Assignment 2 Report

Ninos Yomo — yomon

March 2, 2018

1 Testing of the Original Program

My approach to testing was simple and systematic. I would test each individual module, and from each module, I would test the functionality of its methods. For each method, if applicable, I would test a general case, a boundary case and an exception case. The testing of boundary cases was a rational decision as I was able to uncover that my index() function from SeqServices.py was not fully correct as it would fail on the boundary case. The exception cases were required to confirm that the module was correctly raising errors when expected. Other than my boundary case for index(), the functionality of my other methods were correct.

2 Results of Testing Partner's Code

The SeqServices passed all cases. Although my test module raised an error for index(), my partner's implementation was correct.

The CurveADT passed in all cases except for eval() because of the different implementation of the local function, where my test module assumes scipy was used and implements a scipy method that would crash for different implementations.

The getC method in Data does not work for indexs of 0 since my partner defined it to be out of bounds even though indexing starts at 0 and the specification defined it in bounds. The eval method for Data did not work as the calculation did not output a value. Finally, the slice method did not work for the same reason eval in CurveADT did not work, the different implementation of the local function f caused .take(0) to raise an error.

3 Discussion of Test Results

3.1 Problems with Original Code

With my program, my exceptions were done differently then what most people would do. Instead of raising an exception while passing a string message, I would raise the exception and have the exceptions class come with a preset message. This would prove to be incompatible with my partner's code as they would attempt to pass a string message.

My implementation for index() in SeqServices was incorrect for the right boundary case since it returns an index of the last element when it should be the index of the element before it to maintain the definition of the specification.

3.2 Problems with Partner's Code

At first, the test module would not work because of the way SeqServices.py was implemented by my partner. A class was made where the methods were defined to be static, which treated the SeqServices as an abstract object while I defined it to simply be a module of usable methods. The requirements specification did not explicitly say if it was to be an abstract object or not. However from context, it is implied that it is not a abstract object.

Another problem was the inconsistency of the import statements. I used methods from imports by calling from x import *, whereas my partner would import specifically what was needed, because of this, the way those methods are called are different and the test module would crash.

My partner incorrectly implemented getC from Data by treating the 0 index as out of bounds even though the requirements specification defined it to be in bounds.

3.3 Problems with Assignment Specification

The assignment specification gave too much freedom when it came to the implementation of the local function f in CurveADT. We could have implemented it using scipy or SeqServices. However, for the scipy implementation, the function .take(0) was used to convert the returned array into a single value, this would work for all scipy implementations however, those who chose to implement using SeqServices as did my partner would find that the test would fail.

The specification did not define a consistent way to import the modules used in other modules. Because of this, we used different ways of importing module that effected the way we wrote our code and thus raised errors during testing because of the inconsistency.

Aside from these slight differences, the formality of the requirements specification of A2 compared to A1 greatly improved the overall ability to implement the program. With it, many ambiguous details were covered which allowed for consistent implementations. An example is the use of discrete math notation to describe the functionality of methods, this allows for no misinterpretation, unlike the previous assignment who's ambiguous description caused inconsistency in implementations.

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 that the sequence is ascending was removed, this would mean the current mathematical specification of: $X_0 \leq x \leq X_{|X|-1}$ would be incorrect. The new specification must explicitly state the minimum and maximum values of the sequence instead of relying on the positional application of the ascending property. This would be: $min(X) \leq x \leq max(X)$ where min and max represent the minimum and maximum values of the sequence X respectively. An alternative way using no functions would be to say: $\exists x : \mathbb{R} \bullet (\neg(\forall y : \mathbb{R} | y \in S \bullet x < y) \wedge \neg(\forall z : \mathbb{R} | z \in S \bullet x > z))$ for the input argument x and the sequence S. If this is true, that means that the argument x is not less than all of the elements in the sequence which means it is greater than at least one element, this proves that it is within the lower bounds. The second half states that it is not greater than all the elements meaning it is less than at least one element and therefore proves it is within the upper bounds. Since it is within both boundaries, it must be within the sequence.

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

   To implement cubic interpolation, this would require that CurveADT.py has its MAX_ORDER value become 3 instead of 2 to allow for cubic functions to be inputted. This would be the only requirement since the rest will be handled by scipy.interp1d() because of the fact that it supports cubic interpolations.

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.

