Assignment 2 Report

Andy Hameed — 400073469

March 3, 2018

The ambiguity of the specifications in assignment 1 caused for multiple interpretations of module access routines and services. In assignment 2, students were given a formal specification in the format of an MIS. This clarified the functional purpose of each access routine and provided an overall better description of the software. However, the implementation was still left in the hands of the programmer and gave flexibility for different solutions to a given specification.

1 Testing of the Original Program

26 tests were used to cover every method in the listed modules, CurveADT, SeqServices and Data. Simple access methods were tested by creating an object and asserting the inputted parameters reflected the get methods that were used in the modules. For more complex methods, ones like linear and quadratic interpolation, I used an online interpolation tool and compared that value to the one obtained by each method, using a small tolerance for float comparison inaccuracy in python.

2 Results of Testing Partner's Code

When testing my partner's code, 25/26 tests passed with only 1 failure due to syntax formatting for exceptions. The full exception specified in the MIS for when Data storage is full, was referenced as FULL by my partner which did not correspond to my reference of the method as "Full" in the exceptions, a small and harmless derivation.

3 Discussion of Test Results

As expected, the formal specifications eliminated misinterpretations of methods and demonstrated the congruency between two versions of a software application despite in-

dividual implementation differences.

3.1 Problems with Original Code

Original code covered could have been more efficient in terms of implementation to enhance performance. The load function, for example, read the input file several times instead of reading it once and storing its values in data types.

3.2 Problems with Partner's Code

Besides naming differences that caused one test, my partner's code passed all but the aforementioned test and seemed correct altogether.

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?

   The new specification for isInBounds can show that there exists some number less than x and some number greater than x:

   $\exists\,(i, j | X_i \leq x \leq X_j)$

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

   I would began by changing the MAX_ORDER variable invariant to equal 3 in order to handle cubic interpolation. I would then edit the interp local function to include cubic interpolation which would be implemented in the SeqServices module along with interpLin and interpQuad.

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 CurveADT module interface could be improved with better specifications for edge cases. The complication is abstracted for the purposes of the assignment but would be useful for completing the specifications for the interface.

   The interface is not minimal since the eval method implements linear and quadratic interpolation which could be classified as two separate services. It also not essential

since we know that the second derivative can be evaluated by taking the first derivative of a given input twice: $dfdx(dfdx(x)) \equiv d2fdx2(x)$. The interface is opaque since it incorporates information hiding and allows the programmer to abstract details such as how a given input is evaluated or it's derivative is calculated and so on. It is also consistent since it uses exception handling and consistent naming conventions such as x to specify some x-value.

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.

   To complement the add method in the Data module, adding a delete method would improve generality of the interface and allow users to use the data in a multitude of other ways. Since there is no size method in the Data interface, exception handling cannot be implemented in advance and must be addressed during a call to the add method.

   The Data interface is not minimal for the same reason being that eval is a method that holds two services, linear and quadratic interpolation. However, the interface is essential since no method could be obtained through a combination of the other methods. The interface is opaque since it applies information hiding, allowing the user to use methods without worrying about how they're implemented. Lastly, the interface is consistent since it uses naming conventions such as i to indicate index, x for x value and other variables in a consistent manner along with proper exception handling for select methods.

