Assignment 2 Report

Harry(Haoyuan) Fu fuh6

March 3, 2018

Report for Assignment2, analysis of my partner's and mine assignment result.

1 Testing of the Original Program

In the testAll.py file, it used many different sets of numbers to focus on testing three modules which are CurveADT, Data and SeqServices. For the reult of testing these three modules, CurveADT has 39 stmts and has 0 miss, Data has 27 stmts and has 0 miss and SeqServices has 15 stmts and 0 miss. According to the test coverage of these three modules, it shows that these modules passed testing and have a logical output. In the description of Assignment2, it says that we do not need to do the test coverage for Load and Plot module, so the test coverage for these two modules are 0 percent. Because we do not need to execute all of codes in our modules, so the test coverage for Exceptions.py is 80 percent which has 6 misses and testAll.py has 99 percent coverage which has 2 misses.

2 Results of Testing Partner's Code

The test coverage for my partner's three modules are: 80 percent for CurveADT.py, 37 percent for Data.py and 93 percent for SeqServices.py. After the test coverage, terminal showed all the failures during the testing. Most failures are in Data.py and all of them are TypeError. Other than there are some AssertionError in testAll.py. I will do some explanation of my partner's result's explanation in the later part.

3 Discussion of Test Results

3.1 Problems with Original Code

From my perspective, the total coverage for my code is 86 percent and get 100 percent in three main modules. So my test case worked well on my code. The reason of I got some misses and the coverage is not totally 100 percent is there are some commands do not need to be actually implemented and tested, so my test case may not covered all the exceptions but the main part of this program works well.

3.2 Problems with Partner's Code

1. In my original Data.py, it passes all the vatiables as staticmethod but my partner didn't make them as staticmethod. As long as my test case doing Data_init() in every test function for Data.py, it will show the TypeError again and again.
2. In my partner's CurveADT.py, the constructor __init__ didn't make self.X and self.Y as list but in my code I made them as list. When my test case trying to call the constructor it will raise a TypeError in CurveADT.py.
3. Because there are some TypeErrors occurred in Data.py and CurveADT.py, the test_All raised some AsserstionError.
4. Above all, all the errors occurred causes the test coverage could not approach to 100 percent.

3.3 Problems with Assignment Specification

This assignment's specification is much different than assignment1 's specification. The most different thing is assignment 2 used MIS. Using MIS helps to improve quality and correctness of our program. But there are some problem of using the MIS which is we have to spend time to understand the requirements.
For example, the specification for Load module is hard to understand and also not spcific, it doesn't have any output so we couldn't find out what this module actually do by this MIS form.
Also during the coding process, I was confused about the definition of environment variable. The definition of environment variable should be more specific in this specification.

4 Answers

1. What is the mathematical specification of the `SeqServices` access program isInBounds(X, x) if the assumption that X is ascending is removed?

Answer:
If the assumption that X is ascending is removed, which means the X is not necessarily ascending. When isInBounds(X,x) called, it will return a value between X[0] and X[-1]. Mathematically, isInBound still picks a number between X[0] and X[-1] if X is not ascending anmore. So remove the assumption of X is ascending will not effect the correctness.

2. How would you modify `CurveADT.py` to support cubic interpolation?

   Answer:
   In my interpQuad function in SeqServices.py, I can simply change the interpQuad(x0, y0, x1, y1, x2, y2, x) to interpCubic(x0, y0, x1, y1, x2, y2, x3, y3, x), I need to provide three points instead of two points to do the cubic interpolation. Then I have to change the basic algorithm to cubic which is change all the **2 to **3. Then we can let interp(X, Y, o, v) in CurveADT.py to call this interpCubic funtion to support cubic interpolation.

3. What is your critique of the CurveADT module's interface. In particular, comment on whether the exported access programs provide an interface that is consistent, essential, general, minimal and opaque.

   Answer:
   In CurveADT, all the function is worked based on the assumption which is isInBounds is true. This determines if a given value is inside the range of a given sequence. Then we interpolates a linearly a curve with this assumption and then interpolates quadratically a curve with this assumption. All the function is based on the assumption which is x is a value between the sequence. If we want to pick the "edges" value as a point what behaviour should be done to this situation has to be provided.

4. What is your critique of the Data abstract object's interface. In particular, comment on whether the exported access programs provide an interface that is consistent, essential, general, minimal and opaque.

