from datetime import datetime

import numpy as np

import matplotlib.pyplot as plt

import math

import matplotlib

from multiprocessing import Pool, Value, Process

from .fileHelper import FileHelperForTrapezoid

import os.path

from scipy import integrate

matplotlib.use('TkAgg')

class TrapezoidMethod():

    A = 3

    B = 1

    C = 5

    D = -5

    Xs = -50

    Xf = 50

    Ys = -70

    Yf = 70

    def setParams(self, a=3, b=1, c=5, d=-5):

        self.A = a

        self.B = b

        self.C = c

        self.D = d

    def setIntervals(self, xs=-50, xf=50, ys=-70, yf=70):

        self.Xs = xs

        self.Xf = xf

        self.Ys = ys

        self.Yf = yf

    def execute(self, n, processesNumber):

        resSum = 0

        processes = []

        dataForWrite = []

        num = Value('f', 0.0)

        hy = math.fabs(self.Yf - self.Ys) / n

        hx = math.fabs(self.Xf - self.Xs) / n

        x = np.linspace(self.Xs, self.Xf, n)

        h = math.fabs(self.Yf - self.Ys) / processesNumber

        itersToN = n / processesNumber

        startTime = datetime.now()

        for index in range(processesNumber):

            ys = self.Ys + h \* index

            p = Process(target=self.calcFromY, args=(x, ys, hy, hx, int(itersToN), num))

            processes.append(p)

            p.start()

        for proc in processes:

            proc.join()

            dataForWrite.append(num.value)

            resSum += num.value

        executeTime = datetime.now() - startTime

        self.writeFile(dataForWrite)

        integral = integrate.dblquad(self.getFunc, -70, 70, lambda x: -50, lambda x: 50)

        return [resSum, executeTime, integral[0]]

    def executeAnalysis(self, n, processesNumber):

        resSum = 0

        allSum = []

        processes = []

        executeTimes = []

        num = Value('f', 0.0)

        hy = (self.Yf - self.Ys) / n

        hx = (self.Xf - self.Xs) / n

        x = np.linspace(self.Xs, self.Xf, n)

        h = (self.Yf - self.Ys) / processesNumber

        for number in range(processesNumber):

            currentProcessNumber = number + 1

            iters = n / currentProcessNumber

            h = (self.Yf - self.Ys) / currentProcessNumber

            startTime = datetime.now()

            for index in range(currentProcessNumber):

                ys = self.Ys + h \* index

                p = Process(target=self.calcFromY, args=(x, ys, hy, hx, int(iters), num))

                processes.append(p)

                p.start()

            for proc in processes:

                proc.join()

                resSum += num.value

            executeTimes.append(datetime.now() - startTime)

            allSum.append(resSum)

            resSum = 0

            processes = []

        return [allSum, executeTimes]

    def calcFromY(self, x, ys, hy, hx, n, num):

        res = []

        for index in range(n + 1):

            y = ys + hy \* index

            res.append(self.calcFromX(x, y, hx))

        num.value = hy \* sum(res)

    def calcFromX(self, x, y, hx):

        resSum = 0

        res = self.getFunc(x, y)

        res[0] = res[0] / 2

        res[len(res) - 1] = res[len(res) - 1] / 2

        resSum = hx \* sum(res)

        return resSum

    def getFunc(self, x, y):

        result = np.sqrt(1 + self.getFuncForX(x)\*\*2 + self.getFuncForY(y)\*\*2)

        return result

    def getFuncForY(self, y):

        result = self.B \* 2 \* y + self.D

        return result

    def getFuncForX(self, x):

        result = self.A \* 2 \* x + self.C

        return result

    def getFuncForWrite(self, x, y):

        result = self.A \* x\*\*2 + self.B \* y\*\*2 + self.C \* x + self.D \* y

        return result

    def getMatrix(self):

        countX = 0

        countY = 0

        x = 0

        y = 0

        countX = int(math.fabs(self.Xf - self.Xs))

        countY = int(math.fabs(self.Yf - self.Ys))

        myFyncZnach = []

        funcArray = []

        x = self.Xs

        for \_ in range(countX):

            y = self.Ys

            for \_ in range(countY):

                funcArray.append(self.getFuncForWrite(x, y))

                y += 1

            x += 1

            myFyncZnach.append(funcArray)

            funcArray = []

        return myFyncZnach

    def draw(self, x, y, z):

        xArray, yArray = np.meshgrid(x, y)

        zArray = np.array(z)

        fig = plt.figure()

        ax = fig.add\_subplot(111, projection='3d')

        ax.plot\_surface(xArray, yArray, np.transpose(zArray), cmap='inferno')

        ax.set\_xlabel('X')

        ax.set\_ylabel('Y')

        ax.set\_zlabel('Z')

        fig.savefig("startup/static/surfaces/surface.png")

        plt.show()

        plt.close()

    def drawAnalysis(self, times, procNumbers):

        procNumbers = [str(item) for item in procNumbers]

        plt.bar(procNumbers, times)

        plt.show()

        plt.close()

    def writeFile(self, result):

        path = './Output/square.csv'

        isExit = os.path.isfile(path)

        if (isExit):

            os.remove(path)

        fileHilper = FileHelperForTrapezoid()

        fileHilper.writeToFile(result)