Assignment 2 Solution

Henry M. 000000000

March 21, 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

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?

   output: $:= \exists\,(i, j \mid i, j \in [0..|X| - 1] : X_i \leq x \leq X_j)$
   exception: none

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

(a) modify MAX_ORDER constant in curveADT from 2 to 3

(b) Add an access program, interpCubic that calls scipy with interp1d with kind=3

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.

   - exceptions should be in SeqServices

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.

   - no size method

E Code for CurveADT.py

```
## @file CurveADT.py
#  @author Steven Palmer and Henry Madej
#  @brief CurveADT
#  @date 20/02/2018

from Exceptions import *
from SeqServices import *



## @brief An abstract data type that represents a Curve
class CurveT:

  MAX_ORDER = 2
  DX = 1E-3

  ## @brief CurveT constructor
  #  @details takes two lists of of x's and y's where the values are sorted in
  #  ascending order, and order i of the function representing the curve
  #  @param X list of x values sorted in ascending order
  #  @param Y list of corresponding y values to the the x's in the X list
  #  @param i order of the function represented by the data points (x,y) pairs
  def __init__(self, X, Y, i):
    if not isAscending(X):
      raise IndepVarNotAscending
    if i < 1 or i > CurveT.MAX_ORDER:
      raise InvalidInterpOrder
    if len(X) != len(Y):
      raise SeqSizeMismatch

    self.__minx = X[0]
    self.__maxx = X[-1]
    self.__o = i

    if self.__o == 1:
      self.__f = lambda v: interpLin(X[index(X, v)], Y[index(X, v)], X[index(X, v) + 1], Y[index(X, v)
          + 1], v)
    elif self.__o == 2:
      self.__f = lambda v: interpQuad(X[index(X, v) - 1], Y[index(X, v) - 1], X[index(X, v)],
          Y[index(X, v)], X[index(X, v) + 1], Y[index(X, v) + 1], v)

  ## @brief minD returns the minimal x value
  #  @return value representing the minimal x value
  def minD(self):
    return self.__minx

  ## @brief maxD returns the maximal x value
  #  @return value representing the maximal x value
  def maxD(self):
    return self.__maxx


  ## @brief order returns the order of the function represented by the curve
  #  @return the value of the order of the curve
  def order(self):
    return self.__o


  ## @brief eval evaluates the function at a given x value
  #  @details if x is within the range of x values for the curve it will return
  #           a y value for the curve
  #  @param x value of x to evaluate the curve at
  #  @exception OutOfDomain - if x value is not within the range of the curve's
  #             x values
  #  @return the value of the curve at x
  def eval(self, x):
    if x < self.__minx or x > self.__maxx:
      raise OutOfDomain
    return self.__f(x)


  ## @brief approximates the first derivate of the curve
  #  @details uses forward divided difference
  #  @param x value of x for the first derivate to be evaluated at
  #  @return the value of the first derivate at evaluated x
```

```
def dfdx(self, x):
  if x < self.__minx or x > self.__maxx:
    raise OutOfDomain
  return self.__f(x + self.DX) - self.__f(x) / self.DX

## @brief approximates the second derivate of the curve
#  @details uses forward divided difference
#  @param x value of x for the second derivate to be evaluated at
#  @return the value of the second derivate at evaluated x
def d2fdx2(self, x):
  if x < self.__minx or x > self.__maxx:
    raise OutOfDomain
  return self.__f(x + 2*self.DX) - 2*self.__f(x + self.DX) + self.__f(x) / self.DX**2
```

F Code for Data.py

```
## @file Data.py
#  @author Steven Palmer and Henry Madej
#  @brief Data
#  @date 20/02/2018

from CurveADT import *
import SeqServices
from Exceptions import *


## @brief An abstract data type for storing data
class Data:

  MAX_SIZE = 10

  S = []
  Z = []

  ## @brief init initial data structure
  @staticmethod
  def init():
    Data.S = []
    Data.Z = []

  ## @brief add adds a pair to the data structure
  #  @exception throws Full exception is structure is full
  #  @exception throws IndepVarNotAscending if z is less than or equal the last
  #             value in the structure
  #  @param s dependant variable value added to the structure
  #  @param z independent variable value aded to the structure
  @staticmethod
  def add(s, z):
    if len(Data.S) == Data.MAX_SIZE:
      raise Full

    if Data.Z:
      if z <= Data.Z[-1]:
        raise IndepVarNotAscending

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

  ## @brief getC gets value of the dependant variable at index i
  #  @param i index of dependant variable value
  #  @return value of dependant variable at index i
  @staticmethod
  def getC(i):
    if not (0 <= i < len(Data.S)):
      raise InvalidIndex

    return Data.S[i]

  ## @brief eval evaluates the a curve at a particular value of x
  #  @details  Uses linear interplation to find the value of the curve at x
  #  @param x the value to evaluate the curve at
  #  @param z value adjacent to the x value
  #  @return value of the curve at the specified value of x
  @staticmethod
  def eval(x, z):
    if not isInBounds(Data.Z, z):
      raise OutOfDomain

    j = SeqServices.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 slice slices a curve returning a subset of the original curve
  #  @param x the x to slice at
  #  @param i the order of the curve that is being sliced
  #  @return CurveT
  @staticmethod
  def slice(x, i):
    Y = [s.eval(x) for s in Data.S]

    return CurveT(Data.Z, Y, i)
```

