Assignment 2 Report

Hamid Ghasemi ghasemih

March 3, 2018

1 Testing of the Original Program

Basically, the testing for CurveT starts with assigning a variable to CurveT class and then make x,y, min, max, o and it automatically makes f out of other objects. Then it uses those objects which have been created in CurveT for all the functions. For Data, we would do the exact same method for data objects although Data is abstract object, so each time we would use variable.data in order to test functions in Data module. And SeqService is not class; therefore, we dont need to assign a variable. We use each function directly in order to test all the methods from the SeqService module. For testing part based on using pytest out of 16 functions, 13 functions passed and 3 functions didnt pass. I wasn't able to make those 3 tests because of the time issue. But beside that all functions from curveT and SeqService are perfectly working while for Data module there are three functions that are working but I didnt make test cases in the first round of assignment. In order to be able to see if the test has passed or not, I made a test function for each method by using assert and run the pytest. If the function displays the expectational output, it will display passed. In overall the number I got from testing is 13 out of 16.

2 Results of Testing Partner's Code

According to my partner codes majority of the cases passed (10/16) by using pytest in the first time of trying; however, there are some issues that exist in my code and my partner code. For Data module I didn't read the specification precisely, so instead of making abstract object, I made abstract data type which is not following the specification though its not wrong. Therefore, When I wanted to test his codes for Data module I had to change his to get it to work. And in his code, he has made DX as a local variable however he has used self in order to use that variable for function dfdx. He used DX as a object data though its not. After fixing all of those, his code passed all the testcases except the ones I didnt make. So in overall 13/16 functions from his code passed by using my test runs.

3 Discussion of Test Results

3.1 Problems with Original Code

From this assignment I have learned that if software engineer doesn't understand the specification it can make a huge impact on the way he/she writes the code. This can affect the whole process, so he/she should ask from others to get a better understanding. Also learned what's the difference between data object and abstract data object and how they are different and why they are different. Plus using pytest can aid a lot and it's very simple to use. As I have said before, I made a mistake defining my data init which instead of making it to be data object, I created abstract data object. In addition to that, when I was constructing my code, I didn't pay attention to bounding which made problems when I was trying to test it but I eventually figured it out.

3.2 Problems with Partner's Code

For my partner the only problems that I found were defining interp from CurveT inside the class of CurveT while the specification has asked to make it local. This is not big of the deal however doesn't follow the requirement. And there is a function eq that has been created in the CurveT although there is not such a function that should be created in the module. That's not wrong but the way it's been written, the method could have been constructed inside of the other functions.

3.3 Problems with Assignment Specification

I didn't find any problem in the specification although there were some situation that specification didn't explain what happens in special cases, for instance index function in the Seqservice module. It checks if x is in the range of the sequence but what would happen if x is outside of the range of the sequence. This can affect the code since we are not making any condition for 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?

Since we cannot assume it is ascending or not, we have to make an exception
exception:= $((\exists(i|i \in [0..|X|-2] : X_{i+1} < X_i)) \Rightarrow$ IndepVarNotAscending)

2. How would you modify `CurveADT.py` to support cubic interpolation?
   Based on the specification in the curveT, there is a condition that has been defined for both interpLin and interpQuad functions. We can extend the if statement and make another condition that when for interp function o == 3 it returns cubic interpolation function. The actual cubic function can be made inside of the CurveT or Seqservice like interpLin and interpQuad functions.

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.
   For CurveADT modules, the exported access programs provide an interface which is 1- consistent 2- essential 3- minimal (each method will do one job) 4- it is not general, according to general the module cannot always be predicted how the module will be used but in CurveADT it is predictable how the module will be used. 5- it is opaque since it hides information from users. The local function interp is hidden from the client.

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.
   For Data modules, the exported access programs provide an interface which is 1- consistent 2- essential 3- minimal (each method will do one job) 4- it is not general and it is not opaque.

