Assignment 1 Report

Hamid Ghasemi

January 31, 2018

1 Testing of the Original Program

Basically, the testing for SeqADT starts with defining a variable to SeqT class and then setting it to a random sequence and use the object that has been created in SeqT for all the functions. And now we can test all functions which have been defined in the SeqT. Checking add function is important since if i is more than array index puts the input element at the end, if not it inserts the element in the desired placement. The testing module is made by try, assert, and except. After assert there is a print statement that if it's true it passes, otherwise it would print fail which is in except section. And all the functions have the same type of testing. For CurveT the test run starts with assigning a variable to a text file such as (CurveT("?.txt")). And then it does the exact same thing as SeqT test. If it has an error, it would print fail, and it prints pass if it is correct. All results passed for both SeqT and CurveT. All the problems are covered although there are not too many test cases for all the functions.

2 Results of Testing Partner's Code

I got 9/9 with changing variables name of my partners code, and got errors without changing them. It's reasonable because in assignment variable name is not specified, so everyone can pick their own name for the variables. His code is reliable and it can be used by other people.

3 Discussion of Test Results

3.1 Problems with Original Code

In original code there were cases that I forgot to cover like what happens when denominator of the function linVal or quadval are equal to zero. That would cause an error if it

is not specified by the code. And another example was that in the SeqT, I forgot that the program will return error if for set function i is outside of range seq[a]. And there were other similar problems to this case for other functions in SeqT.

3.2 Problems with Partner's Code

My partner's code works with my testrun, however in order to run the code the initial function name has to be changed since I defined my sequence with variable name seq while my partner has assigned his sequence with variable name of sequence; therefore, when the testrun runs it gives an error, saying seq is not defined.

3.3 Problems with Assignment Specification

The assignment is straightforward, however there are some parts that are not completely explained, for example in curveT for function quad is not defined what is x0 so x0 can be considered element before x1 or after x2. There were other parts that didn't specify what happens in other cases and this would result a huge difference in writing the program.

4 Answers

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

   Constructor: init for both SeqT and CurveT (first function)
   Accessor: get, size, indexInSeq, linVal, quadVal, npolyVal
   Mutator: add, rm, set

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

   Advantage would be that the function is already in the library so we don't need to spend time on making a function that works. For disadvantage I would say sometimes the libraries that have been created are not in Program; therefore, developer must download them from the internet and this would take time. And another disadvantage is library cannot be changed.

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?

   In Pythons list by using and index we can access the elements while in SeqT we must use accessor to access the elements. Python's list is made for specific job and

cannot be changed while SeqT is a class that has been designed and can be changed very easily. Plus Python's list is easy to use, for example by using numpy there is no need to create another function that does what numpy does. Hence, it would be less time consuming and resources.

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

   Then it means the numbers are not in sorted array. If the array does not get sorted, it would change the output. To search and find number X between xi and xi+1 would give a different output for i. Exp
   x= 2.3 [-1, 1 , 6, 8] i= 1
   x= 2.3 [6, -1, 1, 8] i = 2

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

   For the same x value for linVal(x) and npolyVal(n, x) won't be equal. Since linVal is interpolation approximation, and npoly is regression approximation, they won't necessarily get the same result. However, it is possible that linVal and npoly will get the same result.

   Answer

E Code for SeqADT.py

```
## @file    SeqADT.py
#  @author Hamid Ghasemi 400028420
#  @brief Provides the SeqT class to show the sequence
#  @date    22/1/2018


## @brief Making a sequences
class SeqT:
    ## @brief SeqADT constructor
    #  @details it makes an empty sequence
    def __init__(self):
        self.seq = []

    ## @brief Adds number to sequence
    #  @details It adds value to the sequence based on the index number
    #  @param i is the index of the sequence
    #  @param v is the value that user inputs

    def add(self, i, v):
        if len(self.seq) < i:
            self.seq.append(v)
        else:
            self.seq.insert(i, v)

    ## @brief rm removes the values from the sequence based on the index number
    #  @param i is the index of the sequence
    def rm(self, i):
        del self.seq[i]

    ## @brief set set number of the sequence with number that user inputs
    #  @param i is the index of the sequence
    #  @param v is the value
    def set(self, i,v):
        self.seq[i] = v

    ## @brief get get numbers by using index from sequence
    #  @param i is the index of the sequence
    #  @return The number that user looks for based on the index
    def get(self, i):
        if i > len(self.seq)-1:
            print ("Value is out of boundary")
        else:
            return self.seq[i]

    ## @brief Size Shows the length of the sequence
    #  @return The length of the sequence
    def size(self):
          return len(self.seq)

    ## @brief indexInSeq Checks if value is in the range, and it finds
    #   number before the value which user inputs in sequence and finds its index
    #  @param v is value that function gets from the user
    #  @return i The index of the sequence
    def indexInSeq(self, v):
        for i in range (0, len(self.seq)-1):
            if (self.get(i) <= v and  v <= self.get(i+1)):
                return i
```