G Code for SeqServices.py

```
## @file SeqServices.py
#  @author Steven Palmer and Henry Madej
#  @brief Provides the services for working with sequences
#  @date 20/02/2018


## @brief isAscending checks if a sequence is in ascending order
#  @param X sequence to be checked for ascending order
#  @return boolean False if not in ascending order True otherwise
def isAscending(X):
  for i in range(0, len(X) - 1):
    if X[i + 1] < X[i]:
      return False

  return True

## @brief isInBounds checks if value (x) is within bounds of sequence (X)
#  @param X sequence to be checked against
#  @param x value to be checked for in bounds
#  @return boolean False if not in bounds True otherwise
def isInBounds(X, x):
  return X[0] <= x and x <= X[len(X) - 1]

## @brief interpLin given two points interpolates the y value for a given x
#  @details linear interpolation
#  @param x1 x value of the first point
#  @param y1 y value of the first point
#  @param x2 x value of the second point
#  @param y2 y value of the second point
#  @param x value for which we want to interpolate a y value
#  @return y value for given x value
def interpLin(x1, y1, x2, y2, x):
  return (y2 - y1) / (x2 - x1) * (x - x1) + y1

## @brief interpQuad given three points interpolates the y value for given x
#  @details quadratic interpolation
#  @param x0 x value of the first point
#  @param y0 y value of the first point
#  @param x1 x value of the second point
#  @param y1 y value of the second point
#  @param x2 x value of the third point
#  @param y2 y value of the third point
#  @param x value for which we want to interpolate a y value
#  @return y value for given x value
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 index given a sequence and value, returns index of that value if in
#  sequence
#  @details Seq[i] <= x <= Seq[i+1]
#  @param X sequence to be searched
#  @param x value to look for
#  @return index of closest value to given x in sequence
def index(X, x):
  for i in range(0, len(X) - 1):
    if X[i] <= x and x < X[i+1]:
      return i
```

H Code for Plot.py

```
## @file Plot.py
#  @author Steven Palmer and Henry Madej
#  @brief Plotting functions
#  @date 20/02/2018

from matplotlib.pyplot import *
from Exceptions import *
from CurveADT import *

## @brief PlotSeq plots a sequence of (x, y) pairs
#  @param X corresponding x values of the sequence
#  @param Y corresponding y values of the sequence
def PlotSeq(X, Y):
  if len(X) != len(Y):
    raise SeqSizeMismatch

  figure(dpi=100)
  plot(X, Y)

  show()


## @brief PlotCurve plots a curve
#  @param c the curve to be plotted
#  @param n number of equally spaced points
def PlotCurve(c, n):
  d = (c.maxD() - c.minD())/(n+1)

  curX = c.minD()

  X = []
  Y = []

  for i in range(n):
    curX = curX + d
    X.append(curX)
    Y.append(c.eval(curX))

  PlotSeq(X, Y)
```

I Code for Load.py

```
## @file Load.py
#  @author Steven Palmer and Henry Madej
#  @brief function for retrieving data from file
#  @date 20/02/2018

from Data import *
from CurveADT import *

## @brief Load reads in data from a file, storing it in Data
#  @param s the name of the file to be read
def Load(s):
  with open(s, 'r') as infile:
    contents = infile.readlines()

  contents = [x.strip().split(',') for x in contents]

  zs = contents[0]
  os = contents[1]
  pts = contents[2:]

  Data.init()

  for i in range(len(zs)):
    X = []
    Y = []
    for j in range(0, len(pts)):
      if pts[j][i*2] == '':
        break
      X.append(float(pts[j][i*2]))
      Y.append(float(pts[j][i*2+1]))

    Data.add(CurveT(X,Y,int(os[i])), float(zs[i]))
```

J Code for Partner's CurveADT.py

K Code for Partner's Data.py

L Code for Partner's SeqServices.py

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
```