Assignment 1 Report

Ninos Yomo — yomon

January 30, 2018

Introductory blurb.

1 Testing of the Original Program

For my approach to testing, I decided that I would give multiple test inputs for each method contained in my modules. The first test input would be a straight forward acceptable input that the program is expected to work with. My second round of input would usually be an odd choice for an input where, the program wouldn't nessecarly crash, but who's output could be unpredictable. Finally, the last input would be one that is expected to break and crash the program, just to see if the program could at least survive the input and pass some sort of message. After completely testing my modules, I found that it passed the basic requirements expected for each method, except for quadval which was found to be a little to unaccurate. It was expected to calculate the value within a 10 percent margin of error but failed. However, the criterion for passing was subjectively defined and can not be used to definitvely come to a conculsion on the correctness of the implementation.

2 Results of Testing Partner's Code

The program failed at the very first test. It was able to create the object from the SeqT class however, I was unable to print the content of the internal sequence since we did not name the state variable the same due to the ambiguity of the program specification. When manually changing the testing software to account for the different naming convention, the program managed to pass all of the basic tests but failed once more at the odd inputs since it was programed to handle it differently from mine, again this was due to the lack of clarity in the specification. Once more, it failed when needing to reference another state variable since, again, the specification did not explicity mention what to put leading to different state variable names and thus causing the errors.

3 Discussion of Test Results

3.1 Problems with Original Code

With my original code, the only problem seemed ro be that quadVal was a little too unaccurate for my subjectively defined test of + or - 10 percent as the error margin. My program was also not robust for bad inputs.

3.2 Problems with Partner's Code

The program was to robust, since the test file was not accounting for robust programs, it failed in some bad input tests. Aside from that, the state variable names were all different leading to the test file not being able to originally work when referencing its state variables.

3.3 Problems with Assignment Specification

The specification was just too vauge, it did not explicitly state important details that were required to know so that the program and test files would run properly for all students' codes. For example, the state variables were not explicitly named, this lead to students naming whatever they wanted and when we tested each other, we were very likely to get errors due to the different names, in my case, I called my sequence "sequence" and my partner names his "data". Another problem was that the specification did not define if the program needed to be robust or not. Because of this, some students took the initiative to make theirs robust, my partner did however, my test file did not account for this and gave fails to certain tests where he handled the input differently. Also, for quadVal, since it relied on three points, it failed to mention precisely what points to use, which lead to assumptions.

4 Answers

1. For each of the methods in each of the classes, please classify it as a constructor, accessor or mutator.

   Table on next page.

2. What are the advantages and disadvantages of using an external library like `numpy`?

   An advantage of using external libraries is that it can minimize the amount of code in your module, since external libraries simplify what you have to write, there will be less clutter in your code which makes it easier to read. Another advantage is

Method	Type
SeqT	Constructor
add	Mutator
rm	Mutator
set	Mutator
get	Accessor
size	Accessor
indexInSeq	Accessor
CurveT	Constructor
linval	Accessor
quadVal	Accessor
npolyVal	Accessor

that it enables you to do things with your program that you might not have been able to do on your own. This will lead to more productivity for the user, since they can focus on other aspects of their code. A disadvantage in using external libraries is that the documentation for it may not be very good. This makes it very hard for programmers to learn how to use it in their program, making it useless to them in some cases. Another disadvantage is that it can create a lot of overhead of the code being used is not optimized for preformance and memory efficency, if the programmer needs their program to preform optimally then the external library will be hurting them more then helping.

3. The `SeqT` class overlaps with the functionality provided by Python's in-built list type. What are the differences between `SeqT` and Python's list type? What benefits does Python's list type provide over the `SeqT` class?

   One difference is the way of creating a list. For the SeqT, an object from that class must be explicitly created whereas for the built in type, simply naming a list followed by equating it to "[]" will do. Although they preform essentially the same methods, the names given to them and the way they are implemented are different. To call the sequence in SeqT, the state variable within the object has to be explicitly stated while for the built in type only the list name has to be called. The advantage of using the built in type is because it is easier to work with and it is much more portable then the SeqT type as well as being more optimal in preformance and memory efficency.

4. What complications would be added to your code if the assumption that $x_i < x_{i+1}$ no longer applied?

Then implementation of multiple methods would have to change since they rely on the fact that the sequence is initially in order, because of this they will no longer work. What could be done to fix this is that in the constructor, it sorts the data so that the sequence is in order, with that implementation the methods would not have to be changed.

