Санкт-Петербургский Государственный Электротехнический Университет "ЛЭТИ"

кафедра физики

Задание №2 по дисциплине

"Физические основы информационных технологий"

Название: Численное решение уравнения Лапласа.

|  |  |
| --- | --- |
| Фамилия И.О.: | Гирман А.В. |
| группа: | 1303 |
| Преподаватель: | Альтмарк А.М. |
| Итоговый балл: |  |
| Крайний срок сдачи: | 5.11 |

.

Санкт-Петербург 2023

Условиезадания

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

Таблица с исходными данными

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Вар. | Уравнение внешнего электрода | Уравнения электрода 1 | Уравнения электрода 2 | Потенциал искомой эквипотенциали, В | Потенциал на электроде 1, В | Потенциал на электроде 2,В | Файл с картинкой |
| 22 | x^2 + y^2 = 25 | Abs[1.8 + x]^2 + 0.8\*Abs[y]^2 = 0.6 | 0.5\*Abs[-1.8 + x]^4 + 0.3\*Abs[y]^4 = 0.5 | 2 | 6 | -5 | 22.jpeg |

Основные теоретические положения

Алгоритм построения правильного распределения потенциалов магнитного поля:

1. Пространство внутри внешнего электрода и между внутренними электродами разбивается сеткой, состоящей из точек.
2. Точки на границах электродов имеет такой же потенциал, как и сам электрод.
3. Потенциалы для узлов в пространстве между электродами задаётся случайным образом от φ1 до φ2.
4. Дальше используя уравнение Лапласа, считаем для каждой φi,j её новое значение.
5. Повторяем до тех пор, пока |φs-1i,j - φsi,j | ≤ ɛ , где s – номер итерации.

В итоге получается физически правильное распределение потенциалов в электромагнитном поле.

Выполнение работы

Нарисованы исходные электроды с помощью библиотек python и указанным в таблице функциям.

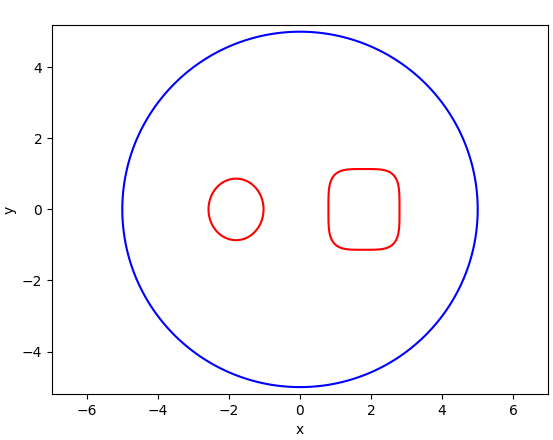


Рисунок 1. Нарисованные электроды.

Нарисована разметка, состоящая из точек серого цвета и шагом в 0.04.

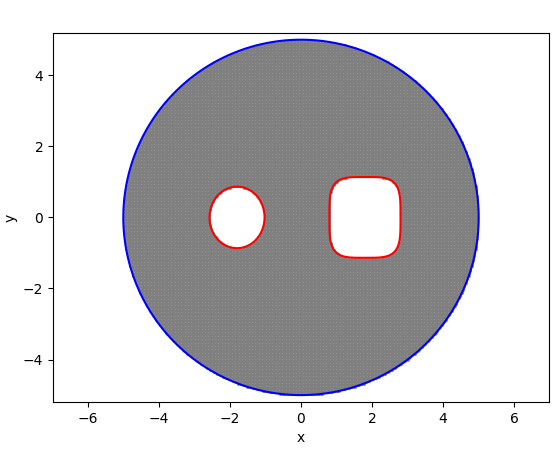


Рисунок 2. Нарисована сетка состоящие из серых точек

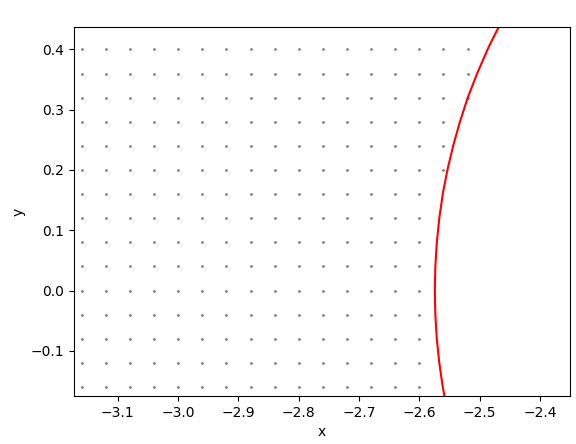


Рисунок 3. Сетка в приближении.

