Assignment 1 Report

Stephanus Jonatan, jonatans

January 30, 2018

Introductory blurb.

1 Testing of the Original Program

Description of approach to testing. Rationale for test case selection. Summary of results. Any problems uncovered through testing.

2 Results of Testing Partner's Code

Summary of results.

Overall, the partner's code ran well. However, when a string was inputted as one of the data points, a ValueError occured which specified that it could not convert a string into a float; this crashed the program. The same error also occured when the any line in the text file being tested was empty. In addition, when trying to run the code with only one data point, an error occurs.

3 Discussion of Test Results

3.1 Problems with Original Code

Under the quadVal() function, negative numbers did not work as well as positive numbers (i.e. they were not as accurate). For example, when trying to find the the y value when x = 3.5, with a data ranging from -4.0 to 4.0 (increments of 1), the value returned was the square of 3.5. However, when trying to find the y value with x = -3.5, the value returned was 17.25 which is not the square of -3.5.

3.2 Problems with Partner's Code

If there is an empty line anywhere in the text file used for testing, nothing can be initialized because it won't be able to change a string into a float. If the author of the partner code used the strip() method before using the split() method, then any white space would have been ignored and would still function. However, due to the assignment specification only specifying to read a text file with data points seperated only by a comma and a space, nothing is really wrong with the code because the requirements were met.

3.3 Problems with Assignment Specification

The assignment specifications did not specify how errors would be handled. An example, other than the previously mentioned problem in section 3.2 of this report, is that when elements are added to a sequence using the add() method in SeqADT.py and it is added outside of the sequence's index, it was not specified whether to append that element to the end, or to not append it at all and provide a maximum index number from the start instead. Depending on the developers decision, it could have gone either way.

4 Answers

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

Methods	Type
SeqADT	
__init__	constructor
add	mutator
rm	mutator
set	mutator
get	accessor
size	accessor
indexInSeq	accessor
CurveADT	
__init__	constructor
linVal	accessor
quadVal	accessor
npolyVal	accessor

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

   Using a library like numpy will save a lot of time and effort from having to build certain methods that you want your program to do. However, there is a learning curve into learning how to use an external library. Also, depending on the method you are using in the external library, a developer might be able to make a faster method that does the same thing as the method in the library. However, despite these few disadvantages, the time and effort saved from using a pre-existing library usually outweighs the cons.

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?

   The difference between SeqT list type is and Python's pre-existing list type is that Python already has methods for its list type, whereas SeqT is a custom list type that must have methods made by the developer. This is why using Python's pre-existing list type is better; the developer does not have to spend time creating methods that euther construct, access, or mutate the list like SeqT list types. However, if the developer wants custom outputs for the list, then creating a custom list type like SeqT will be beneficial.

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

   If this assumption was no longer applied, it would cause problems when using the linVal() and quadVal() functions since it relies on a custom accessor indexInSeq() that was built with intention of it being used on a sorted sequence. This would then cause problems in the output of both linVal() and quadVal(), resulting in a more inaccurate estimate of a y value at a user specified x value.

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

   The only difference between linVal() and npolyVal() - assuming the test file for both has data for a linear graph - is differences in rounding when finding the y value at a given x value. However, this does not matter because the difference is too small to make a significant change.

E Code for SeqADT.py

```
## @file SeqADT.py
#  @author Stephanus Jonatan
#  @date January 21, 2018

## @brief SeqT is a class that creates an empty list/sequence.
#  @details SeqT has a constructor, and a few accessors and mutators.
class SeqT(object):
    ## @brief Initializes an empty Sequence.
    def __init__(self):
        self.Seq = []

    ## @brief add(i, v) inserts an element into a list (mutator).
    #  @details It either appends or inserts at a specific position in the list.
    #  @param i is the index of the list to which the user wants to add a value to.
    #  @param v is the value the user wants to add to the list.
    #  @return An updated list is returned.
    def add(self, i, v):
        # Checks if sequence is empty
        if self.Seq == 0:
            return self.Seq.append(v)
        # Checks if index i exists
        elif i >= len(self.Seq):
            return self.Seq.append(v)
        else:
            return self.Seq.insert(i, v)

    ## @brief Removes an element from a list (mutator).
    #  @param i is the index of the element that the user wants to remove.
    #  @return An updated list is returned.
    def rm(self, i):
        return self.Seq.pop(i)

    ## @brief Modifies an element in a list (mutator).
    #  @param i is the index of the element that the user wants to modify.
    #  @param v is the value that the user wishes to replace the old value with.
    #  @return An updated list is returned.
    def set(self, i, v):
        # Checks if index i exists
        if i >= len(self.Seq):
            # prints error message when index doesn't exists
            print("Index does not exists. Nothing to modify")
        else:
            self.Seq[i] = v
            return self.Seq

    ## @brief Accesses an index of a list (accessor).
    #  @param i is the index that is being accessed.
    #  @return Returns the value at the index i.
    def get(self, i):
        # Checks if index i exists
        if i >= len(self.Seq):
            # prints error message when index doesn't exists
            print("Index does not exists")
        else:
            return self.Seq[i]

    ## @brief Checks for the size of a list.
    #  @returns the the size of a list.
    def size(self):
        return len(self.Seq)

    ## @brief Finds the approximate position for a value in a sorted list (accessor).
    #  @details The value does not have to exist in the list, but has to be >= than the first element
      and <= the last element of the list.
    #  @param v is the value being searched for in the list.
    #  @return Returns the approximate index of the value v.
    def indexInSeq(self, v):
        for i in range(0, self.size()):
            if (self.get(i) <= v) and (v <= self.get(i + 1)):
                return i
```

