Assignment 1 Report

Haoyuan(Harry) Fu fuh6

January 31, 2018

Itroductory blurb.

1 Testing of the Original Program

In my testSeqs.py testing file, I used at least three good cases and two bad cases for every function in my two modules which are SeqADT.py and CurveADT.py. The Good cases are recalling every function with inputs in expected range, bad cases are inputs without the right range. For linVal and quadVal, these two functions that need input test txt file, I created a temporary txt file in my test file, it makes user dont need to create test txt file every time. For the testing of the original program, after I do make test testSeqs.py, it gives me my expected output which is all the test of functions are passed. After I checked my test file again, I dont think I still have any uncovered problem I need to solve.

2 Results of Testing Partner's Code

At the first time I ran my test file on my partners code I got all the functions test failed and raised a type error. But I realized that there might be some problem with my test file which I will explain later in the discussion part. After I changed my test file, I got all the SeqADTs functions are passed individually but stopped at CurveADT and generate the same type error.

3 Discussion of Test Results

3.1 Problems with Original Code

The original code passed every individual function test and worked correctly, so there is no any problem with the original code. But during my coding process, there are several problems with assignment specification which I will talk about later.

3.2 Problems with Partner's Code

During the first time I ran my test file on my partners code, I got almost all the individual functions in SeqADT failed, but after I checked my test case and my partners code, I found that I my test file I used some try statements for bad cases and raise an assertion error and use except statement to pass IndexError and print test failed except AssertionError, ValueError and IndexError. Thats because I had already raised IndexError in every individual function in my SeqADT to make sure all the functions can work correctly. But my partners code did not raise the IndexError and cause a test failed. After I commented all the bad cases in my test file, it generated all the functions in SeqADT are passed but still have TypeError in my partners CurveADT module. After I checked my partners code, I found that he didnt raised a ValueError in his indexInSeq function. It causes that when he want to sign the value of indexInSeq to a variable i, this variable became a Nonetype, and Nonetype cant do any addition with an integer and then generate the TypeError. I think this is a possible problem in my partners code.

3.3 Problems with Assignment Specification

I found there are two problems with assignment specification during my coding process. One of them is all the exceptions of functions are not mentioned in the assignment. For example, the add(i) function in SeqADT module, if the input value i is a negative number the whole function will generate an error, so I raised exception for every function in my assignment to make sure all the functions can work correctly. The second problem is for the linval and quadVal function, the description of the value of x1 and x2 are not clearly defined. We know that there should be at least three data for points, at least on data points to the left of x and at least one points to the right, but we have no idea of which side would determine the result of y. this would cause several different methods to create this function.

4 Answers

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

   Answer
   Constructor: SeqT(), CurveT()

Accessor: get(), size(), indexInSeq()
Mutator: add(), rm(), set(), linVal(), quadVal(), npolyVal()

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

   Answer
   The advantages of using numpy is that numpy has lots of built-in function that can be directly used to write a complex function. For example, in our assignment, the npolyVal is hard to solve if we are using the python built-in functions, but since we import numpy, we can just use polyfit and poly1d to solve it directly. So, using the numpy is an efficient way to solve some complex questions or equations. The disadvantage of using numpy is its not portable. Since programs would be run on many different computers, and every user has their different operating system, downloading the source file of external library could be an inefficient problem. Also, if other programmer hasnt used this external library before, its hard to them to figure out how this function work. This will cause a time wasting.

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?

   Answer
   The difference between SeqT and Pythons in-built list is that for SeqT, we used a constructor to make it only use the methods that are defined in the class CurveT. But basically, the SeqT is based on Pythons built-in list function, so except it is a kind of private method its still do what the pythons list can do. The benefit that the pythons list is it is a built-in function. We can use this function anywhere anytime we want to. It is much easier than defining a class to create a list. Also, it is more intuitive than SeqT wrapper for list. The SeqT class was built to achieve the module CurveADT.

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

   Answer
   If the assumption does not apply, it means that the input values would have no orders, so it will become an unsorted list. We have to pay more attention and time to sort this list to a sorted list to make sure our program run correctly. Or we have to find a searching algorithm to make sure we can find the exact value we want.

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

Answer
The linVal(x) function and npolyVal(n,x) function will not generate same value if the x is equal. The linVal is calculated by only two reference values but npolyVal is calculated the whole list of value. So npolyVal is more accurate than linVal.

