Санкт-Петербургский Государственный Электротехнический Университет "ЛЭТИ"

кафедра физики

Задание №2 по дисциплине

"Физические основы информационных технологий"

Название: Численное решение уравнения Лапласа.

|  |  |
| --- | --- |
| Фамилия И.О.: | Смирнов Д.Ю. |
| группа: | 1303 |
| Преподаватель: | Альтмарк А.М. |
| Итоговый балл: |  |
| Крайний срок сдачи: | 5.11 |

.

Санкт-Петербург 2023

Условие задания

Дана электростатическая система, состоящая из трех электродов. Внешний электрод (на рисунке 1 отмечен синим цветом) обладает потенциалом 0 В. Внутренние электроды (на рисунке отмечены красным цветом и пронумерованы как 1 и 2) обладают потенциалами, отличными от 0. Исходные данные нужно взять в файле FOIT\_IDZ2.xlsx. Для одной из указанных в таблице эквипотенциальных линий необходимо найти длину и записать её в файл IDZ2.txt. Контуры электродов можно построить по формулам, указанным в таблице и сравнить с соответствующим изображением в jpeg – файле. Координаты в данном задании можно считать безразмерными.

Помимо текстового файла IDZ2.txt в папке IDZ2 должен находиться Word-файл с отчетом, а также файл с кодом (Python, Mathcad, Mathematica). Для лучшего понимания отчетности смотрите папку “Пример организации яндекс-папки студентов”.

Пример содержания файла IDZ2.txt:

4.53258

2

1

Рисунок 1. Пример электростатической системы

**Вариант 25**

Исходные данные:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Уравнение внешнего электрода | Уравнение электрода 1 | Уравнение электрода 2 | Потенциал искомой эквипотенциали, В | Потенциал на электроде 1, В | Потенциал на электроде 2, В |
| x^2 + y^2 = 25 | 0.8\*Abs[1.8 + x]^4 + Abs[-1.8 + y]^4 = 0.7 | 0.8\*Abs[-1.8 + x]^3.5 + Abs[1.8 + y]^3.5 = 0.5 | -4 | -5 | -6 |

**Выполнение работы**

Разбиваем на сетку квадрат, в который помещается внешний электрод. Каждому узлу сетки присвоено значение в зависимости от расположения, если узел на внешнем электроде или в не его, то потенциал 0, если узел находится в/на электроде 1 или 2, то потенциал соответствует электроду, иначе потенциал узла случайное значение в [-5.5, 0).

После этого, запускается цикл, который на каждой итерации обновляет значение потенциала в узле, как среднее значение 4-х смежных с ним, если разница между потенциалом на узле на текущей итерации и прошлой итерации меньше, чем заданная точность, то цикл останавливается.

После этого для каждого узла, в котором потенциал примерно равен искомому выполняется поиск двух точек на границе квадрата (квадрат это 4 узла: текущий, справа, снизу, справа по диагонали) с искомым потенциалом, если таковые нашлись, то вычисляется расстояние между точками. Суммируя полученные расстояния получаем примерную длину искомой эквипотенциали.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАМЫ**

Файл main.py:

import numpy as np

import matplotlib.pyplot as plt

import copy

import sympy as sp

# CONSTS

def OutsideElectrode(x, y):

return np.power(x, 2) + np.power(y, 2) - 25

def InsideElectrode1(x, y):

return 0.8 \* np.power(np.abs(1.8 + x), 4) + np.power(np.abs(-1.8 + y), 4) - 0.7

def InsideElectrode2(x, y):

return 0.8\*np.power(np.abs(-1.8 + x), 3.5) + np.power(np.abs(1.8 + y), 3.5) - 0.5

def pointsPerLine():

return 300

def start():

return -5 - 0.1

def step():

return np.abs(start() \* 2) / pointsPerLine()

def epsilon():

return 0.01

def Electrode1Potential():

return -5

def Electrode2Potential():

return -6

def findPotential():

return -4

class Point:

def \_\_init\_\_(self, x, y, potential, inside=True):

self.potential = potential

self.x = x

self.y = y

self.inside = inside

def \_\_repr\_\_(self):

return str(self.potential)

def getMesh():

return [[0] \* pointsPerLine() for \_ in range(pointsPerLine())]