E Code for CurveADT.py

```
## @file    CurveADT.py
#  @author Hamid Ghasemi 400028420
#  @brief   Provides function and a class
#  @date    08/02/2018

from SeqServices import *
from Exceptions import *

## @brief interp is a local function
#  @details Gets all the inputs and determine y as interpLin if o = 1, interpQuad when
#  o = 2
#  @param X is the first input sequence
#  @param Y is the second input sequence
#  @param o is the order of the function that user inputs
#  @param v is the input value
#  @return interpLin, interpQuad or False

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)
    else:
        return False

## @brief Making class curveT
class CurveT():
    global MAX_ORDER, DX

    MAX_ORDER = 2
    DX = 0.001

    ## @brief CurveT constructor
    #  @details It finds the maximum, minimum, order of function, and
    #  at the end it makes function f by using interp function
    #  @param X is a sequence that user inputs
    #  @param Y is a sequence that user inputs
    #  @param i is value that user inputs
    def __init__(self, X, Y, i):
        if (isAscending(X) == False):
            raise IndepVarNotAscending(”It’s not ascended”)
        elif (len(X) != len(Y)) :
            raise SeqSizeMismatch(”X and Y dont have same length”)
        elif (i not in range(1,MAX_ORDER +1)):
            raise InvalidInterpOrder(”i does not follow the condition”)

        self.minx = X[0]
        self.maxx = X[-1]
        self.o    = i
        self.f = lambda v : interp(X,Y,self.o,v)

    ## @brief minD is a funtion
    #  @details It returns minimum value of sequence X that has been
    #  defined in the constructor
    #  @return minimum
    def minD(self):
        return self.minx

    ## @brief maxD is a funtion
    #  @details It returns maximum value of sequence X that has been
    #  defined in the constructor
    #  @return maximum
    def maxD(self):
        return self.maxx

    ## @brief order is a funtion
    #  @details It returns the order that user has inputed
    #  @return maximum
    def order(self):
        return self.o

    ## @brief eval is a funtion
    #  @details It returns function f based on x
    #  @param x is a value user inputs
    #  @return function f
```

```
def eval(self, x):
    if ( self.minx > x or x > self.maxx):
        raise OutOFDomain(”x is not in the range”)
    return self.f(x)

## @brief dfdx is a funtion
#  @details It returns a value by using the equation (f(x+DX)−f(x))/DX
#  @param x is a value user inputs
#  @return function f
def dfdx(self, x):
    if ( self.minx > x or x > self.maxx):
        raise OutOFDomain(”x is not in the range”)
    d = (self.f(x+DX)−self.f(x))/DX
    return d

## @brief d2fdx2 is a funtion
#  @details It returns a value by using the equation (f(x+2∗DX)−2∗f(x+DX)+f(x))/(DX∗∗2)
#  @param x is a value user inputs
#  @return function f
def d2fdx2(self, x):
    if ( self.minx > x or x > self.maxx ):
        raise OutOFDomain(”x is not in the range”)
    d = (self.f(x+2∗DX)−2∗self.f(x+DX)+self.f(x))/(DX∗∗2)
    return d
```

F Code for Data.py

```
## @file    Data.py
#  @author Hamid Ghasemi 400028420
#  @brief  Provides multiples functions and a class
#  @date   08/02/2018

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

## @brief Making class Data
class Data:
    global MAX_SIZE
    MAX_SIZE = 10

    ## @brief Data_init is a funtion we define our objects in the class
    #  @details It makes two empty sequences S and Z
    def init(self):

        self.S = [ ]
        self.Z = [ ]

    ## @brief Data_add is a funtion that add a value to sequence S and Z
    #  @details First checks if the sequence has reached the max size
    #  or not then checks if s and z are the bigger values than the last
    #  the last value of S and Z. If the condition meets then it adds the value
    #  s and z.
    #  @param s is a sequence
    #  @param z is a sequence
    def add(self, s, z):
        if ( len(self.S) == MAX_SIZE ):
            raise Full(”There is no space”)
        elif ( self.Z != []):
            if (z <= self.Z[len(self.Z)-1]):
                raise IndepVarNotAscending(”It's not ascended”)

        self.S = self.S + s
        self.Z = self.Z + z

    ## @brief Data_getC is a funtion
    #  @details It returns a value from the sequence by using an index i
    #  @param i is a value that uses as an index of sequence S and Z
    #  @return value from the sequence
    def getC(self, i):
        if ( i < 0 or i >= len(self.S)):
            raise InvalidIndex(”i is out of range”)

        return self.S[i]

    ## @brief Data_eval is a funtion
    #  @details It uses index and interpLin functions to return a value
    #  @param x is a input value
    #  @param z is a input value
    #  @return a value by using interpLin function
    def eval(self, x, z):
        if (isInBounds(self.Z, z) == False):
            raise OutOfDomain(”z is out of range”)

        j = index(self.Z,z)
        d = interpLin(self.Z[j], self.S[j].eval(x), self.Z[j+1], self.S[j+1].eval(x), z)
        return d

    ## @brief Data_slice is a funtion
    #  @details It checks if i is within index of sequence Z then uses x and i values
    #  and returns CurveT based on Z,Y, and i
    #  @param x is a input value
    #  @param i is a input value
    #  @return CurveT
    def slice(self,x,i):
        while(0 <= i <= len(self.Z)-1):
            Y = self.S[i].eval(x)
            return CurveT(self.Z, Y, i)
```