E Code for SeqADT.py

```
## @file SeqADT.py
#  @author Haoyuan(Harry) Fu
#  @brief Provides the SeqT ADT class for representing sequences
#  @date 1/22/2018


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

    ## @brief SeqT constructor
    #  @details Initializes a SeqT object
    #  @param self The object pointer
    def __init__(self):
        self.seq = []

    ## @brief Adds a value into the sequence at the given index
    #  @details The value can only be added within the existing sequence
    #           or immediately after the last entry in the existing sequence
    #  @param self The object pointer
    #  @param i The index where the value should be added within the sequence
    #  @param v The value to be added to the sequence
    #  @exception IndexError raises for illegal index
    def add(self, i, v):
        if i < 0 or i > len(self.seq):
            raise IndexError('illegal index')
        self.seq.insert(i, v)

    ## @brief Removes a value from the sequence
    #  @details The index can only be within the existing sequence
    #  @param self The object pointer
    #  @param i The index where the value located in the sequence
    #  @exception IndexError raises for illegal index
    def rm(self, i):
        if i < 0 or i > len(self.seq) - 1:
            raise IndexError('illegal index')
        self.seq.pop(i)

    ## @brief Modifies a value at the given index of the sequence
    #  @details The index can only be within the existing sequence
    #  @param self The object pointer
    #  @param i The index where the value located in the sequence
    #  @param v The value used to replace the old one
    #  @exception IndexError raises for illegal index
    def set(self, i, v):
        if i < 0 or i > len(self.seq) - 1:
            raise IndexError('illegal index')
        self.seq[i] = v

    ## @brief Takes a value at the given index of the sequence
    #  @details The index can only be within the existing sequence
    #  @param self The object pointer
    #  @param i The index where the value located in the sequence
    #  @return The value at the given index of the sequence
    #  @exception IndexError raises for illegal index
    def get(self, i):
        if i < 0 or i > len(self.seq) - 1:
            raise IndexError('illegal index')
        return self.seq[i]

    ## @brief Retrieves the length of the sequence
    #  @param self The object pointer
    #  @return The length of the sequence
    def size(self):
        return len(self.seq)

    ## @brief Finds the index of the value approximately
    #  @details For the sequence object s, returns the index i
    #           such that s.get(i) <= v <= s.get(i+1)
    #  @param self The object pointer
    #  @param v The value to be found in the sequence
    #  @return The index of the value
    #  @exception ValueError raises for illegal value
    def indexInSeq(self, v):
        for i, _v_ in enumerate(self.seq[:-1]):
            if _v_ <= v and v <= self.seq[i + 1]:
                return i
```

```
raise ValueError('illegal value')
```