E Code for CurveADT.py

```
## @file CurveADT.py
#  @author Andy Hameed
#  @brief implements an abstract data type for a curve
#  @date 21/02/2018

from SeqServices import *
from Exceptions import *

## @brief An Abstract Data type representing a curve
class CurveT:

    ## @brief CurveT constructor
    #  @details Initializes a CurveT object with a set of X and Y coordinates
    #  @param X The set of x values for the curve
    #  @param Y The set of y values for the curve
    #  @param i Order of polynomials representing curves
    def __init__(self,X, Y, i):
        self.MAX_ORDER = 2
        self.DX = 1e-3

        if not (isAscending(X)):
            raise IndepVarNotAscending("values must be in ascending order!")
        elif (len(X) != len(Y)):
            raise SeqSizeMismatch("Sequence size X does not match Y")
        elif (i < 1 or i > self.MAX_ORDER):
            raise InvalidInterpOrder("Order of interpolation should be 1 or 2")
        else:
            self.minx = X[0]
            self.maxx = X[-1]
            self.o = i
            self.f = lambda v : interp(X,Y,self.o,v)

    ## @brief Accessor method for minimum x value in X
    # @return minimum value x
    def minD(self):
        return self.minx

    ## @brief Accessor method for maximum x value in X
    # @return maximum value x
    def maxD(self):
        return self.maxx

    ## @brief Accessor method for the order of a curve
    # @return order of curve
    def order(self):
        return self.o

    ## @brief eval method performs interpolation on a value x
    # @return interpolated y value
    def eval(self, x):
        if not (self.minD() <= x and x <= self.maxD()):
            raise OutOfDomain("No longer in the domain")
        else:
            return self.f(x)

    ## @brief dfdx method finds the first direvative at a given point x
    # @param x is the point where derivative is taken
    # @return First direvative of curve at x
    def dfdx(self, x):
        if not (self.minD() <= x and x <= self.maxD()):
            raise OutOfDomain("No longer in the domain")
        else:
            First_Dir = (self.f(x + self.DX) - self.f(x))/self.DX
            return First_Dir

    ## @brief d2fdx2 method finds the second direvative at a given point x
    # @param x is the point where second derivative is taken
    # @return Second direvative of curve at x
    def d2fdx2(self, x):
        if not (self.minD() <= x and x <= self.maxD()):
            raise OutOfDomain("No longer in the domain")
        else:
            Sec_Dir = (self.f(x + 2*self.DX) - (2*self.f(x + self.DX)) + self.f(x))/((self.DX)**2)
            return Sec_Dir

## @brief A function that performs linear or quadratic interpolations depending
```

```
# @details on input
# @param X set of x-values
# @param Y set of y-values
# @param o order of interpolation
# @param v value being interpolated for
# @return intepolated value y at the given x-value v
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)
```

F Code for Data.py

```
## @file Data.py
#  @author Andy Hameed
#  @brief Implements Data abstract object
#  @date 21/02/2018

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

class Data:

    # Maximum size of the data
    MAX_SIZE = 10

    # Empty Sequences
    S = []
    Z = []

    ## @brief Data initializer
    @staticmethod
    def init():
        Data.S = []
        Data.Z = []

    ## @brief Add Method is used to append a new curve and value for a curve
    # @param s is the new curve being added
    # @param z is the new curve value being added
    @staticmethod
    def add(s,z):
        if (len(Data.S) == Data.MAX_SIZE):
            raise Full("Data is full")
        elif(len(Data.Z) > 0 and z <= Data.Z[-1]):
            raise IndepVarNotAscending("values must be in ascending order!")
        else:
            Data.S.append(s)
            Data.Z.append(z)

    ## @brief getC Method is used to find the curve at a certain index
    # @param i is the specified index of the curve
    @staticmethod
    def getC(i):
        if not (i >= 0 and i < len(Data.S)):
            raise InvalidIndex("Index is not valid")
        else:
            return Data.S[i]

    ## @brief eval Method is used to interpolate a curve given its value and an x value for
         interpolation
    # @param x is the x value of interpolation
    # @param z is the value of the curve being interpolated
    @staticmethod
    def eval(x,z):
        j = index(Data.Z,z)
        if not (isInBounds(Data.Z,z)):
            raise OutOfDomain("No longer in the domain")
        else:
            return interpLin(Data.Z[j],Data.S[j].eval(x),Data.Z[j+1],Data.S[j+1].eval(x),z)

    ## @brief slice method creates a curve of order i with y values
    # @details corresponding to the interpolation at a given x value
    # @param x is the value of interpolation on each curve
    # @param i is the order of the curve being created
    @staticmethod
    def slice(x,i):
        #Evaluating each curve in S at given x value before creating new curve
        Y = list(map(lambda s: s.eval(x),Data.S))
        return CurveT(Data.Z,Y,i)
```

G Code for SeqServices.py

```
## @file SeqServices.py
#  @author Andy Hameed
#  @brief implements necessary methods for that service defined sequences
#  @date 21/02/2018


def isAscending(X):
    for i in range(len(X) - 1):
        if (X[i+1] < X[i]):
            return False
    return True

def isInBounds(X,x):
    if (X[0] <= x <= X[len(X)-1]):
        return True
    return False

def interpLin(x1, y1, x2, y2, x):
    return ((y2-y1)/(x2-x1))*(x - x1) + y1

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

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

H Code for Plot.py

```
## @file Plot.py
#  @author Andy Hameed
#  @brief Implements curve plotting
#  @date 21/02/2018

import CurveADT
import Exceptions
import numpy as np
from graphics import *
import matplotlib.pyplot as plt