5. Will `linVal(x)` equal `npolyVal(n, x)` for the same `x` value? Why or why not?

   No, because npolyVal is a very accurate estimation that requires the input of the rank of the polynomial whereas linVal is a simple estimator. They also use two different methods of estimation, linVal uses interpolation and npolyVal uses regression, this will make it even more unlikely that they will have the same value. Finally, the values being used are real numbers, this leads to decimal numbers which makes it even more unlikely that they match up perfectly.

E Code for SeqADT.py

```
## @file SeqADT.py
# @title SeqADT
# @author Ninos Yomo
# @date 01/13/2018

## @brief This class represents a sequence.
# @details This class represents a sequence using a list while allowing operations that
     add, remove and get elements along with other functions.
class SeqT:

    ## @brief Initializes object of class SeqT
    # @details Initializes object of class SeqT with an empty list that the other class
         methods use.
    def __init__(self):
        self.sequence = []

    ## @brief Adds element to the sequence.
    # @details Adds an element v to a specific index of the sequence i.
    # @param i An integer, the specified index to add the element into the sequence.
    # @param v A real number, the value to be stored in the sequence.
    def add(self, i, v):
        self.sequence.insert(i, v)

    ## @brief Removes element in sequence.
    # @details Removes an element in the sequence with index specified by i.
    # @param i An integer, the specified index to remove the element in the sequence.
    def rm(self, i):
        del self.sequence[i]

    ## @brief Changes element in sequence.
    # @details Changes an element at a specific index i to become the value v.
    # @param i An integer, the specified index to change the element in the sequence.
    # @param v A real number, the value to be stored in the sequence.
    def set(self, i, v):
        self.sequence[i] = v

    ## @brief Returns an element from the sequence.
    # @details Returns a specific element at index i from the sequence.
    # @param i An integer, the specified index to get the element from the sequence.
    # @return Returns the value of the element of index i from the sequence.
    def get(self, i):
        return self.sequence[i]

    ## @brief Returns length of sequence.
    # @details Returns the number of elements that are in the sequence.
    # @return The number of elements in the sequence.
    def size(self):
        return len(self.sequence)

    ## @brief Finds index of element in sequence.
    # @details Finds the specific index of element v in the sequence, where s.get(i) <= v
         <= s.get(i+1). This is done by truncating the value v. Once truncated, a loop
         iterates over the list and compares the value of each element and checks if it
         matches with v. Once matched, the index is returned.
    # @param v A real number, the value to be searched in the sequence.
    # @return The index of the element v in the sequence.
    def indexInSeq(self, v):
        v = int(v)
        for i in range(0,(len(self.sequence) + 1)):
            if self.sequence[i] == v:
                return i
```

F Code for CurveADT.py

```
## @file CurveADT.py
# @title CurveADT
# @author Ninos Yomo
# @date 01/13/2018

import SeqADT
import numpy

## @brief Holds x and y values in sequence from input file.
# @details This class has two sequences from SeqADT.py that will store data from an input
    file. Specifically, the input file contains data about some function, once the
    function is stored, interpolation can be conducted on it as well as regression.
class CurveT:

    ## @brief Constructor for the CurveT class.
    # @details Constructs two sequences, the sequences will store data from an input file
        s. The input file will contain two columns with data entry seperated by commas.
        The first column represents x values and the other y values. These will be
        stored in their respective sequences.
    # @param s A string that holds the name of the input file used.
    def __init__(self, s):

        ## xVals is a field of the object created by class CurveT which is a sequence
            used to store x points from input file.
        self.xVals = SeqADT.SeqT()
        ## yVals is a field of the object created by class CurveT which is a sequence
            used to store y points from input file.
        self.yVals = SeqADT.SeqT()

        ## file is a field opens the input file s and is used to extract the data from it.
        file = open(s,"r")

        for i in file:
            ## A field that is used to hold each line of data and split it two
                corresponding sequences as the loop iterates through the file.
            lineTemp = i.split(",")
            self.xVals.add(self.xVals.size(), int(lineTemp[0]))
            self.yVals.add(self.yVals.size(), int(lineTemp[1]))

        file.close()

    ## @brief Interpolates value x in function using a linear model.
    # @details A value x is to be interpolated in the function that is stored in two
        sequences. An interpolation formula with a linear model is used and the
        calculated y value is returned.
    # @param x The x value of the function who's y value is to be interpolated.
    # @return The calculated y value.
    def linVal(self, x):

        for i in range(self.xVals.size()):
            if self.xVals.get(i) > x:
                return (((self.yVals.get(i) - self.yVals.get(i-1))/(self.xVals.get(i) -
                    self.xVals.get(i-1)))*(x - self.xVals.get(i-1)) +
                    self.yVals.get(i-1))

    ## @brief Interpolates value x in function using a quadratic model.
    # @details A value x is to be interpolated in the function that is stored in two
        sequences. An interpolation formula with a quadratic model is used and the
        calculated y value is returned.
    # @param x The x value of the function who's y value is to be interpolated.
    # @return The calculated y value.
    def quadVal(self, x):

        for i in range(self.xVals.size()):
            if self.xVals.get(i) > x:
                return (self.yVals.get(i-1) + ((self.yVals.get(i) -
                    self.yVals.get(0))/(self.xVals.get(i) - self.xVals.get(0)))*(x -
                    self.xVals.get(i-1)) + ((self.yVals.get(i) - 2*self.yVals.get(i-1) +
                    self.yVals.get(0))/(2*(self.xVals.get(i)-self.xVals.get(i-1)))**2)*((x
                    - self.xVals.get(i-1))**2))

    ## @brief A function that uses regression to find a y value given an x.
    # @details Uses polyfit from package numpy to regress a function, uses the highest
        order n and at x value x. This is then mapped using poly1d and the y value is
        calculated and returned.
```

```
# @param n The highest order of the function, using a very high number will result in
    errors.
# @param x The x value whos y value is to be returned.
# @return The calculated y value.
def npolyVal(self, n, x):

    ## The bestfit to the function which is found using numpy.polyfit(x,y,n) which is
        passed our x, y sequences and the rank n.
    bestfit = numpy.polyfit(self.xVals.sequence, self.yVals.sequence, n)
    ## The estimated function, values at x can now be calculated.
    f = numpy.poly1d(bestfit)
    print(f(x))
```