   Answer:
   The variable Z in Data has no domain, we can't get the actual size of Z out of our program. So when we excute Data.add(s,z) and Data.eval(s,z) we have to guess the value of z to make sure the value of z is fit for the program. To make Data more general and flexible it has to provide an accessor.

Users have no idea how to use this Data module,and generality can't predict what task would this module to be done. So we have to make it more general and flexiable by adding a del method to improve the generality.

E Code for CurveADT.py

```
## @file CurveADT.py
#  @author Harry Fu
#  @brief Provides the CurveT ADT class for representing curves
#  @details Provides the CurveT ADT class for representing curves with the assumption:
#           The user will not request function evaluations that would cause the
#           interpolation to select index values outside of where the function
#           is defined.
#  @date 2/14/2018


from SeqServices import *
from Exceptions import *


## @brief An ADT that represent the curve
#  @details With the assumption: The user will not request function
#           evaluations that would cause the interpolation to select
#           index values outside of where the function is defined
class CurveT:
    # the maximum order
    MAX_ORDER = 2
    # the delta x
    DX = 1e-3

    ## @brief CurveT constructor
    #  @details Initializes a CurveT object
    #  @param self The object pointer
    #  @param X The x sequence
    #  @param Y The y sequence
    #  @param i The order
    #  @exception IndepVarNotAscending raises for non-ascending independant variable sequence
    #  @exception SeqSizeMismatch raises for mismatched sequence sizes
    #  @exception InvalidInterpOrder raises for illegal interpolation order
    def __init__(self, X, Y, i):
        if not isAscending(X):
            raise IndepVarNotAscending(X)
        if len(X) != len(Y):
            raise SeqSizeMismatch(X, Y)
        if i < 1 or i > CurveT.MAX_ORDER:
            raise InvalidInterpOrder(i)

        self.X, self.Y = list(X), list(Y)
        self.minx, self.maxx = X[0], X[-1]
        self.o = i
        self.f = lambda v: interp(self.X, self.Y, self.o, v)

    ## @brief Gets the minimum x
    #  @param self The object pointer
    #  @return The minimum x
    def minD(self):
        return self.minx

    ## @brief Gets the maximum x
    #  @param self The object pointer
    #  @return The maximum x
    def maxD(self):
        return self.maxx

    ## @brief Gets the order of curve
    #  @param self The object pointer
    #  @return The order
    def order(self):
        return self.o

    ## @brief Evaluates the y value at the given x
    #  @param self The object pointer
    #  @param x The x value
    #  @return The evaluated y value
    #  @exception OutOfDomain raises for illegal x value
    def eval(self, x):
        self.__sanityCheckX__(x)
        return self.f(x)

    ## @brief Approximates first derivative using forward divided difference
    #  @param self The object pointer
    #  @param x The x value
    #  @return The first derivative
    #  @exception OutOfDomain raises for illegal x value
    def dfdx(self, x):
        self.__sanityCheckX__(x)
        return (self.f(x + CurveT.DX) - self.f(x)) / CurveT.DX

    ## @brief Approximates second derivative using forward divided difference
    #  @param self The object pointer
    #  @param x The x value
    #  @return The second derivative
    #  @exception OutOfDomain raises for illegal x value
    def d2fdx2(self, x):
        self.__sanityCheckX__(x)
        return (self.f(x + 2 * CurveT.DX) - 2 * self.f(x + CurveT.DX) + self.f(x)) / (CurveT.DX ** 2)

    ## @brief Sanity-Checks the x value
    #  @param x The x value
    #  @exception OutOfDomain raises for illegal x value
    def __sanityCheckX__(self, x):
        if not self.minx <= x <= self.maxx:
            raise OutOfDomain(self.minx, x, self.maxx)


## @brief Interpolates the curve
#  @param X The x sequence
#  @param Y The y sequence
#  @param o The order of the curve
#  @param v The v value
#  @return the interpolated y value
def interp(X, Y, o, v):
    i = index(X, v)
    if o == 1:
        return interpLin(X[i], Y[i], X[i + 1], Y[i + 1], v)
    return interpQuad(X[0], Y[0], X[1], Y[1], X[2], Y[2], v) if i == 0 else \
        interpQuad(X[i - 1], Y[i - 1], X[i], Y[i], X[i + 1], Y[i + 1], v)
```

