brief Method to write data from file
# @details Uses loops to write data into Sequence, filling empty slots with commas
# @param s file name of type string
def Load(s):

        Data.init()
        f = open(s, 'r')
        S = []
        X = []
        Y = []
        Z = []
        k = 0
        stringVal = ""
        for i in f:
                l = 0
                m = 0
                j = 0
                if(k == 0):
                        while(l < len(i)):
                                if(i[l] == "," or i[l] == "\n"):
                                        Z.append(float(stringVal))
                                        stringVal = ""
                                        l += 1
                                elif(i[l] != ","):
                                        stringVal += i[l]
                                        l += 1

                if(k >= 2):
                        while(l < len(i)):
                                if(i[l] == "," or i[l] == "\n"):
                                        if(j % 2 == 0):
                                                if(k == 2):
                                                        X.append([float(stringVal)])
                                                else:
                                                        if(stringVal == " "):
                                                                X[m].append(",")
                                                        else:
                                                                X[m].append(float(stringVal))
                                        else:
                                                if(k == 2):
                                                        Y.append([float(stringVal)])
                                                else:
                                                        if(stringVal == " "):
                                                                Y[m].append(",")
                                                        else:
                                                                Y[m].append(float(stringVal))
                                                m += 1
                                        j += 1
                                        stringVal = ""
                                        l += 1
                                elif(i[l] != ","):
                                        stringVal += i[l]
                                        l += 1


                k += 1

        S.append(X)
        S.append(Y)
        Data.add(S, Z)
```

J Code for Partner's CurveADT.py

```
## @file CurveADT.py
# @title CurveT
# @author Sophia Tao
# @brief Abstract data type representing a curve.
# @date Feb 20, 2018

import SeqServices
from Exceptions import OutOfDomain, SeqSizeMismatch, IndepVarNotAscending, InvalidInterpOrder

MAX_ORDER = 2 # state constant for maximum order of curve
DX = 0.001 # state constant for DX

## @brief finds the interpolation of the function.
# @param X list of X values.
# @param Y list of Y values.
# @param o the order of interpolation.
# @param v the value to interpolate at.
# @return the value of the interpolation.
def interp(X,Y,o,v):
    i = SeqServices.index(X,v)
    if (o == 1):
        return SeqServices.interpLin(X[i], Y[i], X[i+1], Y[i+1], v)
    else:
        return SeqServices.interpQuad(X[i], Y[i], X[i+1], Y[i+1], X[i+2], Y[i+2], v)

class CurveT:

    ## @brief Constructor for CurveT class.
    # @param self The CurveT object pointer.
    # @param X the list of x values.
    # @param Y the list of y values.
    def __init__(self, X, Y, i):
        print(X)
        if (len(X) != len(Y)):
            raise SeqSizeMismatch("length of X and Y not equal")
        if not SeqServices.isAscending(X):
            raise IndepVarNotAscending("X values not ascending")
        if i < 1 or i > MAX_ORDER:
            raise InvalidInterpOrder("Order not 1 or 2")
        self.__o = i
        self.__minx = X[0]
        self.__maxx = X[len(X)-1]
        self.__f = lambda v : interp(X, Y, i, v)

    ## @brief getter for minimum X value.
    # @return the curve's minimum X value.
    def minD(self):
        return self.__minx

    ## @brief getter for maximum X value.
    # @return the curve's maximum X value.
    def maxD(self):
        return self.__maxx

    ## @brief getter for the curve's order.
    # @return the curve's order.
    def order(self):
        return self.__o

    ## @brief retrieves function for interpolating the curve at a value..
    # @param x x-value to be used in interpolation.
    # @return a function for interpolating the curve at x.
    def eval(self, x):
        if not (self.__minx <= x) or not (x <= self.__maxx):
            raise OutOfDomain("x out of domain")
        return self.__f(x)

    ## @brief finds first derivative at x.
    # @param x x-value to be used.
    # @return the first derivative at x.
    def dfdx(self,x):
        if not (self.__minx <= x) and (x <= self.__maxx):
            raise OutOfDomain("x out of domain")
        return (self.__f(x+DX)-self.__f(x))/DX

