ПРИЛОЖЕНИЕ Б

(обязательное)

Листинг программного кода

**TrapezoindMethod.py**

from datetime import datetime

import numpy as np

import matplotlib

matplotlib.use('TkAgg')

import matplotlib.pyplot as plt

import math

from multiprocessing import Value, Process

from .fileHelper import FileHelperForTrapezoid

import os.path

from scipy import integrate

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 = (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

        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):

        surfaceImg = "startup/static/surfaces/surface.png"

        isError = False

        if (os.path.isfile(surfaceImg)):

            os.remove(surfaceImg)

        try:

            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(surfaceImg)

            plt.show()

            plt.close()

            errorMessage = False

        except ValueError:

            errorMessage = True

            plt.close()

        return errorMessage

    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)

**FileHelper.py**

import math

import csv

class FileHelperForTrapezoid():

    Xs = 0

    Xf = 0

    Ys = 0

    Yf = 0

    Z = []

    def setParams(self, xs, xf, ys, yf):

        self.Xs = xs

        self.Xf = xf

        self.Ys = ys

        self.Yf = yf

    def setMatrix(self, z):

        self.Z = z

    def writeToFiles(self,

                    xArraysPath="Output/xArray.csv",

                    yArraysPath="Output/yArray.csv",

                    matrixPath="Output/zArray.csv"):

        xs = self.Xs

        xf = self.Xf

        ys = self.Ys

        yf = self.Yf

        z = self.Z

        x = int(math.fabs(xf - xs))

        y = int(math.fabs(yf - ys))

        xArr = []

        yArr = []

        for index in range(x):

            xArr.append(xs + index)

        for index in range(y):

            yArr.append(ys + index)

        with open(xArraysPath, "w", newline='') as csvFile:

            writer = csv.writer(csvFile)

            writer.writerows(map(lambda val: [val], xArr))

        with open(yArraysPath, "w", newline='') as csvFile:

            writer = csv.writer(csvFile)

            writer.writerows(map(lambda val: [val], yArr))

        with open(matrixPath, "w", newline='') as csvFile:

            writer = csv.writer(csvFile)

            for row in range(x):

                writer.writerow(map(lambda val: val, z[row]))

    def writeToFile(self, result, squarePath="Output/square.csv"):

        with open(squarePath, "a", newline='') as csvFile:

            writer = csv.writer(csvFile)

            writer.writerows(map(lambda val: [val], result))

    def readOfFiles(self,

                   xArraysPath="Output/xArray.csv",

                   yArraysPath="Output/yArray.csv",

                   matrixPath="Output/matrix.csv"):

        x = []

        y = []

        z = []

        with open(xArraysPath, "r") as csvFile:

            reader = csv.reader(csvFile)

            for row in reader:

                x.append(int(row[0]))

        with open(yArraysPath, "r") as csvFile:

            reader = csv.reader(csvFile)

            for row in reader:

                y.append(int(row[0]))

        with open(matrixPath, "r") as csvFile:

            reader = csv.reader(csvFile)

            for row in reader:

                z.append([float(item) for item in row])

        return [x, y, z]

**Views.py**

from django.shortcuts import render

from django.http import HttpResponse, HttpResponseBadRequest

import tkinter as tk

from tkinter.filedialog import askopenfilename

from .businessLayer.trapezoidMethod import TrapezoidMethod

from .businessLayer.fileHelper import FileHelperForTrapezoid

from .businessLayer.models.analysisModel import AnalysisModel

from multiprocessing import Pool

import math

def home(request):

    return render(request, "home.html")

def surface(request):

    return render(request, "surface.html")

def getFile(request):

    root = tk.Tk()

    root.withdraw()

    path = askopenfilename(defaultextension='.csv',

                           initialdir="./Output/",

                           filetypes=[('CSV files', '\*.csv')])

    root.destroy()

    fileName = path.split("/").pop()

    html = """

        <input type="hidden" value="{0}"/>

        <div class="path">{1}</div>

    """

    data = html.format(path, fileName)

    return HttpResponse(data)

def getDataForSurface(request):

    error = request.GET.get("Error", "")

    if (error == ""):

        xpath = request.GET.get("XPath", "")

        ypath = request.GET.get("YPath", "")

        zpath = request.GET.get("ZPath", "")

        fileHelper = FileHelperForTrapezoid()

        x, y, z = fileHelper.readOfFiles(xpath, ypath, zpath)

        trapezoid = TrapezoidMethod()

        isError = trapezoid.draw(x, y, z)

        if (isError):

            return HttpResponseBadRequest()

        else:

            return HttpResponse()

    else:

        return HttpResponse(error)

def calculation(request):

    return render(request, "calculation.html")

def analysis(request):