G Code for SeqServices.py

```
## @file    SeqServices.py
#  @author Hamid Ghasemi 400028420
#  @brief   Provides some functions
#  @date    08/02/2018


## @brief isAscending is a function that check if the array is ascending
#  @details Uses the input sequence that user inserts and check if the sequence
#  is ascending or not
#  @param X is the input sequence
#  @return Boolean value True or False
def isAscending(X):
    for i in range (0, len(X)-2):
        if X[i+1] < X[i]:
            return False

    return True


## @brief isInBounds is a function
#  @details Uses the input value and sequence that user inserts and check if the value
#  is within sequence.
#  @param X is the input sequence
#  @param x is input value
#  @return Boolean value True if value of x is within sequence or False
def isInBounds(X,x):
    if (X[0] <= x and x <= X[len(X)-1]):
        return True
    else:
        return False


## @brief interpLin is a function
#  @details Uses the input values that user inserts and find the interpolation linear
#  between all the values
#  @param x1 is input value
#  @param y1 is input value
#  @param x2 is input value
#  @param y2 is input value
#  @param x is input value
#  @return y that is made by x1,y1,x2,y2,x values
def interpLin(x1,y1,x2,y2,x):
    if (x2 - x1 == 0):
        print (”Error, denominator is zero”)
    else:
        y = ((y2-y1)/(x2-x1))*(x-x1) + y1
        return y
## @brief interpLin is a function
#  @details Uses the input values that user inserts and find the interpolation quadratic
#  by using all the values
#  @param x0 is input value
#  @param y0 is input value
#  @param x1 is input value
#  @param y1 is input value
#  @param x2 is input value
#  @param y2 is input value
#  @param x  is input value
#  @return y that is made by x1,y1,x2,y2,x values

def interpQuad(x0,y0,x1,y1,x2,y2,x):

    if (x2 - x0 == 0) or (x2 - x1 == 0):
        print (”Error, there is zero in denominator”)
    else :
        y = y1 + ((y2-y0)/(x2-x0))*(x-x1)+((y2-2*y1+y0)/(2*((x2-x1)**2))*(x-x1)**2)
        return y

def index(X,x):
    for i in range (0, len(X)-1):
        if (X[i] <= x and  x < X[i+1]):
            return i
```

H Code for Plot.py

```
## @file    plot.py
#  @author Hamid Ghasemi 400028420
#  @brief  Display graphs of the sequence
#  @date   08/02/2018

from CurveADT import *

from matplotlib import pyplot as plt

## @brief PlotSeq is a function that plots sequences
#  @details Gets all the inputs sequence and plots the diagram based
#  on sequence X and Y
#  @param X is the first input sequence
#  @param Y is the second input sequence
def PlotSeq(X, Y):
    if (len(X) != len(Y)):
        raise SeqSizeMismatch (”The length of X and Y are not equal”)


    plt.plot(X, Y, ’ro’)
    plt.xlabel(’x axiom’)
    plt.ylabel(’y axiom’)
    plt.show()

## @brief PlotCurve is a function that plots a curve
#  @details Gets all the inputs sequence and plots the diagram based
#  on sequence X and Y
#  @param c is sequence of CurveT that user inputs
#  @param n is value that user inputs
def PlotCurve(c, n):
    if (c.order()== 2):
        X = range(c.minD() + ((c.maxD()- c.minD())/n), c.maxD() - ((c.maxD()-c.minD())/n),
             (c.maxD()-c.minD())/n )
        Y = [c.eval(x) for x in X]
        PlotSeq(X, Y)
    elif (c.order()== 1):
        X = range(c.minD(), c.maxD() - ((c.maxD()-c.minD())/n), (c.maxD()-c.minD())/n )
        Y = [c.eval(x) for x in X]
        PlotSeq(X, Y)
```