    ## @brief finds second derivative at x.
```

```
# @param x x-value to be used.
# @return the second derivative at x.
def d2fdx2(self,x):
    if not (self.__minx <= x) and (x <= self.__maxx):
        raise OutOfDomain("x out of domain")
    return (self.__f(x+2*DX)-2*self.__f(x+DX)+self.__f(x))/(DX**2)
```

K Code for Partner's Data.py

```
## @file Data.py
# @title Data
# @author Sophia Tao
# @brief Methods for manipulating data.
# @date Feb 20, 2018

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

class Data:
    # state variables
    MAX_SIZE = 10 #maximum size of a curve
    S = [] #list of CurveT objects
    Z = [] #list of curve values

    ## @brief Initializer
    # @details Initializes two sequences for the independent and dependent variables.
    @staticmethod
    def init():
        Data.S = []
        Data.Z = []

    ## @brief Method for adding data values.
    # @details Adds a CurveT to list S and curve value z to list Z.
    # @param s the element to add to S.
    # @param z The element to add to Z.
    @staticmethod
    def add(s,z):
        if (len(Data.S) == Data.MAX_SIZE): raise Full('The list is full')
        if (len(Data.Z) > 0 and z <= Data.Z[len(Data.Z)-1]):
            print(z)
            print(Data.Z[len(Data.Z)-1])
            raise IndepVarNotAscending('Independent variables added must be greater than previous')
        Data.S.append(s)
        Data.Z.append(z)

    ## @brief Method for retrieving a member of the sequence.
    # @param i the index to add.
    # @return the value of the sequence at given index.
    @staticmethod
    def getC(i):
        if i < 0 or i >= len(Data.S): raise InvalidIndex("Index not valid")
        return Data.S[i]

    ## @brief Method for evaluating a curve at a particular x-value.
    # @details Uses linear interpolation to find the value of the curve.
    # @param x the value to evaluate the curve at.
    # @param z An adjacent x value.
    @staticmethod
    def eval(x,z):
        if not isInBounds(Data.Z,z): raise InvalidIndex("The independent variable is not in bounds")
        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 Method for slicing a curve.
    # @param x the x to slice at
    # @param i the order of the curve
    @staticmethod
    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
# @title SeqServices
# @author Sophia Tao
# @brief Methods for sequence operations.
# @date Feb 20, 2018


## @brief tells whether a sequence is ascending.
# @param X a list.
# @return boolean of whether X is ascending

def isAscending(X):
    return all(X[j] <= X[j+1] for j in range(len(X)-1))

## @brief tells whether an item is within range of sequence.
# @details whether the item is greater than the minimum and less than the maximum.
# @param X the index.
# @param x the value.
# @return boolean of whether x is within bounds of the list X.
def isInBounds(X, x):
    return x <= X[len(X)-1] and x >= X[0]

## @brief Finds linear interpolation at a point.
# @details Uses formula (y2-y1)/(x2-x1)*(x-x1)+y1.
# @param x1 the first known x value.
# @param x2 the second known x value.
# @param y1 the first known y value.
# @param y2 the second known y value.
# @param x the x-point at which to extrapolate from.
# @return the linear interpolation evaluated at x.
def interpLin(x1,y1,x2,y2,x):
    return (y2-y1)/(x2-x1)*(x-x1)+y1

## @brief Finds quadratic interpolation at a point.
# @details Uses formula y1+(y2-y0)/(x2-x0)*(x-x1) + (y2-2*y1+y0)/(2*(x2-x1)**2)*(x-x1)**2
# @param x0 the first known x value.
# @param x1 the second known x value.
# @param x3 the third known x value.
# @param y0 the first known y value.
# @param y1 the second known y value.
# @param y2 the third known y value.
# @param x the x-point at which to extrapolate from.
# @return the linear interpolation evaluated at x.
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 index of the term immediately less than given value.
# @param X The sequence.
# @param x The value to find the index of.
# @return The index of the term immediately less than given value. None if there is an error.
def index(X, x):
    for index, item in enumerate(X):
        if(item <= x):
            if X[index+1] >= x:
                return index
```

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
```