   Firstly, the interface is certainly essential as no existing method's functionality can be recreated with another in the same module. It is also minimal since all of the methods in CurveADT strictly perform only their sole intended task. However, the boundaries could have been better defined as some methods may not work for them like solving for the differentials on the boundary points since it requires you to evaluate for a point out of bounds.

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.

   The most obvious critique of Data is the lack of a method that deletes a curve from Data. It is known that Data has a very limited capacity of 10 curves, because of this, it would be useful for the module to implement a deletion method to make room for new curves. Another detail is the lack of a getter method for the size of Data. Without it, checking for the Full() exception is made awkward as it is done by manually examining the length of Data, while a simpler size() function can help clean up the code and check for the Full() exception ahead of time.

E Code for CurveADT.py

```python
## @file CurveADT.py
# @title CurveT
# @author Ninos Yomo
# @date 02/20/2018

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

## @brief Creates a curve object.
# @details This class creates a curve object that has two sequences X and Y which holds the X and Y
    values of the curve respectively, it also stores the order o
# and has a function f which interpolates the value of the curve at a point when passed as an argument
    to f. Interpolates in a linear and quadratic method based
# on the value of o which determines if the order is linear or quadratic.
class CurveT:
    ## MAX_ORDER the maximum order of a curve.
    MAX_ORDER = 2
    ## DX the constant used to estimate derivatives.
    DX = 0.001

    ## @brief Creates a CurveT object.
    # @details The constructor for the CurveT class. Creates a Curve object by setting its parameters
        such as minx, maxx and order (o). As well as defining the
    # function f to be the interpolation of the X, Y values based on the order of the function.
    # @param X The sequence of x values of the curve.
    # @param Y The sequence of y values of the curve.
    # @param i The order of the curve. 1 is linear and 2 is quadratic.
    def __init__(self, X, Y, i):
        if isAscending(X) == False:
            raise IndepVarNotAscending()
        if len(X) != len(Y):
            raise SeqSizeMismatch()
        if i < 1 or i > 2:
            raise InvalidInterpOrder()
        self.minx = X[0]
        self.maxx = X[len(X)-1]
        self.o = i
        self.f = interpolate.interp1d(X,Y,i)

    ## @brief Returns the minimum value in the sequence X.
    # @return Returns the minimum value in the sequence X.
    def minD(self):
        return self.minx

    ## @brief Returns the maximum value in the sequence X.
    # @return Returns the maximum value in the sequence X.
    def maxD(self):
        return self.maxx

    ## @brief Returns the order value of the curve.
    # @return Returns the order value of the curve.
    def order(self):
        return self.o

    ## @brief Interpolates a value x on the curve.
    # @param x The value x on the curve being interpolated.
    # @return Interpolates the value of the curve at a point x by calling the interpolation function f.
    def eval(self, x):
        if x < self.minx or x > self.maxx:
            raise OutOfDomain()
        return self.f(x)

    ## @brief Estimates the first order derivative at the point x.
    # @details Estimates the first order derivative at the point x by calling the interpolation
        function f and applying a first order derivative formula.
    # @param x The x value of the point whose derivative is being calculated.
    # @return The estimate first order derivative of the point x on the curve.
    def dfdx(self, x):
        if x < self.minx or x > self.maxx:
            raise OutOfDomain()
        return (self.f(x+CurveT.DX)-self.f(x))/CurveT.DX

    ## @brief Estimates the second order derivative at the point x.
    # @details Estimates the second order derivative at the point x by calling the interpolation
        function f and applying a second order derivative formula.
```

```
# @param x The x value of the point whose derivative is being calculated.
# @return The estimate second order derivative of the point x on the curve.
def d2fdx2(self, x):
    if x < self.minx or x > self.maxx:
        raise OutOfDomain()
    return (self.f(x+2*CurveT.DX)-2*self.f(x+CurveT.DX)+self.f(x))/(CurveT.DX*CurveT.DX)
```