# @brief Plotseq creates an x-y plot graph using the inputted sets of values
# @param X is the set of x-values
# @param Y is the set of y-values
def PlotSeq(X, Y):
    if (len(X) != len(Y)):
        raise SeqSizeMismatch("Sequence size does not match")
    else:
        plt.plot(X, Y)
        plt.show()

# @brief PlotCurve plots a curve c using n data points
# @param c is the curve to be plotted
# @param n is the number of data points or intervals used for graphing
def PlotCurve(c,n):
    X = np.linspace(c.minD(),c.maxD(),n,endpoint = False)
    if (c.order() == 2):
        X = X[1:]
    Y = [c.eval(x) for x in X]
    PlotSeq(X,Y)
```

I Code for Load.py

```
## @file Load.py
#  @author Andy Hameed
#  @brief Library for loading curves into the abstract object Data
#  @date 12/02/2018

from CurveADT import *
from Data import *


#What are environment variables?


## @brief Load is a method that initializes the data module to value
# @details found in the input file s
# @param s input file for reading curve values and x-y data points
def Load(s):
    Data.init()
    infile = open(s,"r")
    j = 0

    # Reading the values for each curve z1 ...Zn
    # and the order of interpolation o1 .. on
    temp = infile.readline().strip("\n").split(",")
    CurveValue = [float(x) for x in temp]
    temp = infile.readline().strip("\n").split(",")
    Order = [float(x) for x in temp]

    #Setting reset cursor position to third line of input file
    last_pos = infile.tell()

    #collecting each pair of x-y values for each curve by
    #looping through file (rereading)
    for point in range(0,len(CurveValue)*2,2):
        infile.seek(last_pos)
        temp = []
        for line in infile:
            XY_pair = (line.strip("\n").split(","))[point:point+2]
            if (XY_pair[0] == ''):
                continue
            XY_coord = [float(x) for x in XY_pair]
            temp += XY_coord
        X = temp[::2]
        Y = temp[1::2]
        #Appending a new curve based on collected x-y values
        Data.add(CurveT(X,Y,Order[j]), CurveValue[j])
        j+=1


    infile.close()
```

J Code for Partner's CurveADT.py

```
## @file CurveADT.py
#  @author Baikai, Wang
#  @date 20 Feb 2018
from SeqServices import *
from Exceptions import *
class CurveT:
    # maximum order of curve is 2
    MAX_ORDER = 2

    DX = 1*10**(-3)

    ## @brief CurveT constructor
    #  @details Initializes a CurveT object using a list X and list Y
    #  @param X a list named X
    #  @param p2 a list named Y
    def __init__(self, X, Y, i):
        if (isAscending(X) == False):
            raise IndepVarNotAscending("The list is not ascending!")
        elif (len(X) != len(Y)):
            raise SeqSizeMismatch("The lengths of two lists are not equal!")
        s = []
        for j in range(1, CurveT.MAX_ORDER+1):
            s.append(j)
        if (i not in s):
            raise InvalidInterpOrder("The import order is out of the max order range!")
        self.minx = X[0]
        self.maxx = X[len(X)-1]
        self.o = i
        self.f = (lambda v: __interp__(X, Y, self.o, v))

    ## @brief Gets minimum x point of the curve
    #  @return The minimum point of curve
    def minD(self):
        return self.minx

    ## @brief Gets maximum x point of the curve
    #  @return The maximum point of curve
    def maxD(self):
        return self.maxx

    ## @brief Gets order of the curve
    #  @return The order of curve
    def order(self):
        return self.o

    ## @brief Evaluate a import x in the curve
    #  @details Finds evaluate value for an import x by using the local function
    #  @para x import a x-coordinate
    #  @return The evaluate value of an import x
    def eval(self, x):
        if self.minx > x or x > self.maxx:
            raise OutOfDomain("The variable x out of the function's domain!")
        return self.f(x)

    ## @brief Gets the first derivative of an import x
    #  @details Finds the first derivative of an x by using the first derivative formula
    #  @para x import a x-coordinate
    #  @return The first derivative variable of a x
    def dfdx(self, x):
        if self.minx > x or x > self.maxx:
            raise OutOfDomain("The variable x out of the function's domain!")
        return ((self.f(x+CurveT.DX) - self.f(x))/CurveT.DX)