F Code for CurveADT.py

```
## @file CurveADT.py
#  @author Haoyuan(Harry) Fu
#  @brief Provides the CurveT ADT class for representing curves
#  @date 1/22/2018

from SeqADT import *
import numpy as np

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

    ## @brief CurveT constructor
    #  @details Initializes a CurveT object using a given input data file
    #           which should have two columns of data, with each data
    #           entry in a row separated by a comma and a space, and
    #           each row separated by a newline
    #  @param self The object pointer
    #  @param s The filename of the input data file
    def __init__(self, s):
        self.seqX = SeqT()
        self.seqY = SeqT()
        i = 0
        with open(s, 'r') as f:
            for row in f:
                eles = row.split(', ')
                if len(eles) == 2:
                    self.seqX.add(i, float(eles[0]))
                    self.seqY.add(i, float(eles[1]))
                    i += 1

    ## @brief Does Linearly interpolation to get a y at x
    #  @details There should be at least two data points and
    #           there should be at least one data point to
    #           the left of x (overlap allowed) and also
    #           at least one data point to the right of x
    #           (overlap allowed)
    #  @param self The object pointer
    #  @param x The given x
    #  @return The interpolated y
    #  @exception ValueError raises for illegal value
    def linVal(self, x):
        try:
            i1 = self.seqX.indexInSeq(x)
        except IndexError:
            raise ValueError('illegal x')
        i2 = i1 + 1
        x1, y1 = self.seqX.get(i1), self.seqY.get(i1)
        x2, y2 = self.seqX.get(i2), self.seqY.get(i2)
        y = (y2 - y1) * (x - x1) / (x2 - x1) + y1
        return y

    ## @brief Does quadratically interpolation to get a y at x
    #  @details There should be at least three data points and
    #           there should be at least one data point to
    #           the left of x (overlap allowed) and also
    #           at least one data point to the right of x
    #           (overlap allowed)
    #  @param self The object pointer
    #  @param x The given x
    #  @return The interpolated y
    #  @exception ValueError raises for illegal value or insufficient data points
    def quadVal(self, x):
        try:
            i1 = self.seqX.indexInSeq(x)
        except IndexError:
            raise ValueError('illegal x')
        if i1 == 0:
            if self.seqX.size() < 3:
                raise ValueError('insufficient data points')
            i0, i1, i2 = 0, 1, 2
        else:
            i0, i2 = i1 - 1, i1 + 1
        x0, y0 = self.seqX.get(i0), self.seqY.get(i0)
        x1, y1 = self.seqX.get(i1), self.seqY.get(i1)
        x2, y2 = self.seqX.get(i2), self.seqY.get(i2)
    y = y1 + (y2 - y0) * (x - x1) / (x2 - x0) + (y2 - 2 * y1 + y0) * (x - x1)**2 / (2 * (x2 -
        x1)**2)
    return y

## @brief Uses least squares polynomial fitting to find a y at x
#  @details The ployfit function of numpy is wrapped inside
#  @param self The object pointer
#  @param n The degree of polynomial
#  @param x The given x
#  @return The interpolated y
#  @exception Exception raises by numpy.polyfit if any
#  @the numpy command I found here: https://docs.scipy.org/doc/numpy-1.13.0/reference/index.html
def npolyVal(self, n, x):
    xs = []
    ys = []
    for i in range(self.seqX.size()):
        xs.append(self.seqX.get(i))
        ys.append(self.seqY.get(i))
    zs = np.polyfit(xs, ys, n)
    p = np.poly1d(zs)
    y = p(x)
    return y
```