F Code for Data.py

```
## @file Data.py
# @title Data
# @author Ninos Yomo
# @date 02/22/2018

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

## @brief An abstract object data that stores curves.
# @details An abstract object data that stores curves in a sequence. Along with the curves,
# another sequence stores reals inputted by the user that is associated with its respective curve in
# the other sequence. Up to 10 sequences can be stored.
class Data:

    ## MAX_SIZE the maximum number of curves that the Data object can store
    MAX_SIZE = 10

    ## @brief Initializes the Data abstract object
    # @details Initializes the Data abstract object by creating two sequences that will store the
        curves and associated real values.
    # @param S The sequence that will store the curve objects.
    # @param Z The sequence that will store the real values associated with a curve.
    @staticmethod
    def init():
        Data.S = []
        Data.Z = []

    ## @brief Adds a curve and real to their respective sequences.
    # @details Adds a curve s to the sequence of curves S and a real z to the sequence of reals z.
    # @param s The curve that will be added to sequence S.
    # @param z The real number that will be added to sequence Z.
    @staticmethod
    def add(s,z):
        if len(Data.S) == Data.MAX_SIZE:
            raise Full()
        if len(Data.Z) > 0 and z <= Data.Z[len(Data.Z)-1]:
            raise IndepVarNotAscending()
        Data.S.append(s)
        Data.Z.append(z)

    ## @brief Returns a curve that was searched for.
    # @details Given an index value i, returns the curve in sequence S at that index if it exisits.
    # @param i The index value of the sequence S that will be referenced.
    # @return The curve at the index i of the sequence S.
    @staticmethod
    def getC(i):
        if i < 0 or i > len(Data.S) or len(Data.S) == 0:
            raise InvalidIndex()
        return Data.S[i]

    ## @brief Estimates the eval(x) at some given z value by interpolating it with the curves
    # beside the z value.
    # @details Takes two consecutive curves in Data.S based on the given z value and evaluates
    # them at a point x, it then performs a linear interpolation using their z values and their
    # evaluated values at x.
    # @param x The point where the curves will be evaluated at.
    # @param z The point where the estimated eval(x) value will be interpolated.
    # @return The estimated eval(x) at z.
    @staticmethod
    def eval(x,z):
        if isInBounds(Data.Z,z) == False:
            raise OutOfDomain()
        return
            interpLin(Data.Z[index(Data.Z,z)],Data.S[index(Data.Z,z)].eval(x).take(0),Data.Z[index(Data.Z,z)+1],Data.S[inde

    ## @brief Creates a new curve by evaluating each curve in Data at some point.
    # @details Creates a new curve. First, the independent variable of the new curve is the sequence
    # of z values in Data. The dependent variable will be the curves evaluated at some x value.
    # @param x The point where each curve will be evaluated at.
    # @param i The order of the new curve.
    # @return The new curve.
    @staticmethod
    def slice(x,i):
        __Y__ = []
        for j in range(len(Data.Z)):
```

```
    __Y__.append(Data.S[j].eval(x).take(0))
print(__Y__)
print(Data.Z)
return CurveT(Data.Z,__Y__,i)
```

G Code for SeqServices.py

```
## @file SeqServices.py
# @title SeqServices
# @author Ninos Yomo
# @date 02/20/2018

## @brief Checks if a sequence is in ascending order.
# @details Takes a sequence X, and determines whether or not the sequence is in ascending order by
    checking if any element is less than its previous.
# @param X A sequence whose ascending property will be verified.
# @return Returns the boolean value of the ascending property of the sequence.
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 is bounded in a sequence.
# @details Checks if a value x is found to be in the sequence X by seeing if it is within the bound
    values, such that it x >= X[0] & x <= X[len(X)-1]
# @param X The sequence being examined.
# @param x The value being checked for in the sequence boundaries.
# @return Returns the boolean that says if the value is bounded in the sequence or not.
def isInBounds(X, x):
    if x < X[0] or x > X[len(X) -1]:
       return False
    return True

## @brief Estimates the value of y in some point (x,y) of a line.
# @details Estimates the value of y at point x by using point (x1,y1) which is before (x,y) and
    (x2,y2) which is after (x,y), for some line.
# @param x1 The x-value of the point before (x,y).
# @param y1 The y-value of the point before (x,y).
# @param x2 The x-value of the point after (x,y).
# @param y2 The y-value of the point after (x,y).
# @param x The x-value of the point (x,y).
# @return Returns the estimated value of y in the point (x,y).
def interpLin(x1, y1, x2, y2, x):
    return ((y2 - y1)/(x2 - x1))*(x - x1)

## @brief Estimates the value of y in some point (x,y) of a quadratic.
# @details Estimates the value of y at point x by using points (x0,y0) and (x1,y1) which are before
    (x,y) and (x2,y2) which is after (x,y), for some line.
# @param x0 The x-value of the point (x0,y0) before (x,y).
# @param y0 The y-value of the point (x0,y0) before (x,y).
# @param x1 The x-value of the point (x1,y1) before (x,y).
# @param y1 The y-value of the point (x1,y1) before (x,y).
# @param x2 The x-value of the point (x2,y2) after (x,y).
# @param y2 The y-value of the point (x2,y2) after (x,y).
# @param x The x-value of the point (x,y).
# @return Returns the estimated value of y in the point (x,y).
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 Searches index of a value in a sequence.
# @details Takes a value x and searches its index in the sequence X.
# @params X The sequence being searched on.
# @params x The value being searched in the sequence.
# @return Returns the index i of the value x in the sequence X. Or, if the function was called with a
    value not bounded, tells the user the value was not found.
def index(X, x):
    for i in range(len(X)):
        if X[0] > x:
            print("Element Not Found! " + str(x) + " is not bounded in the list!")
            break
        if i == len(X)-1 and X[i] >= x:
            return i
        elif i == len(X)-1 and X[i] < x:
            print("Element Not Found! " + str(x) + " is not bounded in the list!")
            break
        if X[i] <= x and X[i+1] > x:
            return i
```