G Code for testSeqs.py

```
import SeqADT
import CurveADT

print("Testing file SeqADT")

    #SeqADT object creation test
print("Creating object 'test' from class 'SeqT'...")
test = SeqADT.SeqT()
print("Expecting empty sequence to print:")
if len(test.sequence) != 0:
    print("FAIL!")
else:
    print(test.sequence)
    print("PASS")

print("")

print("Testing method 'add' from class 'SeqT'")
                                                                    #Testing add method
    from SeqT
print("Using proper argument (adding '0' at index 0)...")
test.add(0,0)
print("Expecting '[0]' to print:")
if len(test.sequence) != 1:
    print("FAIL")
else:
    print(test.sequence)
    print("PASS")

print("Using inproper argument (adding '-1' at index -1)...")
test.add(-1,-1)
print("Expecting '[-1, 0]' to print:")
if test.sequence[0] != -1:
    print("FAIL")
else:
    print(test.sequence)
    print("PASS")

print("Using proper argument (adding '10' in the middle of the sequence using index '1')")
test.add(1,10)
print("Expecting '[-1, 10, 0]' to print:")
if test.sequence[1] != 10:
    print("FAIL")
else:
    print(test.sequence)
    print("PASS")

print("")

print("Testing method 'rm' from class 'SeqT'")
                                                                    #Testing rm method
    from SeqT
print("Using proper argument (removing '-1' at index 0)...")
test.rm(0)
print("Expecting '[10, 0]' to print:")
if test.sequence[0] != 10:
    print("FAIL")
else:
    print(test.sequence)
    print("PASS")

print("Using inproper argument (removing non-existent element at index 100)...")
print("Error expected (since index out of bounds):")
try:
    test.rm(100)
    print("FAIL, should have caught error!")
except:
    print("Error caught! Sequence unchanged.")
    print("PASS")

print("")

print("Testing method 'set' from class 'SeqT' using sequence '[1,2,3]'")
                                            #Testing set method from SeqT
test.rm(0)
test.rm(0)
```

```
test.add(0,3)
test.add(0,2)
test.add(0,1)
print("Using proper argument (setting value at 0 to go from 1 to 0)...")
test.set(0,0)
print("Expecting '[0,2,3]' to print:")
if test.sequence[0] != 0:
    print("FAIL")
else:
    print(test.sequence)
    print("PASS")

print("Using inproper argument (setting non-existent element at index 100 to 0)...")
print("Error Expected:")
try:
    test.set(100,0)
    print("FAIL, should have caught error!")
except:
    print("Error caught! Sequence unchanged.")
    print("PASS")

print("")

print("Testing method 'get' from class 'SeqT' using sequence '[0,2,3]'")
                                        #Testing get method from SeqT
print("Using proper argument (getting value at 0)...")
print("Expecting '0' to print:")
if test.get(0) != 0:
    print("FAIL")
else:
    print(test.get(0))
    print("PASS")
print("Using inproper argument (getting value at 100)...")
print("Error Expected:")
try:
    test.get(100)
    print("FAIL, should have caught error!")
except:
    print("Error caught! Sequence unchanged.")
    print("PASS")

print("")

print("Testing method 'size' from class 'SeqT' using sequence '[0,2,3]'")
                                          #Testing size method from SeqT
print("Expecting '3' to print:")
if test.size() != 3:
    print("FAIL")
else:
    print(test.size())
    print("PASS")

print("")

print("Testing method 'indexInSeq' from class 'SeqT' using sequence '[0,2,3]'")
                                      #Testing indexInSeq method from SeqT
print("Using proper argument (searching exact value 0)...")
print("Expecting '0' to print:")
if test.indexInSeq(0) != 0:
    print("FAIL")
else:
    print(test.indexInSeq(0))
    print("PASS")
print("Using odd argument (searching inner value 2.5)...")
print("Expecting '1' to print:")
if test.indexInSeq(2.5) != 1:
    print("FAIL")
else:
    print(test.indexInSeq(2.5))
    print("PASS")
print("Using inproper argument (searching nonexistent value 100)...")
print("Error Expected:")
try:
    print(test.indexInSeq(100))
    print("FAIL, error not caught!")
except:
    print("Error caught! Sequence unchanged.")
    print("PASS")

print("")
```

```
print("------------------------------------------------------")

print("")

print("Testing file CurveADT")

    #Testing file CurveADT
print("Creating object 'test' with input file 'input.txt' of x^2 from 0 to 5...")
try:
    test = CurveADT.CurveT("input.txt")
    print("PASS")
except:
    test = CurveADT.CurveT("src/input.txt")
print("Expecting object 'test' to contain two sequences of the x and y vals:")
try:
    print(test.xVals.sequence)
    print(test.yVals.sequence)
    print("PASS")
except:
    print("FAIL")
print("Attempting to create object 'test2' with non-existent file...")
print("Error expected:")
try:
    test2 = CurveADT.CurveT()
    print("FAIL")
except:
    print("Error caught!")
    print("PASS")

print("")

print("Testing method 'linVal' from class 'CurveT' using 'input.txt'")
print("Using proper argument (interpolating 2.5)...")
print("Expecting roughly 6.25:")
if test.linVal(2.5) < 6.25*0.9 or test.linVal(2.5) > 6.25 * 1.1:
    print("FAIL")
else:
    print(test.linVal(2.5))
    print("PASS")

print("Using boundary case (left side data point 0)...")
print("Expecting 0:")
if test.linVal(0) != 0:
    print("FAIL")
else:
    print(test.linVal(0))
    print("PASS")

print("Using boundary case (right side data point 5)...")
print("Expecting erratic behaviour:")
if test.linVal(5) != None:
    print("FAIL")
else:
    print(test.linVal(5))
    print("PASS")

print("")

print("Testing method 'quadVal' from class 'CurveT' using 'input.txt'")
print("Using proper argument (interpolating 2.5)...")
print("Expecting roughly 6.25:")
if test.quadVal(2.5) < 6.25*.9 or test.quadVal(2.5) > 6.25*1.1:
    print("FAIL")
else:
    print(test.quadVal(2.5))
    print("PASS")
print("Using boundary case (left side data point 0)...")
print("Expecting 0:")
if test.quadVal(0) != 0.0:
    print("FAIL")
else:
    print(test.quadVal(0))
    print("PASS")
print("Using boundary case (right side data point 5)...")
print("Expecting erratic behaviour:")
if test.quadVal(5) != None:
    print("FAIL")
else:
    print(test.quadVal(5))
```

```
    print("PASS")

print("")

print("Testing method 'npolyVal' from class 'CurveT' using 'input.txt'")
print("Using proper arguments (interpolating 2.5 with proper rank of 2)...")
print("Expecting roughly 6.25:")
try:
    test.npolyVal(2,2.5)
    print("PASS")
except:
    print("FAIL")
print("Using inproper arguments (interpolating 2.5 with inproper rank of 100)...")
print("Expecting roughly 6.25 with error:")
try:
    test.npolyVal(100,2.5)
    print("PASS")
except:
    print("FAIL")
print("Using absurd arguments (interpolating 100 with inproper rank of 100)...")
print("Expecting wild result:")
try:
    test.npolyVal(100,100)
    print("PASS")
except:
    print("FAIL")
```