    return render(request, "analysis.html")

def calcAnalysis(request):

    a = int(request.GET.get("A", 1))

    b = int(request.GET.get("B", 1))

    c = int(request.GET.get("C", 1))

    d = int(request.GET.get("D", 1))

    xs = int(request.GET.get("Xs", 1))

    xf = int(request.GET.get("Xf", 1))

    ys = int(request.GET.get("Ys", 1))

    yf = int(request.GET.get("Yf", 1))

    n = int(request.GET.get("N", 0.1))

    procNum = int(request.GET.get("Proc", 1))

    results, executeTimes = \_\_calcAnalysis\_\_(

        a, b, c, d, xs, xf, ys, yf, n, procNum)

    analysisData = []

    procNumbers = []

    tames = [item.seconds for item in executeTimes]

    for index in range(len(results)):

        analysisModel = AnalysisModel()

        analysisModel.Result = results[index]

        analysisModel.ExecuteTime = executeTimes[index]

        analysisModel.ProcessesNumber = index + 1

        procNumbers.append(index + 1)

        analysisData.append(analysisModel)

    trapezoid = TrapezoidMethod()

    trapezoid.drawAnalysis(tames, procNumbers)

    data = {"Results": analysisData}

    return render(request, "analysis/calcAnalysis.html", context=data)

def calcSquare(request):

    a = int(request.GET.get("A", 1))

    b = int(request.GET.get("B", 1))

    c = int(request.GET.get("C", 1))

    d = int(request.GET.get("D", 1))

    xs = int(request.GET.get("Xs", 1))

    xf = int(request.GET.get("Xf", 1))

    ys = int(request.GET.get("Ys", 1))

    yf = int(request.GET.get("Yf", 1))

    n = int(request.GET.get("N", 1))

    procNum = int(request.GET.get("Proc", 1))

    isSaveFile = request.GET.get("SaveFile", "false")

    isShowApprox = request.GET.get("ShowApprox", "false")

    result, executeTime, integral, z = \_\_calcTrapezoid\_\_(

        a, b, c, d, xs, xf, ys, yf, n, procNum)

    if (isSaveFile == "true"):

        \_\_writeFile\_\_(xs, xf, ys, yf, z, request)

    if (isShowApprox == "true"):

        approx = math.fabs(result - integral)

        data = {"Result": result, "ExecuteTime": executeTime, "ProcNum": procNum, "Integral": integral, "Approx": approx}

        return render(request, "calculation/calcSquareWithApprox.html", context=data)

    else:

        data = {"Result": result, "ExecuteTime": executeTime, "ProcNum": procNum}

        return render(request, "calculation/calcSquare.html", context=data)

def fullScreenCard(request):

    return render(request, "home/fullScreenCard.html")

def \_\_calcTrapezoid\_\_(a, b, c, d, xs, xf, ys, yf, step, procNum):

    trapezoid = TrapezoidMethod()

    trapezoid.setParams(a, b, c, d)

    trapezoid.setIntervals(xs, xf, ys, yf)

    result, executeTime, integral = trapezoid.execute(step, procNum)

    z = trapezoid.getMatrix()

    return [result, executeTime, integral, z]

def \_\_calcAnalysis\_\_(a, b, c, d, xs, xf, ys, yf, n, procNum):

    trapezoid = TrapezoidMethod()

    trapezoid.setParams(a, b, c, d)

    trapezoid.setIntervals(xs, xf, ys, yf)

    results, executeTimes = trapezoid.executeAnalysis(n, procNum)

    return [results, executeTimes]

def \_\_writeFile\_\_(xs, xf, ys, yf, z, request):

    xFile, yFile, zFile = request.GET.getlist("Files[]", [])

    fileHelper = FileHelperForTrapezoid()

    fileHelper.setParams(xs, xf, ys, yf)

    fileHelper.setMatrix(z)

    fileHelper.writeToFiles(xFile, yFile, zFile)

**Urls.py**

from django.urls import path

from startup import views

from django.contrib.staticfiles.urls import staticfiles\_urlpatterns

urlpatterns = [

    path("", views.home, name="home"),

    path("surface/", views.surface, name="surface"),

    path("calculation/", views.calculation, name="calculation"),

    path("analysis/", views.analysis, name="analysis"),

    path("calcAnalysis/", views.calcAnalysis, name="calcAnalysis"),

    path("calcSquare/", views.calcSquare, name="calcSquare"),

    path("getFile/", views.getFile, name="getFile"),

    path("getDataForSurface/", views.getDataForSurface, name="getDataForSurface"),

    path("fullScreenCard/", views.fullScreenCard, name="fullScreenCard"),

]

urlpatterns += staticfiles\_urlpatterns()