F Code for CurveADT.py

```
## @file    CurveADT.py
#  @author Hamid Ghasemi 400028420
#  @brief Provides the SeqT class to show the sequence
#  @date    22/1/2018


from SeqADT import *
import numpy

## @brief Making class curveT
class CurveT:
    ## @brief CurveT constructor
    #  @details It opens a textfile and reads it, then makes two arrays equal to
    #  first and second column which are bunch of numbers
    #  @param s is string that user inputs
    def __init__(self, s):
        f = open(s, 'r')
        Data = f.readlines()
        self.x = SeqT()
        self.y = SeqT()
        Num = [ ]
        for i in Data:
            Num = i.split(', ')
            self.x.seq.append(float(Num[0]))
            self.y.seq.append(float(Num[1]))
        f.close()


    ## @brief linVal is a function that using interpolation between data values
    #  @details Use the input value that user inserts and find the interpolation
    #  between numbers before and after the input value that present in the sequence
    #  @param x is the input value
    #  @return y is the value that returns at the end
    def linVal(self, x):
        d = self.x.indexInSeq(x)

        x1 = self.x.seq[d]
        x2 = self.x.seq[d+1]

        y1 = self.y.seq[d]
        y2 = self.y.seq[d+1]

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

    ## @brief quadVal is a function that using interpolation between data values
    #  @details Use the input value that user inserts and find the interpolation
    #  between numbers before and after the input value that present in the sequence
    #  @param x is the input value
    #  @return y is the value that returns at the end
    def quadVal(self, x):
        d = self.x.indexInSeq(x)
        if d == 0:
            print ("There is no number for x0 and y0")
        else:
            x0 = self.x.seq[d-1]
            x1 = self.x.seq[d]
            x2 = self.x.seq[d+1]

            y0 = self.y.seq[d-1]
            y1 = self.y.seq[d]
            y2 = self.y.seq[d+1]

            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

    ## @brief npolyVal is a function that finds approximatation value
    #  @details It makes the line of best fit for polynomials
    #  @param n is the integer value that user inputs
    #  @param x is the real value which user inputs
```

```
#  @return y is the value that is best fit that returns at the end
def npolyVal(self, n, x):
    x_seq = self.x.seq
    y_seq = self.y.seq
    line = numpy.polyfit(x_seq, y_seq, n)
    y = numpy.polyval(line, x)
    return y
```

