Assignment 1 Solution

Henry M. 000000000

January 22, 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.

3 Discussion of Test Results

3.1 Problems with Original Code

3.2 Problems with Partner's Code

3.3 Problems with Assignment Specification

Potential problems with the original assignment specification include the following:

- The `add` method is undefined for `i` less than zero, or greater than the length of the sequence.
- The `rm` method is undefined for `i` less than zero, or greater than or equal to the length of the sequence.
- The `set` method is undefined for `i` less than zero, or greater than or equal to the length of the sequence.

- The `get` method is undefined for `i` less than zero, or greater than or equal to the length of the sequence.
- The `indexInSeq` method is undefined for `v` not lying between any of the data points. The specification also does not say what to do when the condition is satisfied by more than one interval of data points.
- The `CurveT` constructor does not say what to do in the case where the file does not exist, or the data is in an incorrect format. Should the constructor check that the `x` values are increasing?
- For `quadVal(x)` where should the 3 data points be located relative to `x`?
- For testing purposes, what is the relative error allowed between the calculated values and the expected values?

4 Answers

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

   For the abstract data type SeqADT the following table classifies each method:

Method	Type
SeqT	constructor
add	mutator
rm	mutator
set	mutator
get	accessor
size	accessor
indexInSeq	accessor

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

   - Advantage - Less work, don't have to rebuild the software to perform the functionality
   - Advantage - Do not have unit test software
   - Disadvantage - have to learn the interface for numpy

- Disadvantage - at the will of the package/library maintainer for bug fixes

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?

   Differences between SeqT and Python's list type:

   - Python's lists have a more natural syntax and greater functionality. The SeqT type is essentially a wrapper for list, that provides less functionality.

   Benefits python list provide over SeqT:

   - Python list provided more functionality of SeqT and is fully tested.
   - The syntax for Python list's is more natural and expressive than using the interface for SeqT.

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

   Simple solution would be to sort the values internally so they are $x_i < x_{i+1}$, otherwise you would have to perform a search for nearby values for indexInSeq.

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

   Only if the input function (set of points) describe a line, otherwise they will not. This is because `linVal` *interpolates* between *2* points and `npolyVal` does a *regression* with *all* the data points.

E Code for SeqADT.py

```
## @file SeqADT.py
#  @author Henry Madej
#  @brief Provides the Abstract Data Type (ADT) for representing Sequences
#  @date 06/01/2018


## @brief An abstract data type that represents a Sequence
class SeqT():
  ## @brief SeqT constructor
  #  @details Initializes a SeqT object with the empty sequence
  def __init__(self):
    self.seq = []

  ## @brief add, adds values to/within existing sequence
  #         or immediately after the last entry in the existing sequence.
  #  @param i The index at which v will be inserted, if i >= length of the
  #         sequence v will be inserted at the end of the sequence
  #  @param v The real number to be inserted into the sequence
  def add(self, i, v):
    self.seq.insert(i, v)

  ## @brief rm, deletes a value within a sequence at index i
  #  @param i deletes the value at index i shrinking the length of the sequence
  #         by 1
  def rm(self, i):
    del self.seq[i]

  ## @brief set, sets the value at index i in the sequence to value v
  #  @param i the index of the value to be mutated
  #  @param v the new value of the index i of the sequence
  def set(self, i, v):
    self.seq[i] = v

  ## @brief get, gets the value of the sequence at index i
  #  @param i the index of the value to be retrieved from the sequence
  #  @return the value of the sequence at index i
  def get(self, i):
    return self.seq[i]

  ## @brief size, returns the size of the sequence
  #  @return the current size of the sequence
  def size(self):
    return len(self.seq)

  ## @brief indexInSeq, returns the index of the value v such that
  #         sequence[i] <= v <= sequence[i+1] if such a value exists otherwise
  #         returns -1
  #  @param v the value to be checked if in sequence
  #  @return the index of the value if it is in the sequence and satisfies
  #            sequence[i] <= v <= sequence[i+1]
  def indexInSeq(self, v):
    index = -1
    for i in range(self.size()-1):
      if self.seq[i] <= v and v <= self.seq[i+1]:
        index = i
        break;
    return index
```