G Code for testSeqs.py

```
## @file testSeqs.py
#  @author Haoyuan(Harry) Fu
#  @brief Provides the testcase suite for SeqADT.py and CurveADT.py
#  @date 1/22/2018

from SeqADT import SeqT
from CurveADT import CurveT

# for testing floating point equality after arithmetic
# from Python development documentation:
#   https://www.python.org/dev/peps/pep-0485/#proposed-implementation
#       I used this web to check my polynomial data: https://arachnoid.com/polysolve/
def isClose(a, b, rel_tol = 1e-09, abs_tol = 0.0):
    return abs(a - b) <= max(rel_tol * max(abs(a), abs(b)), abs_tol)

# @brief Tests the add method of the SeqT ADT class
# @details Checks if the SeqT.add method works with
#          the assumption that the SeqT.get method works fine
def test_SeqT_add():
    try:
        seq = SeqT()
        seq.add(0, 10.0)
        assert seq.get(0) == 10.0
        seq.add(0, 20.0)
        assert (seq.get(0) == 20.0 and seq.get(1) == 10.0)
        seq.add(1, 30.0)
        assert (seq.get(0) == 20.0 and seq.get(1) == 30.0 and
                seq.get(2) == 10.0)
        seq.add(3, 40.0)
        assert (seq.get(0) == 20.0 and seq.get(1) == 30.0 and
                seq.get(2) == 10.0 and seq.get(3) == 40.0)
        try:
            seq.add(-1, 50.0)
            raise AssertionError
        except IndexError:
            pass
        try:
            seq.add(5, 50.0)
            raise AssertionError
        except IndexError:
            pass
        print('test of SeqT.add PASSED.')
    except (AssertionError, ValueError, IndexError):
        print('test of SeqT.add FAILED.')


# @brief Tests the rm method of the SeqT ADT class
# @details Checks if the SeqT.rm method works with
#          the assumption that the SeqT.get and SeqT.add method works fine
def test_SeqT_rm():
    try:
        seq = SeqT()
        seq.add(0, 10.0)
        seq.add(1, 20.0)
        seq.add(2, 30.0)
        seq.add(3, 40.0)
        seq.rm(0)
        seq.rm(2)
        assert (seq.get(0) == 20.0 and seq.get(1) == 30.0)
        try:
            seq.rm(-1)
            raise AssertionError
        except IndexError:
            pass
        try:
            seq.rm(2)
            raise AssertionError
        except IndexError:
            pass
        print('test of SeqT.rm PASSED.')
    except (AssertionError, ValueError, IndexError):
        print('test of SeqT.rm FAILED.')


# @brief Tests the set method of the SeqT ADT class
# @details Checks if the SeqT.set method works with
#           the assumption that the SeqT.get and SeqT.add method works fine
def test_SeqT_set():
    try:
        seq = SeqT()
        seq.add(0, 10.0)
        seq.add(1, 20.0)
        seq.add(2, 30.0)
        seq.add(3, 40.0)
        seq.set(0, 50.0)
        seq.set(3, 60.0)
        assert (seq.get(0) == 50.0 and seq.get(3) == 60.0)
        try:
            seq.set(-1, 70.0)
            raise AssertionError
        except IndexError:
            pass
        try:
            seq.set(4, 80.0)
            raise AssertionError
        except IndexError:
            pass
        print('test of SeqT.set PASSED.')
    except (AssertionError, ValueError, IndexError):
        print('test of SeqT.set FAILED.')


# @brief Tests the get method of the SeqT ADT class
# @details Checks if the SeqT.get method works with
#           the assumption that the SeqT.add method works fine
def test_SeqT_get():
    try:
        seq = SeqT()
        try:
            seq.get(0)
            raise AssertionError
        except IndexError:
            pass
        seq.add(0, 10.0)
        seq.add(1, 20.0)
        seq.add(2, 30.0)
        seq.add(3, 40.0)
        assert (seq.get(0) == 10.0 and seq.get(3) == 40.0)
        try:
            seq.get(-1)
            raise AssertionError
        except IndexError:
            pass
        try:
            seq.get(4)
            raise AssertionError
        except IndexError:
            pass
        print('test of SeqT.get PASSED.')
    except (AssertionError, ValueError, IndexError):
        print('test of SeqT.get FAILED.')


# @brief Tests the size method of the SeqT ADT class
# @details Checks if the SeqT.size method works with
#           the assumption that the SeqT.add amd SeqT.rm method works fine
def test_SeqT_size():
    try:
        seq = SeqT()
        assert seq.size() == 0
        seq.add(0, 10.0)
        seq.add(1, 20.0)
        seq.add(2, 30.0)
        seq.add(3, 40.0)
        assert seq.size() == 4
        seq.rm(0)
        seq.rm(0)
        seq.rm(0)
        seq.rm(0)
        print('test of SeqT.size PASSED.')
    except (AssertionError, ValueError, IndexError):
        print('test of SeqT.size FAILED.')


# @brief Tests the indexInSeq method of the SeqT ADT class
# @details Checks if the SeqT.indexInSeq method works with
#           the assumption that the SeqT.add method works fine
def test_SeqT_indexInSeq():
    try:
        seq = SeqT()
        seq.add(0, 10.0)
        seq.add(1, 20.0)
        seq.add(2, 30.0)
        seq.add(3, 40.0)
        assert (seq.indexInSeq(15.0) == 0 and seq.indexInSeq(25.0) == 1 and
                seq.indexInSeq(35.0) == 2)