I Code for Load.py

```
## @file    Load.py
#  @author Hamid Ghasemi 400028420
#  @brief  Read data from the file infile associated with the string s
#  @date   08/02/2018

import csv
from CurveADT import *
from Data import *

## @brief Initializes four empty sequences
#  @details reads a textfile with data of a polynomial and save x and y in return
def Load(s):
    k = Data()
    k.init()
    o = []
    x = []
    y = []
    with open(s) as csvfile:
            lines = csv.reader(csvfile, delimiter = ',')
            number = 0
            for row in lines:
                if number == 0:
                    for i in range(0, len(row), 1):
                        k.Z.append(float(row[i]))
                    m = len(k.Z)
                    x = [[0 for x in range(1)] for y in range(m)]
                    y = [[0 for x in range(1)] for y in range(m)]
                    number += 1
                elif number == 1:
                    for i in range(0, len(row), 1):
                        o.append(float(row[i]))
                    number += 1
                else:
                    data = 0
                    for i in range(0, 2*m - 1, 2):
                        a = row[i]
                        b = row[i + 1]
                        if a != '' or b != '':
                            x[data].append(float(a))
                            y[data].append(float(b))
                        data += 1
```

J Code for testAll.py

```
from CurveADT import *
from Data import *
from SeqServices import *


def test_isAscending():
    assert isAscending([1, 3 ,4 ,5]) == True
    assert isAscending([3, 3 ,4 ,5]) == False
    assert isAscending([3, 4 ,2 ,5]) == False

def test_isInBounds():
    assert isInBounds([1, 3 ,4 ,5],6) == False
    assert isInBounds([1, 3 ,4 ,5],2) == True
    assert isInBounds([1, 3 ,4 ,5],5) == True

def test_interpLin():
    assert interpLin(5,1,6,1,3) == 0
    assert interpLin(1,2,5,12,2) == 4.5
    assert interpLin(1,1,2,2,3) == 4

def test_interpQuad():
    assert interpLin(3,0,1,4,2,0,1) == 4
    assert interpLin(2,1,3,4,1,3,1) == 6


def test_index():
    assert index([1, 3 ,4 ,5],3) == 1
    assert index([1, 3 ,4 ,5],4) == 2
    assert index([1, 3 ,4 ,5, 6, 10, 16],12) == 5

############################

x = Data()
x.init()
x.S = [ 1, 3 , 4 , 5]
def test_init():
    assert x.S[0] == 1
    assert x.Z == []
    assert x.S[3] == 5

def test_add():
    x.add(1, 2)
    assert x.S == [ 1, 3 , 4 , 5, 1]
    assert x.Z == [2]
    assert x.S[3] == 5

def test_getC():
    assert x.getC(2)== 4
    assert x.getC(0)== 1

#def test_eval():

#def test_slice():

############################
X= [1, 2 ,3 ,10]
Y=[2, 4, 5, 7]
i= 2
x = CurveT(X, Y, i)

#def test_interp():


def test_minD():
    assert x.minD() == 1

def test_maxD():
    assert x.maxD() == 10

def test_order():
    assert x.order() == 2

def test_aval():
    assert x.eval(11) == 'OutOFDomain'

def test_dfdx():
    assert x.dfdx(21) == 'OutOFDomain'

def test_d2fdx2():
    assert x.d2fdx2(21) == 'OutOFDomain'
```

K Code for Partner’s CurveADT.py

```
## @file CurveADT.py
# @title CurveT
# @author Somar Aani
# @date 07/02/2018

from SeqServices import *
from Exceptions import *

## @brief Class representing a curve
class CurveT:

    MAX_ORDER = 2
    DX = 0.001

    ## @brief CurveT constructor
    # @details constructs the curve using the data provided
    # @param Real[]: X list of X values for the curve
    # @param Real[]: Y list of Y values for the curve
    # @param Integer: i order to be used in interpolation
    # @return new CurveT object
    def __init__(self, X, Y, i):
        if(not isAscending(X)):
            raise IndepVarNotAscending(”X must be ascending”)
        if(len(X) != len(Y)):
            raise SeqSizeMismatch(”X and Y must be same length!”)
        if(i != 1 and i != 2):
            raise InvalidInterpOrder(”i must be either t1 or 2”)

        #need to store to compare if equal
        self.X = X
        self.Y = Y

        self.minx = X[0]
        self.maxx = X[len(X) - 1]
        self.o = i
        self.f = lambda v: self.__interp__(X, Y, self.o, v)

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

    ## @brief Finds the smallest x value in data
    # @return Real: smallest x-value
    def minD(self):
        return self.minx

    ## @brief Finds the largest x value in data
    # @return Real: largest x-value
    def maxD(self):
        return self.maxx

    ## @brief Returns order of interpolation of function
    # @return Integer: order of interpolation
    def order(self):
        return self.o

    ## @brief Approximates curve at the given point
    # @details Uses linear (o = 1) or quadratic (o = 2) interpolation to approximate value of curve at
        x
    # @param x Real: value to be evaluated
    # @return Real: value of curve at x
    def eval(self, x):
        if(x > self.maxx or x < self.minx):
            raise OutOfDomain(”x out of curve range”)
        return self.f(x)

    ## @brief Approximates the first derivative of the curve
    # @details Uses forward divided difference to approximate the value at x for the first derivative
        of the curve
    # @param x Real: value to be evaluated
    # @return Real: value of derivate curve at x
    def dfdx(self, x):
        if(x > self.maxx or x < self.minx):
            raise OutOfDomain(”x out of curve range”)
        return (self.f(x + self.DX) − self.f(x))/self.DX

    ## @brief Approximates the second derivative of the curve
    # @details Uses forward divided difference to approximate the value at x for the second derivative
         of the curve
    # @param x Real: value to be evaluated
    # @return Real: value of second derivate curve at x
    def d2fdx2(self, x):
        if(x > self.maxx or x < self.minx):
            raise OutOfDomain(”x out of curve range”)
        return (self.f(x + 2*self.DX) − 2*self.f(x + self.DX) + self.f(x))/(self.DX ** 2)

    def __eq__(self, other):
        if isinstance(self, other.__class__):
            return self.o == other.o and self.X == other.X and self.Y == other.Y
        return false
```