# VARS

dotsMesh = getMesh()

def drawElectords():

plt.axis('equal')

x = np.linspace(-5, 5, 1000)

y = np.linspace(-5, 5, 1000)

xv, yv = np.meshgrid(x, y)

plt.contour(x, y, OutsideElectrode(xv, yv), levels=[0], colors='blue')

plt.contour(x, y, InsideElectrode1(xv, yv), levels=[0], colors='red')

plt.contour(x, y, InsideElectrode2(xv, yv), levels=[0], colors='red')

def mesh():

global dotsMesh

for i in range(0, pointsPerLine()):

for j in range(0, pointsPerLine()):

currentX = start() + j \* step()

currentY = start() + i \* step()

if OutsideElectrode(currentX, currentY) >= 0:

dotsMesh[i][j] = Point(currentX, currentY, 0, False)

elif InsideElectrode1(currentX, currentY) <= 0:

dotsMesh[i][j] = Point(currentX, currentY, Electrode1Potential(), False)

elif InsideElectrode2(currentX, currentY) <= 0:

dotsMesh[i][j] = Point(currentX, currentY, Electrode2Potential(), False)

else:

dotsMesh[i][j] = Point(currentX, currentY, np.random.uniform(

(Electrode1Potential() + Electrode2Potential()) / 2,

0

),

True

)

def calculation():

global dotsMesh

count = 0

tmpDotsMesh = copy.deepcopy(dotsMesh)

flag = True

while flag:

count += 1

for i in range(0, pointsPerLine()):

for j in range(0, pointsPerLine()):

if(dotsMesh[i][j].inside):

newPotential = (dotsMesh[i-1][j].potential + dotsMesh[i][j-1].potential + dotsMesh[i+1][j].potential + dotsMesh[i][j+1].potential) / 4

tmpDotsMesh[i][j] = Point(

dotsMesh[i][j].x,

dotsMesh[i][j].y,

newPotential

)

flag = False

for i in range(0, pointsPerLine()):

for j in range(0, pointsPerLine()):

if (np.abs(tmpDotsMesh[i][j].potential - dotsMesh[i][j].potential) > epsilon()):

flag = True

dotsMesh[i][j] = tmpDotsMesh[i][j]

#print(count)

def solve(ltp, rtp, lbp, rbp):

points = []

a, k1, k2 = sp.symbols('a k1 k2')

eq1 = a + k1 \* ltp.x + k2 \* ltp.y - ltp.potential

eq2 = a + k1 \* lbp.x + k2 \* lbp.y - lbp.potential

eq3 = a + k1 \* rtp.x + k2 \* rtp.y - rtp.potential

solve1 = sp.nsolve((eq1, eq2, eq3), (a, k1, k2), (1, 1, 1))

topX = float((-4 - solve1[0] - solve1[2] \* ltp.y) / solve1[1])

if ltp.x < topX < rtp.x:

points.append([topX, ltp.y])

leftY = float((-4 - solve1[0] - solve1[1] \* ltp.x) / solve1[2])

if lbp.y <= leftY <= ltp.y:

points.append([lbp.x, leftY])

a, k1, k2 = sp.symbols('a k1 k2')

eq1 = a + k1 \* rtp.x + k2 \* rtp.y - rtp.potential

eq2 = a + k1 \* rbp.x + k2 \* rbp.y - rbp.potential

eq3 = a + k1 \* lbp.x + k2 \* lbp.y - lbp.potential

solve2 = sp.nsolve((eq1, eq2, eq3), (a, k1, k2), (1, 1, 1))

bottomX = float((-4 - solve2[0] - solve2[2] \* lbp.y) / solve2[1])

if lbp.x <= bottomX <= rbp.x:

points.append([bottomX, lbp.y])

rightY = float((-4 - solve2[0] - solve2[1] \* rtp.x) / solve2[2])

if rbp.y <= rightY <= rtp.y:

points.append([rbp.x, rightY])

if len(points) == 2:

plt.plot([points[0][0],points[1][0]], [points[0][1],points[1][1]], c='green')

return np.sqrt(np.power(np.abs(points[0][0] - points[1][0]), 2) + np.power(np.abs(points[0][1] - points[1][1]),2))

return 0

def findEquipotential():

global dotsMesh

l = 0

for i in range(0, pointsPerLine()):

for j in range(0, pointsPerLine()):

if dotsMesh[i][j].inside == False:

continue

if np.abs(dotsMesh[i][j].potential - findPotential()) <= 0.2:

l += solve(dotsMesh[i][j],

dotsMesh[i][j+1],

dotsMesh[i-1][j],

dotsMesh[i-1][j + 1]

)

print(l)

def main():

drawElectords()

mesh()

calculation()

findEquipotential()

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

main()