F Code for Data.py

```
## @file Data.py
#  @author Harry Fu
#  @brief Provides the Data manipulation methods
#  @detals Provides the Data manipulation methods with the assumption:
#          Data init() is called before any other access program.
#  @date 2/14/2018

from CurveADT import CurveT
from SeqServices import *
from Exceptions import *


## @brief An class of data manipulation methods
class Data:
    MAX_SIZE = 10

    # sequence of CurveT
    S = []
    # sequence of value
    Z = []

    ## @brief Initializes data module
    @staticmethod
    def init():
        Data.S, Data.Z = [], []

    ## @brief Adds a curve-value pair
    #  @param s The curve component
    #  @param z The value component
    #  @exception Full raises for already-full sequence
    #  @exception IndepVarNotAscending raises for illegal z value
    @staticmethod
    def add(s, z):
        if len(Data.S) == Data.MAX_SIZE:
            raise Full('{} {}'.format(Data.S, Data.MAX_SIZE))
        if len(Data.Z) > 0 and z <= Data.Z[-1]:
            raise IndepVarNotAscending(Data.Z)
        Data.S.extend([s])
        Data.Z.extend([z])


    ## @brief Gets a curve
    #  @param i The index of curve
    #  @return The fetched curve
    #  @exception InvalidIndex raises for illegal index
    @staticmethod
    def getC(i):
        if not 0 <= i < len(Data.S):
            raise InvalidIndex(0, i, len(Data.S))
        return Data.S[i]

    ## @brief Evaluates y value using the given x value and z value
    #  @exception OutOfDomain raises for illegal value
    @staticmethod
    def eval(x, z):
        if not isInBounds(Data.Z, z):
            raise OutOfDomain(Data.Z[0], z, Data.Z[-1])
        j = index(Data.Z, z)
        return interpLin(Data.Z[j], Data.S[j].eval(x), Data.Z[j + 1], Data.S[j + 1].eval(x), z)


    ## @brief Slices the curves
    #  @param x The x value
    #  @param i The order
    #  @return The sliced Curve
    @staticmethod
    def slice(x, i):
        return CurveT(Data.Z, [s.eval(x) for s in Data.S], i)
```

G Code for SeqServices.py

```
## @file SeqServices.py
#  @author Harry Fu
#  @brief Provides the Sequence Services
#  @date 2/14/2018


## @brief Determines if a given sequence is ascending
#  @details Determines if a given sequence if ascending
#           with the assumption: length of X must > 1
#  @param X The sequence
#  @return True if ascending otherwise False
def isAscending(X):
    for x_prev, x_next in zip(X, X[1:]):
        if x_prev > x_next:
            return False
    return True


## @brief Determines if a givenv alue is inside the range of a given sequence
#  @details Determines if a given value is inside the range of a given sequence
#           with the assumption: isAscending(X) is True
#  @param X The sequence
#  @param x The value
#  @return True if in-bound otherwise False
def isInBounds(X, x):
    return X[0] <= x <= X[-1]


## @brief Interpolates linearly a curve
#  @details Interpolates linearly a curve
#           with the assumption: x1 < x2
#           with the assumption: isInBounds([x1, x2], x)
#  @param x1 The x coordinate of the start point of the curve
#  @param y1 The y coordinate of the start point of the curve
#  @param x2 The x coordinate of the end point of the curve
#  @param y2 The y coordinate of the end point of the curve
#  @param x The value
#  @return The interpolated y value
def interpLin(x1, y1, x2, y2, x):
    return (y2 - y1) * (x - x1) / (x2 - x1) + y1


## @brief Interpolates quadratically a curve
#  @detals Interpolates quadratically a curve
#          with the assumption: x0 <= x1 < x2
#          with the assumption: isInBounds([x0, x1, x2], x)
#  @param x0 The x coordinate of the 1st point of the curve
#  @param y0 The y coordinate of the 1st point of the curve
#  @param x1 The x coordinate of the 2nd point of the curve
#  @param y1 The y coordinate of the 2nd point of the curve
#  @param x2 The x coordinate of the 3rd point of the curve
#  @param y2 The y coordinate of the 3rd point of the curve
#  @param x The x value
#  @return The interpolated y value
def interpQuad(x0, y0, x1, y1, x2, y2, x):
    return (
        y1 + (y2 - y0) * (x - x1) / (x2 - x0) +
        (y2 - 2 * y1 + y0) * ((x - x1) ** 2) / (2 * ((x2 - x1) ** 2))
    )


## @brief Finds the proximate index of the given value in the given sequence
#  @details Finds the index of the given value in the given sequence
#           with the assumption: isAscending(X) is True
#           with the assumption: isInBounds(X, x) is True
#           with the assumption: X[0] != X[-1]
#  @param X The sequence
#  @param x The value
#  @return The proximate index of x
def index(X, x):
    for i, (x_prev, x_next) in enumerate(zip(X, X[1:])):
        if x_prev <= x < x_next:
            return i
```