H Code for Partner's SeqADT.py

```
## @file SeqADT
#  @author Thomas Shang
#  @brief Provides the SeqT ADT class for representing curves
#  @date 1/22/2018


## @brief An ADT that represents a Sequence
class SeqT:

        ## @brief SeqT constructor
        #  @details Initializes a SeqT object that contains an empty array
        def __init__(self):
                self.data = []


        ## @brief Inserts or appends the value v based on the value i
        #  @details If the value of i is equivalent to the current size of the data array
             the value is appened.
        #           Otherwise if the value of i is within the indexes of the array the
             value is inserted at the value i.
        #                       If the value i is neither of the above values the value
             is not inserted
        # @param i desired index for added value
        # @param v real number to be added
        def add(self, i, v):
                if(i == self.size()):
                        self.data.append(v)
                elif(i < self.size()):
                        out = self.data[:i];
                        back = self.data[i:]
                        out.append(v);
                        out.extend(back)
                        self.data = out
                else:
                        print ("illegal index")

        ## @brief Removes value from sequence at position i
        #  @param i index of value to be removed
        def rm(self, i):
                del self.data[i]

        ## @brief Sets value of stored at index i to v
        # @param i index of value to be changed
        # @param v value of new value

        def set(self, i, v):
                self.data[i] = v

        ## @brief Returns value stored at index i in sequence
        # @param i index of value to be returned
        # @return value stored in sequence at index i
        def get(self, i):
                return self.data[i]

        ## @brief Returns size of sequence
        # @return size of sequence
        def size(self):
                return len(self.data)


        ## @brief Returns index value where v falls between the index value i and i + 1
        #  @param v real value
        #  @return index value of value v
        def indexInSeq(self , v):
                for i in range(self.size()-1):
                        if(self.get(i) <= v and self.get(i+1) >= v):
                                return i

                print("Value not within sequence")
```