F Code for CurveADT.py

```
## @file CurveADT.py
#  @author Stephanus Jonatan
#  @date January 21, 2018

from SeqADT import SeqT
from numpy import *

## @brief CurveT is a class that analyzes data of a polynomial.
#  @details CurveT uses interpolation and regression to estimate a y value for a given x value.
#  @details CurveT uses imported classes SeqT and the numpy library.
class CurveT(object):
    ## @brief Initializes two empty sequences that stores x and y coordinates from a .txt file.
    #  @details reads a .txt file with data of a polynomial and stores each x and y coordinate
        sequentially in their own respective lists.
    def __init__(self, s):
        # Initializes two empty sequences
        self.Points_x = SeqT()
        self.Points_y = SeqT()
        # Reads file "s" that contains data points (e.g. x, y) separated by a
        # comma and a space, and with each point on a newline.
        Seq_file = open(s, 'r')
        lines = Seq_file.readlines()
        # Reads each line one at a time, and stores it in i
        for i in lines:
            # Strips all new line characters in the line, and stores it back into i
            i = i.strip()
            # Splits a string at every occurring comma into elements, and
            # adds it to a list, add_point.
            add_point = i.split(",")
            # Adds add_point[0] and add_point[1] into lists Points_x
            # and Points_y respectively as an int data type.
            self.Points_x.add(self.Points_x.size(), float(add_point[0]))
            self.Points_y.add(self.Points_y.size(), float(add_point[1]))

    ## @brief linVal(x) calculates the approximate y value of a given x value using linear
        interpolation.
    #  @param x is the given value.
    def linVal(self, x):
        # finds approximate position (i) of value x in sequence Points_x
        # (either the exact position, or the position to the left of it)
        i = self.Points_x.indexInSeq(x)
        # finds the x and y value at the ith position
        x1 = self.Points_x.Seq[i]
        y1 = self.Points_y.Seq[i]
        # finds the x and y values at the position to the right of i
        x2 = self.Points_x.Seq[i + 1]
        y2 = self.Points_y.Seq[i + 1]

        # linear interpolation formula
        y = ((y2 - y1) / (x2 - x1)) * (x - x1) + y1
        return y

    ## @brief quadVal(x) calculates the approximate y value of a given x value using quadratic
        interpolation.
    #  @param x is the given value.
    def quadVal(self, x):
        # finds approximate position (i) of value x in sequence Points_x
        # (either the exact position, or the position to the left of it)
        i = self.Points_x.indexInSeq(x)
        # finds the x and y values at the position to the left of i
        x0 = self.Points_x.Seq[i - 1]
        y0 = self.Points_y.Seq[i - 1]

        # finds the x and y value at the ith position
        x1 = self.Points_x.Seq[i]
        y1 = self.Points_y.Seq[i]

        # finds the x and y values at the position to the right of i
        x2 = self.Points_x.Seq[i + 1]
        y2 = self.Points_y.Seq[i + 1]

        # quadratic interpolation formula
        y = y1 + ((y2 - y0) / (x2 - x0)) * (x - x1) + \
            ((y2 - 2 * y1 + y0) / (2 * (x2 - x1) ** 2)) * (x - x1) ** 2
        # y = y0 * (((x - x1) * (x - x2)) / ((x0 - x1) * (x0 - x2))) + \
        #     y1 * (((x - x0) * (x - x2)) / ((x1 - x0) * (x1 - x2))) + \
```

```
    #     y2 * (((x - x0) * (x - x1)) / ((x2 - x0) * (x2 - x1)))
    return y

## @brief npolyVal(x) calculates the approximate y value of a given x value using regression.
#  @details Uses a best fit polynomial to approximate the y value.
#  @param n is the degree of the best fit polynomial.
#  @param x is the given value.
def npolyVal(self, n, x):
    # Checks if degree is at least 1 or greater
    if n >= 1:
    # Evaluates the coefficients of a polynomial with data points
    # Points_x and Points_y, with a polynomial degree of n, and stores
    # it in a list, C.
        C = polyfit(self.Points_x.Seq, self.Points_y.Seq, n)
        # Evaluates polynomial coefficients, C, at the value of x
        y = polyval(C, x)
        return y
    # Otherwise print an error message
    else:
        print("Improper Degree. Must have a degree of at least 1 or greater.")
```