H Code for Plot.py

```
## @file Plot.py
# @title Plot
# @author Ninos Yomo
# @date 02/21/2018

from CurveADT import *
import matplotlib.pyplot as plt

## @brief Plots two sequences.
# @details Plots two sequences with the first argument as the independent
# variable and the second argument as the dependent variable.
# @param X The independent sequence.
# @param Y The dependent sequence.
def PlotSeq(X,Y):
    if len(X) != len(Y):
        raise SeqSizeMismatch()
    plt.plot(X,Y)
    plt.show()

## @brief Plots a curve.
# @details Given a curve c, this function plots it while considering how
# many evenly spaced points n to plot.
# @param c The curve being plotted.
# @param n The number of equally spaced points to plot.
def PlotCurve(c,n):
    plotPointsX = []
    plotPointsY = []
    plotPointsY.append(c.f(c.minD()).take(0))
    spacing = (c.maxx - c.minx)/n
    plotPointsX.append(c.minD() + spacing)
    for i in range(1,n-1):
        plotPointsX.append(plotPointsX[i-1] + spacing)
        plotPointsY.append(c.f(plotPointsX[i]).take(0))
    plt.plot(plotPointsX,plotPointsY)
    plt.show()
```

I Code for Load.py

```
## @file Load.py
# @title Load
# @author Ninos Yomo
# @date 02/23/2018

from CurveADT import *
from Data import *

## @brief Populates the Data object with curves from an input file.
# @details Reads some input file called s and creates curves from the input.
# The curves are then added to the Data abstract object.
# @param s The string name of the input file.
def Load(s):

    Data.init()
    file = open(s,'r')

    zVals = []                                  #Dealing with input for curve number and order.
    orderVals = []
    zLine = file.readline()
    orderLine = file.readline()
    zVals = zLine.split(",")
    orderVals = orderLine.split(",")
    for i in range(len(zVals)):
        zVals[i] = zVals[i].strip()
        zVals[i] = zVals[i].strip('\n')
        zVals[i] = float(zVals[i])
        orderVals[i] = orderVals[i].strip()
        orderVals[i] = orderVals[i].strip('\n')
        orderVals[i] = int(orderVals[i])

    xVals = []                                  #Formatting input for xVals and yVals.
    yVals = []
    lines = []
    count = 0
    for line in file:
        lines.append(line.split(','))
        for i in range(len(lines[count])):
            lines[count][i] = lines[count][i].strip()
            lines[count][i] = lines[count][i].strip('\n')
            if lines[count][i] != "":
                lines[count][i] = float(lines[count][i])
        count += 1
    print(lines)

    for i in range(len(lines)):                 #Sorting input
        for j in range(0,len(lines[0])-1,2):
                xVals.append(lines[i][j])
                yVals.append(lines[i][j+1])

    xTemp = []                                  #Populating Data
    yTemp = []
    for i in range(0,len(zVals)):
        for j in range(i,len(xVals),len(zVals)):
            if xVals[j] == "":
                break
            xTemp.append(xVals[j])
            yTemp.append(yVals[j])
        curve = CurveT(xTemp,yTemp,orderVals[i])
        Data.add(curve,zVals[i])
        xTemp = []
        yTemp = []
    file.close()
```

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
```