        assert (seq.indexInSeq(10.000001) == 0 and seq.indexInSeq(20.000001) == 1 and
                seq.indexInSeq(30.000001) == 2)
        assert (seq.indexInSeq(19.999999) == 0 and seq.indexInSeq(29.999999) == 1 and
                seq.indexInSeq(39.000000) == 2)
        try:
            seq.indexInSeq(0.0)
            raise AssertionError
        except ValueError:
            pass
        try:
            seq.indexInSeq(50.0)
            raise AssertionError
        except ValueError:
            pass
        print('test of SeqT.indexInSeq PASSED.')
    except (AssertionError, ValueError, IndexError):
        print('test of SeqT.indexInSeq FAILED.')


# @brief Tests the linVal method of the CurveT ADT class
# @details Checks if the CurveT.linVal method works with
#          a temporary date file generated on the fly
def test_CurveT_linVal():
    with open('tmp_file', 'w') as f:
        f.write('\n'.join([
            '1.0, 10.0',
            '2.0, 20.0',
            '3.0, 30.0',
            '4.0, 40.0',
            '5.0, 50.0'
        ]))
    try:
        curve = CurveT('tmp_file')
        assert (isClose(curve.linVal(1.5), 15.0) and
                isClose(curve.linVal(2.5), 25.0) and
                isClose(curve.linVal(3.5), 35.0))
        assert (isClose(curve.linVal(1.001), 10.01) and
                isClose(curve.linVal(2.001), 20.01) and
                isClose(curve.linVal(3.001), 30.01))
        assert (isClose(curve.linVal(1.999), 19.99) and
                isClose(curve.linVal(2.999), 29.99) and
                isClose(curve.linVal(3.999), 39.99))
        try:
            curve.linVal(0.5)
            raise AssertionError
        except ValueError:
            pass
        try:
            curve.linVal(5.5)
            raise AssertionError
        except ValueError:
            pass
        print('test of CurveT.linVal PASSED.')
    except (AssertionError, ValueError, IndexError):
        print('test of CurveT.linVal FAILED.')


# @brief Tests the quadVal method of the CurveT ADT class
# @details Checks if the CurveT.quadVal method works with
#          a temporary date file generated on the fly
def test_CurveT_quadVal():
    with open('tmp_file', 'w') as f:
        f.write('\n'.join([
            '1.0, 5.0',
            '2.0, 20.0',
            '3.0, 25.0',
            '4.0, 40.0',
            '5.0, 41.0'
        ]))
    try:
        curve = CurveT('tmp_file')
        assert (isClose(curve.quadVal(1.5), 13.75) and
                isClose(curve.quadVal(2.5), 23.75) and
                isClose(curve.quadVal(3.5), 31.25))
        assert (isClose(curve.quadVal(1.01), 5.1995) and
                isClose(curve.quadVal(2.01), 20.0995) and
                isClose(curve.quadVal(3.01), 25.1005))
        assert (isClose(curve.quadVal(1.99), 19.8995) and
                isClose(curve.quadVal(2.99), 24.9995) and
                isClose(curve.quadVal(3.99), 39.8005))
        try:
            curve.quadVal(0.5)
            raise AssertionError
        except ValueError:
            pass
        try:
            curve.quadVal(5.5)
            raise AssertionError
        except ValueError:
            pass
        print('test of CurveT.quadVal PASSED.')
    except (AssertionError, ValueError, IndexError):
        print('test of CurveT.quadVal FAILED.')

    with open('tmp_file', 'w') as f:
        f.write('\n'.join([
            '1.0, 5.0',
            '2.0, 20.0'
        ]))
    try:
        curve = CurveT('tmp_file')
        try:
            curve.quadVal(1.5)
            raise AssertionError
        except ValueError as e:
            pass
        print('test of CurveT.quadVal PASSED.')
    except (AssertionError, ValueError, IndexError):
        print('test of CurveT.quadVal FAILED.')


# @brief Tests the npolyVal method of the CurveT ADT class
# @details Checks if the CurveT.npolyVal method works with
#          a temporary date file generated on the fly
def test_CurveT_npolyVal():
    with open('tmp_file', 'w') as f:
        f.write('\n'.join([
            '1.0, 10.0',
            '2.0, 20.0',
            '3.0, 50.0',
            '4.0, 30.0',
            '5.0, 10.0'
        ]))
    try:
        curve = CurveT('tmp_file')
        assert (isClose(curve.npolyVal(1, 1.5), 22.5) and
                isClose(curve.npolyVal(1, 2.5), 23.5) and
                isClose(curve.npolyVal(1, 3.5), 24.5))
        assert (isClose(curve.npolyVal(2, 1.5), 20.5357143) and
                isClose(curve.npolyVal(2, 2.5), 37.25) and
                isClose(curve.npolyVal(2, 3.5), 38.25))
        assert (isClose(curve.npolyVal(3, 1.5), 17.66071429) and
                isClose(curve.npolyVal(3, 2.5), 34.625) and
                isClose(curve.npolyVal(3, 3.5), 40.875))
        print('test of CurveT.npolyVal PASSED.')
    except (AssertionError, ValueError, IndexError):
        print('test of CurveT.npolyVal FAILED.')


test_SeqT_add()
test_SeqT_rm()
test_SeqT_set()
test_SeqT_get()
test_SeqT_size()
test_SeqT_indexInSeq()

test_CurveT_linVal()
test_CurveT_quadVal()
test_CurveT_npolyVal()
```

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
```