G Code for testSeqs.py

```
from CurveADT import CurveT
from SeqADT import SeqT


def main():
    # check = True
    # while (check):
    #     Choice1 = input("Which would you like to test:\n"
    #                     "linVal                 (1)\n"
    #                     "quadVal                (2)\n"
    #                     "npolyVal               (3)\n"
    #                     "All other functions    (4)\n"
    #                     "Quit                   (0)\n"
    #                     "---> ")
    #     if (Choice1 == 0):
    #         print("Have a nice day!")
    #         check = False
    #     elif (Choice1 == 1):
    #         file = raw_input("Type the test file name you would like to use for linVal
        (example.txt): ")
    test1 = CurveT("lin_test.txt")
    x = -3.5 #input("What x value would you like to get the y value for?: ")
    y = test1.linVal(x)
    print("linVal: The y value for %f is %f" % (x, y))
    #     elif (Choice1 == 2):
    #         file = raw_input("Type the test file name you would like to use for quadVal
        (example.txt): ")
    test2 = CurveT("quad_test.txt")
    x = -3.5 #input("What x value would you like to get the y value for?: ")
    y = test2.quadVal(x)
    print("quadVal: The y value for %f is %f" % (x, y))
    #     elif (Choice1 == 3):
    #         file = raw_input("Type the test file name you would like to use for npolyVal
        (example.txt): ")
    test3 = CurveT("npoly_test.txt")
    n = 3 #input("What is the degree of the polynomial?: ")
    x = -4.5 #input("What x value would you like to get the y value for?: ")
    y = test3.npolyVal(n, x)
    print("npolyVal: The y value for %f is %f" % (x, y))
    #     elif (Choice1 == 4):
    #         print("Testing all other functions...\n")
    test4 = SeqT()
    test4.add(0, 1)
    test4.add(1, 2)
    test4.add(0, 0)
    if (test4.Seq[0] == 0 and test4.Seq[1] == 1 and test4.Seq[2] == 2):
        test_add = "Pass"
    else:
        test_add = "Fail"
    if (test4.size() == 3):
        test_size = "Pass"
    else:
        test_size = "Fail"
    if (test4.get(1) == 1):
        test_get = "Pass"
    else:
        test_get = "Fail"
    test4.rm(2)
    if (len(test4.Seq) == 2):
        test_rm = "Pass"
    else:
        test_rm = "Fail"
    test4.set(1, 1234)
    if (test4.Seq[1] == 1234):
        test_set = "Pass"
    else:
        test_set = "Fail"
    print("Function Tested:\tResult:\n"
          "add()  \t\t%s\n"
          "rm()   \t\t%s\n"
          "get()  \t\t%s\n"
          "set()  \t\t%s\n"
          "size()\t\t%s\n"
          % (test_add, test_rm, test_get, test_set, test_size))
    #
    #     else:
```

```
    #           print("Invalid choice. Please select options 1-4, or 0 to quit.")

main()
```