H Code for Plot.py

```
## @file Plot.py
#  @author Harry Fu
#  @brief Provides the Plotting methods
#  @details Provides the Plotting methods with the assumption:
#           For plotting the user will select the number of subdivisions
#           to be small enough that there will not be an interpolation
#           problem with the end points.
#  @date 2/14/2018


from matplotlib import pyplot as plt
from CurveADT import CurveT
from Exceptions import *


## @brief Plots a sequence
#  @param X The x sequence
#  @param Y The y sequence
#  @exception SeqSizeMismatch raises for non-matched sequence size
def PlotSeq(X, Y):
    if len(X) != len(Y):
        raise SeqSizeMismatch()
    plt.scatter(X, Y, color='orange')
    plt.show()


## @brief Plots a curve
#  @param c The curve
#  @param n The number of points
def PlotCurve(c, n):
    X = []
    Y = []
    step = (c.maxD() - c.minD()) / (n - 1)
    if c.order() == 1:
        x_start = c.minD()
        for i in range(n - 1):
            X.append(x_start)
            Y.append(c.eval(x_start))
            x_start += step
    else:
        x_start = c.minD() + step
        for i in range(n - 2):
            X.append(x_start)
            Y.append(c.eval(x_start))
            x_start += step
    plt.plot(X, Y, color='green')
    plt.show()
```

I Code for Load.py

```
## @file Load.py
#  @author Harry Fu
#  @brief Provides the Plotting methods
#  @details Provides the Plotting methods with the assumption:
#           The input file will match the given specification.
#  @date 2/14/2018

from Data import Data
from CurveADT import CurveT


## @brief Reads data from file
#  @param s The filename of file
def Load(s):
    Data.init()
    with open(s, 'r') as f:
        row = next(f)
        row = list(map(float, row.split(',')))
        Z = row
        n = len(Z)
        row = next(f)
        row = list(map(int, row.split(',')))
        O = row
        X_Y = [[] for _ in range(n + n)]
        for row in f:
            row = [e.strip() for e in row.split(',')]
            for i, e in enumerate(row):
                if e:
                    X_Y[i].append(float(e))
        for i, o in enumerate(O):
            Data.add(CurveT(X_Y[i + i], X_Y[i + i + 1], o), Z[i])
```

