Assignment 2 Report

Your Name and MACID

February 26, 2018

Introductory blurb.

1 Testing of the Original Program

Description of approach to testing. Rationale for test case selection. Summary of results. Any problems uncovered through testing.

2 Results of Testing Partner's Code

Summary of results.

3 Discussion of Test Results

3.1 Problems with Original Code

3.2 Problems with Partner's Code

3.3 Problems with Assignment Specification

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?

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

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.

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.

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

K Code for Partner's Data.py

L Code for Partner's SeqServices.py

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
```