F Code for CurveADT.py

```
## @file CurveADT.py
#  @author Henry Madej
#  @brief Provides the Abstract Data Type (ADT) for representing Curves
#  @date 07/01/2018

import numpy as np
import re
from SeqADT import *

def __get_points_from_sequence__(sequence, position, number_of_points):
  values = []
  for i in range(number_of_points):
    values.append(sequence.get(position+i))
  return values

## @brief An abstract data type that represents a Curve
class CurveT():
  __PATTERN__ = re.compile('(\d+.\d*|\d*.\d+|\d+),\s(\d+.\d*|\d*.\d+|\d+)')

  ## @brief CurveT constructor
  #  @details Initializes a CurveT object from a set of points provided in file,
  #   filename
  def __init__(self, filename):
    self.x_sequence = SeqT()
    self.y_sequence = SeqT()
    self.__read_file__(filename)


  def __read_file__(self, filename):
    file = open(filename, 'r')
    index = 0
    for line in file:
      match = self.__PATTERN__.match(line)
      if match:
        self.x_sequence.add(index, float(match.group(1)))
        self.y_sequence.add(index, float(match.group(2)))
        index += 1
    file.close()


  ## @brief linVal linear interpolation of the curve to approximate output y
  #         given some input x
  #  @param x the input value of x to the line to predict the output y
  #  @return y the output of the line at input x
  def linVal(self, x):
    if self.x_sequence.size() < 2:
        raise ValueError("Not enough points for linear interpolation")
    position_in_sequence = self.x_sequence.indexInSeq(x)
    x_ = __get_points_from_sequence__(self.x_sequence, position_in_sequence, 2)
    y_ = __get_points_from_sequence__(self.y_sequence, position_in_sequence, 2)
    denominator = x_[1] - x_[0]
    y = ((y_[1] - y_[0]) / denominator) * (x - x_[0]) + y_[0]
    return y

  ## @brief quadVal quadratic interpolation of the curve to approximate output
  #         y given some input x
  #  @param x the input value of x to the quadratic to predict output y
  #  @return y the output of the quadratic at input x
  def quadVal(self, x):
    if self.x_sequence.size() < 3:
      raise ValueError("Not enough points for quadratic interpolation")
    position_in_sequence = self.x_sequence.indexInSeq(x)
    xs = __get_points_from_sequence__(self.x_sequence, position_in_sequence, 3)
    ys = __get_points_from_sequence__(self.y_sequence, position_in_sequence, 3)
    b =  ((ys[2] - ys[0]) / (xs[2] - xs[0])) * (x - xs[1])
    a = ((ys[2] - 2 * ys[1] + ys[0]) / (2 * (xs[2] - xs[1]) ** 2)) * ((x - xs[1]) ** 2)
    return a + b + ys[1]


  ## @brief Regression of n degree polynomial to approximate output y given some
  #         input x
  #  @param n the highest degree of the polynomial to be fitted
  #  @param x the input value of x to the polynomial to predict output y
  #  @return y the output of the polynomial at input x
  def npolyVal(self, n, x):
    xs = np.array(__get_points_from_sequence__(self.x_sequence, 0, self.x_sequence.size()))
```

```
ys = np.array(__get_points_from_sequence__(self.y_sequence, 0, self.y_sequence.size()))
estimated_polynomial_function = np.poly1d(np.polyfit(xs, ys, n))
return estimated_polynomial_function(x)
```