    ## @brief Gets the second derivative of an import x
    #  @details Finds the second derivative of an x by using the second derivative formula
    #  @para x import a x-coordinate
    #  @return The second derivative variable of a x
    def d2fdx2(self, x):
        if self.minx > x or x > self.maxx:
            raise OutOfDomain("The variable x out of the function's domain!")
        return ((self.f(x+2*CurveT.DX) - 2*self.f(x+CurveT.DX) + self.f(x))/(CurveT.DX**2))


## @brief Gets the interpolate value of a specific import v
#  @details Find the interpolate value of a x if it's a linear function or quadratic function,
#           using the interpolate function defined in SeqServices class.
```

```
#  @para X the x-coordinates of a Curve
#  @para Y the represented y values of each x
#  @para o the order of the curve
#  @para v an import variable for this curve
#  @return an interpolate value of the v
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))
```

K Code for Partner's Data.py

```
## @file Data.py
#  @author Baikai, Wang
#  @date 20 Feb 2018

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

class Data:
    # maximum size of sequence is 10
    MAX_SIZE = 10

    #empty sequence
    S = []
    Z = []

    ## @brief Data initializer
    @staticmethod
    def init():
        Data.S = []
        Data.Z = []

    ## @brief insert element to sequences
    @staticmethod
    def add(s, z):
        if (len(Data.S) == Data.MAX_SIZE):
            raise FULL("Maximum size exceeded")

        if (len(Data.Z) != 0):
            if (z <= Data.Z[len(Data.Z)-1]):
                raise IndepVarNotAscending("The list is not ascending!")
        Data.S.append(s)
        Data.Z.append(z)

    ## @brief Gets a curve from the list of Data.S
    @staticmethod
    def getC(i):
        if (i < 0 or i >= len(Data.S)):
            raise InvalidIndex("The variable i is not the correct index")
        return Data.S[i]

    ## @brief Calculate an import value x with a given index position z
    #  @details To evaluate an import x, we need to find the index position
    #           in Data.Z with the import z. Then, using the linear interpolate
    #           function  calculate the variable.
    #  @para x  the variable needs to evaluate
    #  @para z  the import z is to find the postion on Data.Z where x needs to calculate.
    #  @return  the output of the x.
    @staticmethod
    def eval(x, z):
        if (isInBounds(Data.Z, z) == False):
            raise OutOfDomain("The variable x out of the function's 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 Gets a curve from the list of Data.S
    @staticmethod
    def slice(x, i):
        Y = []
        for j in range(len(Data.S)):
            Y.append(Data.S[j].eval(x))
        return CurveT(Data.Z, Y, i)
```

L Code for Partner's SeqServices.py

```
## @file SeqServices.py
#  @author Baikai, Wang
#  @date 20 Feb 2018

## @brief Return a boolean value for testing a list is or not ascending
#  @para  X  a list needs to test
#  @return if the list X is ascending return true, else return false.
def isAscending(X):
        for i in range(len(X)-1):
            if  X[i+1] < X[i]:
                return False
        return True

## @brief Return a boolean value for testing an import is or not in bounds
#  @para  X  a list
#  @para  x  a import x for testing in bounds
#  @Assuming isAscending is True
#  @return if the list x is in bounds return true, else return false.
def isInBounds(X, x):
        if (X[0] <= x) and (x <= X[len(X)-1]):
            return True
        return False

## @brief Calculate a linear interpolate variable
#  @para  x1  an import x1
#  @para  y1  the correspending value of the import x1
#  @para  x2  an import x2
#  @para  y2  the correspending value of the import x2
#  @Assuming isAscending is True
#  @return the variable calculated by the interpolate linear function
def interpLin(x1, y1, x2, y2, x):
        return ((((y2-y1)/(x2-x1))*(x-x1))+y1)


## @brief Calculate a linear interpolate variable
#  @para  x1  an import x1
#  @para  y1  the correspending value of the import x1
#  @para  x2  an import x2
#  @para  y2  the correspending value of the import x2
#  @para  x3  an import x3
#  @para  y3  the correspending value of the import x3
#  @Assuming isAscending is True
#  @return the variable calculated by the interpolate quadratic function
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 Calculate a linear interpolate variable
#  @para  X  an import list
#  @para  x  the import x
#  @Assuming isInBounds is True
#  @return the variable calculated by the interpolate quadratic function
def index(X, x):
        for i in range(len(X)-1):
            if X[i] <= x and x <= X[i+1]:
                return i
```

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
```