L Code for Partner’s Data.py

```
## @file Data.py
# @title Data
# @author Somar Aani
# @date 07/02/2018

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

## @brief Class representing data of curves
class Data:

    MAX_SIZE = 10

    ## @brief Initializes data sequences
    def init():
        Data.S = []
        Data.Z = []

    ## @brief adds values to data
    # @param s CurveT: curve to be added
    # @param z Real: independant variable to be added
    def add(s,z):
        if(len(Data.S) == Data.MAX_SIZE):
            raise Full(”Data is full”)
        if(len(Data.Z) != 0 and z <= Data.Z[len(Data.Z)-1]):
            raise IndepVarNotAscending(”Value being added is not larger than last element”)

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

    ## @brief gets curve at index i
    # @param i Integer: Index of curve to be returned
    # @return CurveT: curve at index i
    def getC(i):
        if(i < 0 or i >= len(Data.S)):
            raise InvalidIndex(”Index is invalid”)
        return Data.S[i]

    ## @brief uses linear interpolation on data
    # @param x: approximates curves at x
    # @param z: approximates data at z
    # @return linear approximation of the data at z
    def eval(x, z):
        if(not isInBounds(Data.Z, z)):
            raise OutOfDomain(”z 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 slices the data to return a new Curve
    # @param x: approximates curves at x
    # @param i: order of interpolation for output curve
    # @return CurveT: independant variables are equivalent to data, and dependant are approximation at
        x
    def slice(x, i):
        Y = list(map(lambda j : (Data.S[j].eval(x)), range(len(Data.Z))))
        return CurveT(Data.Z, Y, i)
```

M Code for Partner's SeqServices.py

```
## @file SeqServices.py
# @title Sequence Services
# @brief library containing some sequence operations
# @author Somar Aani
# @date 07/02/2018
#

## @brief Finds out whether a list is strictly increasing
# @param X Real[]: list to check
# @return Bool: if the list is ascending
def isAscending(X):
    for i in range(len(X) - 1):
        if(X[i+1] < X[i]):
            return False
    return True

## @brief Finds out whether a variable is in bounds of a list
# @param X Real[]: list to check
# @param x Real: value to check
# @return Bool: if x is in range of X
def isInBounds(X, x):
    return X[0] <= x and x <= X[len(X)-1]

## @brief Linear interpolation using 2 points
# @param x1 Real: value of x1
# @param y1 Real: value of y1
# @param x2 Real: value of x2
# @param y2 Real: value of y2
# @param x Real: value which curve is approximated at
# @return Real: approximation at x using x1,y1,x2,y2
def interpLin(x1, y1, x2, y2, x):
    return (y2 - y1)/(x2 - x1)*(x - x1) + y1

## @brief Quadratic interpolation using 3 points
# @param x1 Real: value of x0
# @param y1 Real: value of y0
# @param x1 Real: value of x1
# @param y1 Real: value of y1
# @param x2 Real: value of x2
# @param y2 Real: value of y2
# @param x Real: value which curve is approximated at
# @return Real: approximation at x using x0,y0,x1,y1,x2,y2
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 position of a value in a list
# @param X Real[]: list to check
# @param x Real: the value of the position to be found
# @return Integer: location (i) of integer such that X(i) <= x <= X(i+1)
def index(X, x):
    for i in range(0, len(X) - 1):
        if(X[i+1] > x):
            return i
```

N 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
```