G Code for testSeqs.py

```
from SeqADT import SeqT as SeqT
from CurveADT import CurveT as CurveT

def assertionEqual(test, result, name):
  if test == result:
    print("Test passed, %s == %s, %s " % (test, result, name))
  else:
    print("Test failed, %s != %s, %s " % (test, result, name))


def assertionApproximatelyEqual(test, result, error, name):
  if abs(test - result) < error:
    print("Test passed, Actual: %s  Approximate: %s, %s " % (test, result, name))
  else:
    print("Test failed, Actual: %s  Approximate: %s, %s " % (test, result, name))

def testSeq1():
  seq = SeqT()
  # A constructor (SeqT()) that takes no arguments and creates an object
  # whose state consists of an empty sequence
  assertionEqual(seq.size(), 0, "Empty sequence")

def testSeq2():
  seq = SeqT()
  # add immediately after the last entry in the existing sequence.
  seq.add(0, 2.0)
  assertionEqual(seq.seq[0], 2.0, "Add to an empty sequence")

def testSeq3():
  seq = SeqT()
  # add immediately after the last entry in the existing sequence.
  seq.add(52, 2.0)
  assertionEqual(seq.seq[0], 2.0, "Add to an empty sequence large index")

def testSeq4():
  seq = SeqT()
  # add immediately after the last entry in the existing sequence.
  seq.add(1, 2.0)
  seq.add(1, 3.0)
  seq.add(1, 4.0)
  seq.add(1, 5.0)
  # Values can only be added within the existing sequence
  assertionEqual(seq.seq[1], 5.0, "Add within sequence")

def testSeq5():
  seq = SeqT()
  seq.add(0, 1)
  seq.add(1, 2)
  seq.add(2, 3)
  seq.add(3, 4)
  seq.rm(0)
  #A call to this method modifies the sequence so that the entry at index i
  # is removed. The length of the list will decrease by 1.
  assertionEqual(seq.seq[0], 2, "removed 0th element")
  assertionEqual(seq.size(), 3, "size decreased by 1")

def testSeq6():
  seq = SeqT()
  # undefined behaviour
  seq.rm(0)

def testSeq7():
  seq = SeqT()
  # undefined behaviour
  seq.set(0, 1)

def testSeq8():
  seq = SeqT()
  seq.add(0, 9)
  seq.set(0, 1)
  assertionEqual(seq.seq[0], 1, "mutated item at index 0")

def testSeq9():
  seq = SeqT()
  # undefined behaviour
  seq.get(0)
```

```
def testSeq10():
  seq = SeqT()
  seq.add(0, 1)
  seq.add(1, 2)
  seq.add(2, 3)
  assertionEqual(seq.get(2), 3, "got correct item")


def testSeq11():
  seq = SeqT()
  assertionEqual(seq.size(), 0, "Empty sequence, size 0")

def testSeq12():
  seq = SeqT()
  seq.add(0, 1)
  seq.add(1, 2)
  seq.add(2, 3)
  assertionEqual(seq.size(), 3, "Non empty sequence, correct size")

def testSeq13():
  seq = SeqT()
  seq.add(0, 1.0)
  seq.add(3, 2.0)
  seq.add(4, 5.0)
  seq.rm(2)
  assertionEqual(seq.size(), 2, "Size correct after removal")


def testSeq14():
  seq = SeqT()
  seq.add(0, 1.0)
  seq.add(3, 2.0)
  seq.add(4, 5.0)
  assertionEqual(seq.indexInSeq(2.0), 0, "Correct index")

def testSeq14():
  seq = SeqT()
  seq.add(0, 1.0)
  seq.add(3, 2.0)
  seq.add(4, 5.0)
  assertionEqual(seq.indexInSeq(7.0), -1.0, "Correct index")

def testCurve1():
  curve = CurveT("./src/linear")
  assertionApproximatelyEqual(4.85, curve.linVal(2.5), 0.000005, "Correct approximation")

def testCurve2():
  curve = CurveT("./src/quad")
  assertionApproximatelyEqual(31.575, curve.quadVal(2.5), 0.000005, "Correct approximation")

def testCurve3():
  curve = CurveT("./src/points")
  try:
    curve.linVal(0)
    print("Test failed: No exception! divion by zero!")
  except Exception as err:
    if type(err) == ZeroDivisionError:
      print("Test passed: Exception: {0}".format(err))
    else:
      print("Test failed: Wrong Exception: {0} should be ZeroDivisionError".format(err))

def testCurve4():
    curve = CurveT("./src/point")
    try:
      curve.linVal(12)
      print("Test failed: No exception! not enough points for quadratic interpolation")
    except Exception as err:
      if type(err) == ValueError:
        print("Test passed: {0}".format(err))
      else:
        print("Test failed: Wrong Exception: {0} should be {1}".format(err, ValueError))


def testCurve5():
  curve = CurveT("./src/points")
  try:
    curve.quadVal(12)
    print("Test failed: No exception! not enough points for quadratic interpolation")
  except Exception as err:
```

```
    if type(err) == ValueError:
      print("Test passed: {0}".format(err))
    else:
      print("Test failed: Wrong Exception: {0} should be {1}".format(err, ValueError))

def testCurve6():
  curve = CurveT("./src/vertical")
  try:
    curve.quadVal(12)
    print("Test failed: No exception! division by zero!")
  except Exception as err:
    if type(err) == ZeroDivisionError:
      print("Test passed: {0}".format(err))
    else:
      print("Test failed: Wrong Exception: {0} should be {1}".format(err, ZeroDivisionError))

def testCurve6():
  curve = CurveT("./src/vertical")
  try:
    curve.linVal(12)
    print("Test failed: No exception! division by zero!")
  except Exception as err:
    if type(err) == ZeroDivisionError:
      print("Test passed: {0}".format(err))
    else:
      print("Test failed: Wrong Exception: {0} should be {1}".format(err, ZeroDivisionError))


def test():
  testSeq1()
  testSeq2()
  testSeq3()
  testSeq4()
  testSeq5()
  #testSeq6() undefined behaviour
  #testSeq7() undefined behaviour
  testSeq8()
  #testSeq9() undefined behaviour
  testSeq10()
  testSeq11()
  testSeq12()
  testSeq13()
  testSeq14()
  testCurve1()
  testCurve2()
  testCurve3()
  testCurve4()
  testCurve5()
  testCurve6()


test()
```

H Code for Partner's SeqADT.py

```
## @file SeqADT.py
#  @author Partner
```

I Code for Partner's CurveADT.py

```
## @file CurveADT.py
#  @author Partner
```

J Makefile

```
PY = python
PYFLAGS =
DOC = doxygen
DOCFLAGS =
DOCCONFIG = seqdoc

SRC = src/testSeqs.py

.PHONY: all test doc clean

test:
	$(PY) $(PYFLAGS) $(SRC)

doc:
	$(DOC) $(DOCFLAGS) $(DOCCONFIG)
	cd latex && $(MAKE)

all: test doc

clean:
	rm -rf html
	rm -rf latex
```