I Code for Partner's CurveADT.py

```
## @file CurveADT
#  @author Thomas Shang
#  @brief Provides the CurveT ADT class for representing curves
#  @date 1/22/2018

import numpy
from SeqADT import SeqT

## @brief An ADT that represents a curve
class CurveT:

        ## @brief CurveT constructor
        #  @details Initializes a CurveT object with x and y coordinates stored in
             sepereate SeqT objects
        #  @param s File name input for data to initialize object
        #  @exception FileNotFoundError throws if file name supplied does not exist
        def __init__(self, s):
                try:
                        file = open(s, "r")
                        self.x = SeqT();
                        self.y = SeqT();
                        index = 0
                        for line in file:
                                data = line.split(", ")
                                self.x.add(index,float(data[0]))
                                self.y.add(index,float(data[1]))
                                index += 1
                except FileNotFoundError:
                        print("File does not exist")


        ## @brief Calculates the value of x based on the linear interpolation of the
             surrounding points
        #  @param x value of x coordinate
        #  @return The value of x calculated via linear interpolation
        def linVal(self,x):
                i = self.x.indexInSeq(x)
                return (self.y.get(i+1) - self.y.get(i))/(self.x.get(i+1) -
                     self.x.get(i))*(x - self.x.get(i)) + self.y.get(i)


        ## @brief Calculates the value of x based on the quadratic interpolation of the
             surrounding points
        #  @param x value of x coordinate
        #  @return The value of x calculated via quadratic interpolation
        def quadVal(self,x):
                i = self.x.indexInSeq(x)
                a =  self.y.get(i) + (self.y.get(i+1) - self.y.get(i-1))/(self.x.get(i+1)
                     - self.x.get(i-1))*(x - self.x.get(i))
                b = (self.y.get(i+1) - 2*self.y.get(i) +
                     self.y.get(i-1))/(2*(self.x.get(i+1) - self.x.get(i)) ** 2)*((x -
                     self.x.get(i)) ** 2)
                return a+b

        ## @brief Calculates the value of x by using the provided polynomial returned by
             the polyfit function
        #  @param n degree of polynomial function
        #  @param x value of x coordinate
        #  @return The value of x calculated via the provided polynomial by the polyfit
             function
        def npolyVal(self, n , x):
                p = numpy.polyfit(self.x.data, self.y.data, n)
                y = 0
                for i in range(n+1):
                        y += p[i]*(x ** (n - i))

                return y
```

J Makefile

```
PY = python3
DOXY = doxygen
DOXYCFG = doxConfig

RMDIR = rm -rf

.PHONY: test doc clean

test:
	$(PY) src/testSeqs.py

doc:
	$(DOXY) $(DOXYCFG)
	cd latex && $(MAKE)

clean:
	@- $(RMDIR) html
	@- $(RMDIR) latex
```