H Code for Partner's SeqADT.py

```
## @file SeqADT.py
#  @author Ji Who Choi
#  @brief Provides the Seq ADT class for representing sequence
#  @date 1/21/2018

## @brief An ADT that represents a sequence
class SeqT:
  ## @brief SeqT constructor
  #  @details Initializes a SeqT object
    def __init__(self):
        self.elements = []

  ## @brief Gets the index and the value and insert it to the SeqT object at the
  #  index
  #  @details Assume that the index, i is a positive integer
  #  @param i an integer index where the value, v should be inserted
  #  @param v a value added to the SeqT object
  #  @exception ValueError Throws if supplied i is not within the range of 0 to
  #  the length of the sequence + 1
    def add(self, i, v):
        if (i < len(self.elements) + 1):
           self.elements.insert(i,v)
        else:
            raise ValueError("Index is not within the range")

  ## @brief Gets the index,i and delete the value at i
  #  @details Assume that the index is less than the length of the sequence
  #  @param i an integer index where the value should be deleted
    def rm(self, i):
        if (i < len(self.elements)):
            del self.elements[i]

  ## @brief Gets the index,i and the value, v and sets v at i
  #  @details Assume that the index, i is a positive integer
  #  @param i an integer index where the value, v should be set
  #  @param v a value set to the SeqT object
    def set(self, i, v):
        if (i < len(self.elements)):
            self.elements[i] = v

  ## @brief Gets the index,i and return the value at i
  #  @details Assume that the index, i is a positive integer
  #  @param i an integer index where the value, v should be returned
    def get(self,i):
        return self.elements[i]

  ## @brief Gets the size of a SeqT object
  #  @return the size of a SeqT object
    def size(self):
       return len(self.elements)

  ## @brief Gets the value, v and finds the index of the sequence such that
  #  s.get(i) <= v <= s.get(i+1)
  #  @details Assume that the value, v is in the range. If not in the range,
  #  the Indexerror will occur.
  #  @param v is a real number
  #  @return the index such that s.get(i) <= v <= s.get(i+1)
    def indexInSeq(self, v):
        for i in range(len(self.elements)):
            if (self.get(i) <= v <= self.get(i + 1)):
                return i
```

I Code for Partner's CurveADT.py

```
## @file CurveADT.py
#  @author Ji Who Choi
#  @brief Provides the CurveADT class for representing a curve in the Cartesian
#  x-y plane
#  @date 1/21/2018

import SeqADT
import numpy as np
import os
## @brief An ADT that represents a curve
class CurveT:
  ## @brief CurveT constructor
  #  @details Initializes a CurveT object and assumes the input file, s is in
  #  the same directory as the class. Otherwise, an error occurs.
  #  x and y sequence state variables are set to the values of the first and
  #  second column of s.
  #  @param s the name of a text file in string

    def __init__(self,s):
        #Opens the input textfile,s
        theFolder = os.path.dirname(os.path.abspath(__file__))
        theFile = os.path.join(theFolder, s)
        fileread = open(s, "r")
        filelines = fileread.readlines()
        fileread.close()

        index = 0
        self.xValues = SeqADT.SeqT()
        self.yValues = SeqADT.SeqT()
        for line in filelines:
            splitline = line.split(', ')
            #Split the line to obtain x and y value
            self.x = float (splitline[0])
            self.xValues.add(index, self.x)
            splitline[1] = splitline[1].replace("\n","")
            #replace the end of the newline character to ""
            self.y = float (splitline[1])
            self.yValues.add(index,self.y)
            index += 1

  ## @brief Calculates the value of y using linear interpolation between the
  #  values on either side of x
  #  @details Assumes the point on the curve to the left of x is (x1, y1) and
  #  the point to the right of x is (x2, y2)
  #  @param x a real number
  #  @return the value of y
    def linVal(self, x):
        indexX = self.xValues.indexInSeq(x)
        #finds the first index f the sequence in the range
        x1 = self.xValues.get(indexX)
        #obtains the value at the indexX
        x2 = self.xValues.get(indexX + 1)
        y1 = self.yValues.get(indexX)
        y2 = self.yValues.get(indexX + 1)

        y = ((y2 - y1) / (x2 - x1)) * (x - x1) + (y1)
        return y

  ## @brief Calculates the value of y using quadratic interpolation of three
  #  points with corresponding subscripts of 0, 1, and 2
  #  @details Assumes the point on the curve to the left of x is (x1, y1) and
  #  and the point to the left of (x1, x1) is (x0,y0) and the point to the right
  #  of x is (x2, y2)
  #  @param x a real number
  #  @return the value of y
    def quadVal(self, x):
        indexX = self.xValues.indexInSeq(x)
        x1 = self.xValues.get(indexX)
        #obtains the value at the indexX
        x2 = self.xValues.get(indexX + 1)
        y1 = self.yValues.get(indexX)
        y2 = self.yValues.get(indexX + 1)
        x0 = self.xValues.get(indexX - 1)
        y0 = self.yValues.get(indexX-1)
```

```
        y = y1 + ((y2-y0)/(x2-x0)) * (x - x1) + (y2 - 2 * y1 + y0)/ (2 * (x2-x1) ** 2) * ((x - x1) **
            2)
        return y

    ## @brief Calculates the approximation of the value of y at x using regression
    #  @details Uses the polyfit function from numpy to find the best fit
    #  polynomial and then is evaluated at x
    #  @param n an integer number that is the degree of the polynomial
    #  @param x a real number
    #  @return the approximation of the value of y at x
    def npolyVal(self, n, x):
        z = np.polyfit(self.xValues.elements, self.yValues.elements, n)
        p = np.poly1d(z)
        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
```