J Code for Partner's CurveADT.py

```
## @file CurveADT.py
#  @author Kajoban Kuhaparan
#  @brief Provides a CurveT class for representing a curve
#  @date 20/02/2018

import SeqServices
import Exceptions

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

    MAX_ORDER = 2
    DX = 1 * (10**(-3))

    ## @brief Interpolation function
    #  @details Calculates the linear or quadratic interpolation for a value on the curve
    #           based on the values index.
    #  @param X A sequence of X values representing x values of a curve
    #  @param Y A sequence of Y values representing y values of a curve
    #  @param o The order at which the interpolation should be calculated
    #  @param v The value which the interpolation should be calculated around.
    def interp(X, Y, o, v):

        i = SeqServices.index(X, v)

        if o == 1:
            return SeqServices.interpLin(X[i], Y[i], X[i+1], Y[i+1], v)

        elif o == 2:
            return SeqServices.interpQuad(X[i-1], Y[i-1], X[i], Y[i], X[i+1], Y[i+1], v)


    ## @brief CurveT constructor
    #  @detials Initializes a CurveT object. Stores the minimum and maximum x values,
    #           the order of the curve, and an anonymous function that can be used to
    #           evaluated the function. Before constructing the CurveT object,
    #           ensures that the independant variable is increasing, the independant and
    #           dependant sequences are the same size, and that the order does not exceed the maximum
        order
    #  @param X A sequence of x values corresponding to a curve
    #  @param Y A sequence of y values corresponding to a curve
    #  @param i The order of the curve
    def __init__(self, X, Y, i):

        if SeqServices.isAscending(X) == False:
            raise Exceptions.IndepVarNotAscending("Independant variable x is not ascending")

        elif len(X) != len(Y):
            raise Exceptions.SizeSeqMismatch("Sequence sizes do not match")

        elif i not in range(1, CurveT.MAX_ORDER + 1):
            raise Exceptions.InvalidInterpOrder("Interpolation order is illegal")

        self.minx = X[0]
        self.maxx = X[len(X) - 1]
        self.o = i
        self.f = lambda v: CurveT.interp(X, Y, self.o, v) #creates an anonymous function and returns it

    ## @brief Returns the minimum x value of the curve
    #  @return self.minx The minimum x value of the curve
    def minD(self):

        return self.minx

    ## @brief Returns the maximum x value of the curve
    #  @return self.maxx The maximum x value of the curve
    def maxD(self):

        return self.maxx

    ## @brief Returns the order of the curve
    #  @return self.o The order of the curve
    def order(self):

        return self.o;
```

```
## @brief Returns the y value of the curve evaluated at some x value.
#  @details Ensures that the x value is included in the independant range before evaluating.
#  @param x The x value that the curve is being evaluated at
#  @return self.f(x) The value of the curve evaluated at some x value
def eval(self, x):

    if (not (self.minx <= x)) or (not (x <= self.maxx)):
        raise Exceptions.OutOfDomain("Value to be evaluated out of domain of x")

    return self.f(x)

## @brief Returns the first derivative of the curve evaluated at some x value
#  @details Ensures that the x value is included in the independant range before evaluating.
#  @param x The x value that the first derivative is being evaluated at
#  @return (self.f(x + CurveT.DX) - self.f(x)) / CurveT.DX
#          The first derivative of the curve evaluated at some x value
def dfdx(self, x):

    if (not (self.minx <= x)) or (not (x <= self.maxx)):
        raise Exceptions.OutOfDomain("Value to be evaluated out of domain of x")

    return (self.f(x + CurveT.DX) - self.f(x)) / CurveT.DX

## @brief Returns the second derivative of the curve evaluated at some x value
#  @details Ensures that the x value is included in the independant range before evaluating.
#  @param x The x value that the second derivative is being evaluated at
#  @return (self.f(x + 2*CurveT.DX) - 2*self.f(x + CurveT.DX)+ self.f(x)) / (CurveT.DX**2)
#          The second derivative of the curve evaluated at some x value
def d2fdx2(self, x):

    if (not (self.minx <= x)) or (not (x <= self.maxx)):
        raise Exceptions.OutOfDomain("Value to be evaluated out of domain of x")

    return (self.f(x + 2*CurveT.DX) - 2*self.f(x + CurveT.DX)+ self.f(x)) / (CurveT.DX**2)
```

K Code for Partner's Data.py

```
## @file Data.py
#  @author Kajoban Kuhaparan
#  @brief Provides a Data class for handling input
#  @date 20/02/2018

import CurveADT
import Exceptions
import SeqServices

## @brief A Data object to store and manipulate data
class Data:

    MAX_SIZE = 10

    ## @brief Data constructor
    #  @details Initializes a data object that initially has an empty
    #           sequence of curves and an empty seeqence of special values.
    def init():

        Data.S = []
        Data.Z = []

    ## @brief Adds curves and special values to their specified sequences
    #  @details Ensures that the sequences are not full and that the special values are increasing
    #  @param s A curve to be added
    #  @param z A special value to be added
    def add(s, z):

        if len(Data.S) == Data.MAX_SIZE:
            raise Exceptions.Full("Sequence is full")

        elif len(Data.Z) > 0 and z <= Data.Z[len(Data.Z) - 1]:
            raise Exceptions.IndepVarNotAscending("Independant variable not ascending")

        Data.S.append(s)
        Data.Z.append(z)

    ## @brief Accesses a curve given an index
    #  @details Ensures that the index is in the range required
    #  @param i The index which the curve will be accessed from
    #  @return The curve at the specified index
    def getC(i):

        if i not in range(0, len(Data.S)):
            raise Exceptions.InvalidIndex("Index is not within range")

        return Data.S[i]

    ## @brief Calculates the linear interpolation given an index of a special value and value in a
         curve
    #  @details Ensures that the index is in the range required
    #  @param x The value on the curve
    #  @param z The index of the special value
    #  @return SeqServices.interpLin(Data.Z[j], Data.S[j].eval(x), Data.Z[j+1], Data.S[j+1].eval(x), z)
    #           The linear interpolation at the given values
    def eval(x, z):

        if (not SeqServices.isInBounds(Data.Z, z)):
            raise Exceptions.OutOfDomain("Z value is out of domain")

        j = SeqServices.index(Data.Z,z)

        return SeqServices.interpLin(Data.Z[j], Data.S[j].eval(x), Data.Z[j+1], Data.S[j+1].eval(x), z)

    ## @brief Calculates the data slice given an x value and returns the slice in the form of a curve
    #  @param x The x value the slice will be taken at
    #  @param i The order of the interpolation used to construct the curve
    #  @return CurveT(Data.Z, Y, i) The curve representation of the data slice
    def slice(x, i):

        Y = [Data.S[s].eval(x) for s in range(0,len(Data.Z))]
        return CurveADT.CurveT(Data.Z, Y, i)
```