G Code for testSeqs.py

```
## @file    testSeqs.py
#  @author Hamid Ghasemi 400028420
#  @brief Provides the SeqT class to show the sequence
#  @date   22/1/2018


from SeqADT import *
from CurveADT import *

##  @brief Tests the add funtion method of the SeqT class
#   @detail Checks for the index in sequence, if i is within the length it will add otherwise it adds
     at the end
#   the value to its desired position else at the end of the list
t = SeqT()
def test_add():
    t.add(0, 1)
    t.add(1, 2)
    t.add(2, 3)
    t.add(1, 4)
    # t = [1, 4, 2, 3]
    try:
        assert t.seq[0] == 1 and t.seq[1] == 4
        print ("add test passed")
    except AssertionError:
        print ("add test failed")

## @brief test_rm function remove the value that user has input
#  @detail If index i is in the list return a value
#   return value
def test_rm():
    # t = [1, 4, 2, 3]
    t.rm(1)
    # t = [1, 2, 3]
    try:
        assert t.seq[1] == 2
        print ("remove test passed")
    except AssertionError:
        print ("remove test failed")

## @brief test_set it tests and set value
#  @detail Test the cases where input i is in the list
#   set a value 3 to 4
def test_set():
    # t = [1, 2, 3]
    t.set(2, 4)
    # t = [1, 2, 4]
    try:
        assert t.seq[2] == 4
        print ("set test passed")
    except AssertionError:
        print ("set test failed")

## @brief test_get tests if the required value is outputted
#  @detail Test the cases where input i (index) is in the list
#   return value 2
def test_get():
    # t = [1, 2, 4]
    t.get(1)
    # t [1] = 2
    try:
        assert t.seq[1] == 2
        print ("get test passed")
    except AssertionError:
        print ("get test failed")

## @brief Verifies the size of the sequence
def test_size():
    # t = [1, 2, 4]
    try:
        assert t.size() == 3
        print ("size test passed")
    except AssertionError:
        print ("size test failed")

## @brief tests if the input value is between two numbers within a sequence
# @detail a real value in the middle
```

```
# it must return a index 1
def test_indexInSeq():
    # t = [1, 2, 4]
    # after using indexInSeq it should give us i = 0
    t.indexInSeq(3)
    try:
        assert t.indexInSeq(3) == 1
        print ("indexinseq test passed")
    except AssertionError:
        print ("indexinseq test failed")

"TESTING CurveADT File"

## @brief Tests the y from the method linVal
#  @detail checks for value within a value in the middle of the sequence
#  Check a value within the sequence
d = CurveT("1.txt")
def test_linVal(x):
    # xcontains: x = [1, 3, 4, 6]
    # ycontains: y = [4, 6, 8, 11]

    d.linVal(3.4)
    try:
        assert d.linVal(3.4) == 6.8
        print ("linVal test passed")
    except AssertionError:
        print ("linVal test failed")

## @brief Tests the yvalue from the method quadVal
#  @detail checks for value within a value in the middle of the sequence
#   Check value whithin sequence
def test_quadVal(x):
    # xcontains: x = [1, 3, 4, 6]
    # ycontains: y = [4, 6, 8, 11]

    d.quadVal(3.4)
    try:
        assert round(d.quadVal(3.4)) == 7
        print ("quadVal test passed")
    except AssertionError:
        print ("quadVal test failed")

## @brief npolyVal tests the y value with n degree
#  @detail input number is 1 for n and 3 for x which return 6.54
def test_npolyval(n, x):
    # xcontains: x = [1, 3, 4, 6]
    # ycontains: y = [4, 6, 8, 11]
    c = d.npolyVal(1,3)
    try:

        assert round(c, 2) == 6.54
        print ("npolyval test passed")
    except AssertionError:
        print ("npolyval test failed")



test_add()
test_rm()
test_set()
test_get()
test_size()
test_indexInSeq()
test_linVal("1.txt")
test_quadVal("1.txt")
test_npolyval(1, 3)
```

H Code for Partner's SeqADT.py

```
## @file SeqADT.py
#  @author Victor Chen
#  @brief Provides the SeqT ADT class for representing sequences
#  @date 1/22/2017

## @brief ADT representing a sequence
class SeqT:

    ## @brief SeqT constructor
    #  @details Initializes a SeqT object with an empty sequence.
    def __init__(self):
        self.sequence = []

    ## @brief Adds a new value to the sequence
    #  @param i an integer representing the index to add a new value
    #  @param v a real number to add to the sequence at index i
    #  @exception Exception throws is supplied index is greater than length of the sequence or a
        negative value
    def add(self, i: "int index", v: "real to add"):
        #values can only be added within the existing sequence, or
        #immediately after the last entry
        if self.sequence == [] and i != 0:
            raise Exception("Sequence currently empty! Use index at 0!")
        elif i > len(self.sequence):
            raise Exception("Index greater than sequence length!")
        elif i < 0:
            raise Exception("Index cannot be smaller than zero!")

        if i == len(self.sequence):
            self.sequence.append(v)
        else:
            temp = self.sequence[0:i]
            temp.append(v)
            for index in range(len(self.sequence)-i):
                temp.append(self.sequence[index+i])
                self.sequence = temp

    ## @brief Returns the element at index i in the sequence and pops it from the sequence, reducing
        the sequence length by 1
    #  @param i an integer representing the index of the element to be removed
    #  @exception Exception throws is supplied index is greater than length of the sequence or a
        negative value
    #  @return The removed element
    def rm(self, i: "int index to be removed"):
        try:
            self.sequence.pop(i)
        except:
            if i < 0:
                raise Exception("Index cannot be smaller than zero!")
            elif i > len(self.sequence):
                raise Exception("Index greater than sequence length!")

    ## @brief Replaces the value at i with the value of v
    #  @param i an integer representing the index of the element to be replaced
    #  @param v the value to replace the element at index i
    #  @exception Exception throws is supplied index is greater than length of the sequence or a
        negative value
    def set(self, i: "int index", v: "real"):
        try:
            self.sequence[i] = v
        except:
            if i < 0:
                raise Exception("Index cannot be smaller than zero!")
            elif i > len(self.sequence):
                raise Exception("Index greater than sequence length!")

    ## @brief Returns the current size of the sequence
    #  @return The integer length of the sequence
    def size(self):
        return len(self.sequence)

    ## @brief Returns an index i such that SeqT.get(i) <= v <= SeqT.get(i+1)
    #  @param v the real value being searched for
    #  @return The integer value of the index at which v appears
    #  @exception returns -1 when element v does not exist in the sequence
    def indexInSeq(self, v: "real"):
```

```
for i in range(len(self.sequence)):
    if self.sequence[i] == v:
        return i
return -1
```

