Assignment 2 Report

Your Name and MACID

March 3, 2018

Introductory blurb.

1 Testing of the Original Program

Each method for CurveADT.py, SeqService.py, and Data.py were tested individually while using assertions to check if the desired result was given after calling each method. Everything ran as expected.

2 Results of Testing Partner's Code

Testing the file without making any changes to the original test_ALL.py results in errors because my partner named the variables for the x and y data a different name. In addition, my partner has interpretted the Data class to not be an ADT, and instead use static methods to change Data's variable S and Z. After making the appropriate changes to my test_All.py and placing these changes in test_All_partner.py, all test cases passed except for index() under the class SeqServices.

3 Discussion of Test Results

3.1 Problems with Original Code

After seeing the partner codes, I have realized that I have misinterpreted the Data class specificataion and wrote the class as if it was an ADT.

3.2 Problems with Partner's Code

The only thing wrong with my partner's code was the method index() under the class SeqServices. Upon inspection, index() returned one less than the desired index. For example, the index of 5 in list [1, 3, 5, 7] is two, but index() would return one.

3.3 Problems with Assignment Specification

The only problem with the assignment specifications was for the Data class. The constructor for the data class is assumed to initialize the parameters for an object of Data, rather than re-inizializing varibales of the class Data. Although one can interpret it correctly by realizing Data.py is not named DataADT.py like CurveADT.py, if someone does not read the specifications carefully enough, they can initially assume that Data.py holds code for an ADT.

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?

   First, the exported Constant MAX_ORDER will be changed to 3. Second, interp(X, Y, o, v) will be modifiied with an added condition when o = 3. The added condition will run through a cubic interpolation formula when o = 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.

   CurveADT's module's interface:
   - is consistent because each method name corresponds to the behavior of each respective method.
   - is essential because unnecessary access programs such as interp(X, Y, o, v) are omitted.
   - is not general because each method is specific for lists of x and y data points that assumes the lists are ascending.
   - is minimal because every method does exactly one thing; there are no methods that provide multiple services.
   - is opaque because each method can only be accessed by an object of CurveT, and

the data structure of an object of CurveT can only accessed by accessor methods in the CurveT class; this enforces information hiding.

4. What is your critique of the Data abstract object's interface. In

   Data's module's interface:
   - is consistent because each method name corresponds to the behavior of each respective method
   - is enssential because there are no unnecessary access programs to omit
   - is not general because it is assumed that the lists under the Data class have equal length. (i.e. the method add())
   - is minimal because every method does exactly one thing; there are no methods that provide multiple services.
   - is not opaque because every method can be accessed so long as the Data class has been imported. Every variable under the Data class such as S and Z is also easily accessed without any accessor methods.

E Code for CurveADT.py

```
## @file CurveADT.py
#  @author Stephanus Jonatan
#  @date February 20, 2018

from scipy import interpolate
from SeqServices import *
from Exceptions import *

MAX_ORDER = 2
DX = 0.001

## @brief Determines which interpolation formula to use
#  @details Returns the estimated y value for a given value
#  @param X is sequence of X-points
#  @param Y is sequence of Y-points
#  @param o is degree of interpolation
#  @param v is the given value
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)

## @brief CurveT is a class that analyzes data of a polynomial.
#  @details CurveT uses interpolation to estimate a y value for a given x value.
#  @details CurveT the scipy library.
class CurveT(object):

    ## @brief Initializes a interpolation function.
    #  @details Uses a given list of x and y coordinates, and a degree of interpolation to create the
        interpolation function.
    #  @param X is the list of x coordinates.
    #  @param Y is the list of y coordinates.
    #  @param i is the degree of interpolation.
    def __init__(self, X, Y, i):
        if not isAscending(X):
            raise IndepVarNotAscending("X is not in ascending order")
        if len(X) == len(Y):
            self.x_data = X
            self.y_data = Y
        else:
            raise SeqSizeMismatch("The number of X-points does not equal the number of Y-Points.")
        if i <= MAX_ORDER:
            self.o = i
            self.f = interpolate.interp1d(self.x_data, self.y_data, self.o)
        else:
            raise InvalidInterpOrder("The degree of interpolation is more than %d" % MAX_ORDER)

    ## @brief minD() returns the smallest element in list self.x_data.
    def minD(self):
        return min(self.x_data)

    ## @brief maxD() returns the largest element in list self.x_data.
    def maxD(self):
        return max(self.x_data)

    ## @brief order() returns the degree of interpolation.
    def order(self):
        return self.o

    ## @brief eval(x) returns an approximate y value for a given x value by interpolation.
    #  @details The given x value must be within the list of x_data.
    #  @param x is the given value used to approximate a y value.
    def eval(self, x):
        if self.minD() <= x <= self.maxD():
            y = self.f(x)
            return y
        else:
            raise OutOfDomain("%f is not within the range" % x)

    ## @brief dfdx(x) returns the value y at a given x value for the derivative of the interpolation
        function.
    #  @param x is the given x value.
    def dfdx(self, x):
        if self.minD() <= x <= self.maxD():
```

```
            return (self.f(x + DX) - self.f(x)) / DX
        else:
            raise OutOfDomain("%f is not within the range" % x)

    ## @brief dfdx2(x) returns the value y at a given x for the second derivative of the interpolation
        function.
    #  @param x is the given x value.
    def dfdx2(self, x):
        if self.minD() <= x <= self.maxD():
            return (self.f(x + 2*DX) - 2*self.f(x + DX) + self.f(x)) / pow(DX, 2)
        else:
            raise OutOfDomain("%f is not within the range" % x)
```