L Code for Partner's SeqServices.py

```
## @file SeqServices.py
#  @author Kajoban Kuhaparan
#  @brief Provides services for curve objects
#  @date 20/02/2018

## @brief Confirms whether a range of independant values are increasing
#  @param X A sequence of independant values
#  @return False if the values are not ascending
#  @return True if the values are ascending
def isAscending(X):

    for i in range(0, len(X) - 2):
        if X[i+1] < X[i]:
            return False
        return True

## @brief Confirms whether a value is in the bounds of some range
#  @detials Assumes that isAscending is true
#  @param X A sequence that represents a range of values
#  @param x A value to be within the range
#  @return True if the specified value falls in the range
#  @return False if the specified value does not fall in the range
def isInBounds(X, x):

    return (X[0] <= x) and (x <= X[len(X) - 1])

## @brief Returns the evaluation of a linear interpolation function
#  @detials Assumes that isAscending is true
#  @param x1 The value corresponding to the x value on the left of the pivot
#  @param y1 The value corresponding to the y value on the left of the pivot
#  @param x2 The value corresponding to the x value on the right of the pivot
#  @param y2 The value corresponding to the y value on the right of the pivot
#  @param x The pivot value in which the interpolation is being done around
#  @return ( ( ( (y2 - y1) / (x2 - x1) )*(x - x1) ) + y1 )
#          The evaluation of the linear interpolation using the given values
def interpLin(x1, y1, x2, y2, x):

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

## @brief Returns the evaluation of a quadratic interpolation function
#  @detials Assumes that isAscending is true
#  @param x0 The value corresponding to the x value on the left of the pivot
#  @param y0 The value corresponding to the y value on the left of the pivot
#  @param x1 The value corresponding to the first x value on the right of the pivot
#  @param y1 The value corresponding to the first y value on the right of the pivot
#  @param x2 The value corresponding to the second x value on the right of the pivot
#  @param y2 The value corresponding to the second y value of the right of the pivot
#  @param x The pivot value in which the interpolation is being done around
#  @return (y1 + ( ((y2 - y0)/(x2 -x0) )*(x - x1)) + (((y2 - 2*y1 + y0)/(2*((x2 - x1)**2)))*((x -
    x1)**2)) )
#          The evaluation of the quadratic interpolation using the given values
def interpQuad(x0, y0, x1, y1, x2, y2, x):

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

## @brief Returns the index such that the specified value falls between two values in ascending order
#  @detials Assumes that isAscending and isInBounds are true
#  @param X A sequence that represents a range of independant values
#  @param x A value that should fall in the range
#  @return i The index such that the value x falls between its neighboring values in ascending order
def index(X, x):

    for i in range(0, len(X) - 1):
        if X[i] <= x and x <= X[i+1]:
            return i
```

M Makefile

```
PY = pytest
PYFLAGS = --cov
DOXY = doxygen
DOXYCFG = doxConfig

RMDIR = rm -rf

.PHONY: test doc clean

test:
        $(PY) $(PYFLAGS) src

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

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