I Code for Partner's CurveADT.py

```
## @file CurveADT.py
#  @author Victor Chen
#  @brief Provides the CurveT ADT class for representing sequences
#  @date 1/22/2017

import numpy

## @brief ADT representing a curve
class CurveT:

    ## @brief CurveT constructor
    #  @details Initializes a curveT object with x and y co-ordinates from an specified text file
    #  @param s the name of the text file formatted such that x and y coordinates are per line and
         separated by a ", "
    def __init__(self, s):
        self.x = []
        self.y = []

        txt = open(s, "r")
        lines = txt.readlines()

        for i in lines:
            splitLine = i.split(", ")
            self.x.append(int(splitLine[0]))
            self.y.append(int(splitLine[1]))

    ## @brief Returns the linear interpolation around the value of the x-coordinate
    #  @param x an integer representing the x value of the linear interpolation formula
    #  @exception Exception throw is supplied when x is not found in the coordinates or if x is out of
         range for the formula
    #  @return The linear interpolation around x
    def linVal(self, x: "real"):

        if self.x[0] == x or self.x[len(self.x) - 1] == x:
            raise Exception("X value out of range!")

        found = False
        for i in range(len(self.x)):
            if self.x[i] == x:
                y1 = self.y[i-1]
                y2 = self.y[i+1]
                x1 = self.x[i-1]
                x2 = self.x[i+1]
                found = True
                break

        if found:
            return ((y2 - y1) / (x2 - x1) * (x - x1)) + y1
        else:
            raise Exception("x not found!")

    ## @brief Returns the quadratic interpolation around the value of the x-coordinate
    #  @param x an integer representing the x value of the quadratic interpolation formula
    #  @exception Exception throw is supplied when x is not found in the coordinates or if x is out of
         range for the formula
    #  @return The quadratic interpolation around x
    def quadVal(self, x: "real"):

        if self.x[0] == x or self.x[1] == x or self.x[len(self.x) - 1] == x:
            raise Exception("x value out of range!")

        found = False
        for i in range(len(self.x)):
            if self.x[i] == x:
                x0 = self.x[i-2]
                x1 = self.x[i-1]
                x2 = self.x[i+1]
                y0 = self.y[i-2]
                y1 = self.y[i-1]
                y2 = self.y[i+1]
                found = True
                break

        if found:
            return y1 + ((y2-y0)*(x-x1))/(x2-x0) + (y2 - (2*y1) + y0)*((x-x1)**2)/(2*((x2-x1)**2))
        else:
```

```
        raise Exception("X not found!")


## @brief Returns the npoly-interpolation around the value of x to the degree of n
#  @param n the integer value representing the degree of the polynomial
#  @param x the value on which the interpolation is being done around
#  @return The npoly-interpolation around x
def npolyVal(self, n: "integer degree", x: "real"):

    p = numpy.poly1d(numpy.polyfit(self.x, self.y, n))

    return p(x)
```

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
```