F Code for Data.py

```
## @file Data.py
#  @author Stephanus Jonatan
#  @date February 20, 2018

from CurveADT import *

MAX_SIZE = 10
## @brief Data is a class that takes in data of a polynomial into a list.
#  @details Data
#  @details Data uses the classes CurveT, and SeqServices for interLin and index.
class Data(object):

    ## @brief Initializes two empty lists.
    #  @param S is a sequence of CurveT
    #  @param Z a sequence of real numbers
    def __init__(self):
        self.S = []
        self.Z = []

    ## @brief Mutates two lists S and Z and concactenates elements s and z.
    #  @details Checks if list S is full and if the last element in Z is bigger or equal to element z
    #  @param s is an element being added to list S
    #  @param z is an element being added to list Z
    def Data_add(self, s, z):
        if len(self.S) == MAX_SIZE:
            raise Full("The sequence full. Cannot add anymore data")
        elif len(self.Z) > 0 and z <= self.Z[len(self.Z) - 1]:
            raise IndepVarNotAscending("Element %f is smaller than the last value of Z, %f" % (z,
                self.Z[len(self.Z) - 1]))
        else:
            self.S.append(s)
            self.Z.append(z)

    ## @brief Accesses list S of index i.
    #  @details Checks if list S has index i.
    #  @param i is the index being accessed.
    def Data_getC(self, i):
        if (0 <= i & i <= len(self.S)):
            return self.S[i]
        else:
            raise InvalidIndex("Index (%d) does not exist" % i)

    ## @brief Returns an estimated y value for a given z value using Interpolation.
    #  @param x is the
    #  @param z is the value used to determine the approximate index of list Z.
    def Data_eval(self, x, z):
        if isInBounds(self.Z, z):
            j = index(self.Z, z)
            return interpLin(self.Z[j], self.S[j].eval(x), self.Z[j + 1],
                             self.S[j + 1].eval(x), z)
        else:
            raise OutOfDomain("z is out of the domain of Z")


    ## @brief Returns CurveT with list Z as the x data and maps .
    #  @param x is the
    #  @param z is the value used to determine the approximate index of list Z.
    def Data_slice(self, x, i):
        Y = []
        for a in range(0, len(self.Z), 1):
            Y.append(self.S[a].eval(x))
        return CurveT(self.Z, Y, i)
```

G Code for SeqServices.py

```
## @file SeqServices.py
#  @author Stephanus Jonatan
#  @date February 20, 2018

## @brief Checks if sequence X consists of ascending data points.
#  @param X is the sequence being checked
def isAscending(X):
    if sorted(X) == X:
        return True
    else:
        return False

## @brief Checks if the value x is within sequence X.
#  @param X is the sequence being checked
#  @param x is the value being checked
def isInBounds(X, x):
    if (X[0] <= x & x <= X[len(X)-1]):
        return True
    else:
        return False

## @brief Estimates a y value for a given x value using linear interpolation.
#  @details SeqServices
#  @param x1 is a known data point; either smaller or exactly equal to x
#  @param y1 is the corresponding y value for x1
#  @param x2 is a known data point; either exactly equal of larger than x
#  @param y2 is the corresponding y value for x2
#  @param x is the value used to estimate y
def interpLin(x1, y1, x2, y2, x):
    return (y2-y1)*(x - x1)/(x2 - x1) + y1

## @brief Estimates a y value for a given x value using quadratic interpolation.
#  @param x0 is a known data point; smaller than x
#  @param y0 is the corresponding y value for x0
#  @param x1 is a known data point; either smaller or exactly equal to x
#  @param y1 is the corresponding y value for x1
#  @param x2 is a known data point; either exactly equal of larger than x
#  @param y2 is the corresponding y value for x2
#  @param x is the value used to estimate y
def interpQuad(x0, y0, x1, y1, x2, y2, x):
    t1 = (y2-y0)*(x-x1)/(x2-x0)
    t2 = (y2-2*y1+y0)*((x-x1)**2)/(2*(x2-x1)**2)
    return y1 + t1 + t2

## @brief Estimates the index of a value x in sequence X.
#  @details Assumes that sequence X is sorted with ascending order.
#  @param X is the sequence being checked
#  @param x is the value
def index(X, x):
    for i in range(0, len(X)-1):
        if (X[i] <= x and x < X[i+1]):
            return i
    return None
```

H Code for Plot.py

```
## @file Plot.py
#  @author Stephanus Jonatan
#  @date February 20, 2018

from CurveADT import *
import matplotlib.pyplot as win

## @brief Returns a x-y graph.
#  @details Data points are in sequences X and Y.
#  @param X is a sequence filled with x data points
#  @param Y is a sequence filled with y data points
def PlotSeq(X, Y):
    if len(X) != len(Y):
        raise SeqSizeMismatch("The sequences are not the same size")
    win.plot(X, Y, 'b')
    win.xlabel("x-axis")
    win.ylabel("y-axis")
    win.show()

## @brief Returns a x-y graph of curve c.
#  @details plots c at n equally spaced points
#  @param c is a curve of CurveT
#  @param n is the number of points inbetween each plotted data point of c
def PlotCurve(c, n):
    interval = (c.maxD() - c.minD()) / n
    X_data = []
    Y_data = []
    if c.order() == 1:
        for i in range(c.minD(), c.maxD(), interval):
            Y_data.append(c.eval(i))
    elif c.order() == 2:
        for i in range(c.minD(), c.maxD(), interval):
            Y_data.append(c.eval(i))
    X_data += range(c.minD(), c.maxD(), interval)
    PlotSeq(X_data, Y_data)
```

I Code for Load.py

```
## @file Load.py
#  @author Stephanus Jonatan
#  @date February 20, 2018

import csv
from Data import *

## @brief Load is a class that reads a csv file.
#  @details The csv file contains data on a set of curves and loads each curve into Data
#  @details z stores the value of each curve.
#  @details o stores the order of each curve
#  @details x stores the x-points of each curve
#  @details y stores the y-points of each curve
#  @param s is the name of the csv file (expected .csv extension)
def Load(s):
    Data.__init__()
    z = []
    o = []
    x = []
    y = []
    with open(s) as csvfile:
        readCSV = csv.reader(csvfile, delimiter = ',')
        count = 0
        for row in readCSV:
            if count == 0:
                for i in range(0, len(row), 1):
                    z.append(float(row[i]))
                m = len(z)
                x = [[0 for a in range(1)] for b in range(m)]
                y = [[0 for a in range(1)] for b in range(m)]
                count += 1
            elif count == 1:
                for i in range(0, len(row), 1):
                    o.append(float(row[i]))
                count += 1
            else:
                data_list = 0
                for i in range(0, 2*m - 1, 2):
                    add_x = row[i]
                    add_y = row[i + 1]
                    if add_x != '' or add_y != '':
                        x[data_list].append(float(add_x))
                        y[data_list].append(float(add_y))
                    data_list += 1
            for i in range (0, m, 1):
                Data.Data_add(CurveT(x[i],y[i],o[i]), z[i])
```

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
```