После этого для каждых четырёх точек считает уравнение верхнего и нижнего треугольника, по коэффициентам которых находятся точки потенциала, относящихся к эквипотенциали.

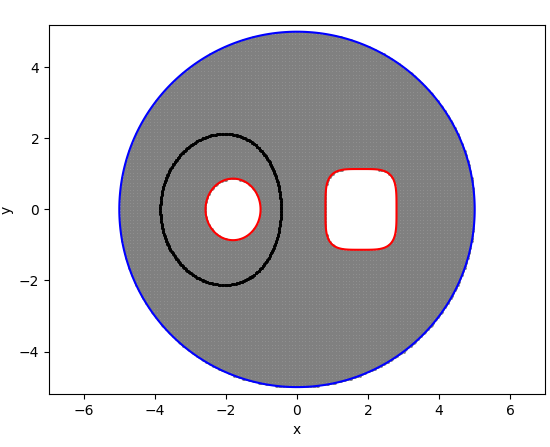


Рисунок 4. Найденная эквипотенциаль.

**ПРИЛОЖЕНИЕ А**

**ПРОГРАММА IDZ2.PY**

import numpy as np

import sympy as sp

import random as ran

import matplotlib.pyplot as plt

step = 0.04

find\_phi = 2

phi\_1 = 6

phi\_2 = -5

size = 260

offset = 0.2

epsilon = 0.01

x = np.arange(-5 - offset, 5 + offset + step, step)

y = np.arange(-5 - offset, 5 + offset + step, step)

array\_of\_points = []

iterationFrag = 0

iterationLen = 0

x\_points = []

y\_points = []

color = []

class Point:

def \_\_init\_\_(self, x, y, phi, inElectrode):

self.x = x

self.y = y

self.phi = phi

self.inElectrode = inElectrode

def array\_init():

for i in range(0, size):

array\_of\_points.append([])

for j in range(0, size):

x\_cord = -5 - offset + j \* step

y\_cord = -5 - offset + i \* step

array\_of\_points[i].append(Point(x\_cord, y\_cord, 0, False))

def external\_electrode(x, y):

return x \*\* 2 + y \*\* 2 - 25

def electrode\_1(x, y):

return np.abs(x + 1.8) \*\* 2 + 0.8 \* np.abs(y) \*\* 2 - 0.6

def electrode\_2(x, y):

return 0.5 \* np.abs(-1.8 + x) \*\* 4 + 0.3 \* np.abs(y) \*\* 4 - 0.5

def initElectrode():

X, Y = np.meshgrid(x, y)

external\_electrode\_value = external\_electrode(X, Y)

electrode\_1\_value = electrode\_1(X, Y)

electrode\_2\_value = electrode\_2(X, Y)

plt.contour(X, Y, external\_electrode\_value, [0], colors='blue')

plt.contour(X, Y, electrode\_1\_value, [0], colors='red')

plt.contour(X, Y, electrode\_2\_value, [0], colors='red')

plt.xlabel('x')

plt.ylabel('y')

plt.axis('equal')

def fragmentation():

for i in range(0, size):

for j in range(0, size):

x\_cord = -5 - offset + j \* step

y\_cord = -5 - offset + i \* step

if electrode\_1(x\_cord, y\_cord) <= 0:

array\_of\_points[i][j] = Point(x\_cord, y\_cord, phi\_1, False)

elif electrode\_2(x\_cord, y\_cord) <= 0:

array\_of\_points[i][j] = Point(x\_cord, y\_cord, phi\_2, False)

elif ((x\_cord \*\* 2 + y\_cord \*\* 2 < 25) and (np.abs(x\_cord + 1.8) \*\* 2 + 0.8 \* np.abs(y\_cord) \*\* 2 > 0.6)

and (0.5 \* np.abs(-1.8 + x\_cord) \*\* 4 + 0.3 \* np.abs(y\_cord) \*\* 4 > 0.5)):

array\_of\_points[i][j] = Point(x\_cord, y\_cord, ran.uniform(phi\_2, phi\_1), True)

x\_points.append(array\_of\_points[i][j].x)

y\_points.append(array\_of\_points[i][j].y)

plt.scatter([x\_points], [y\_points], c='gray', s=1)

def algorithm():

global iterationFrag

new\_phi\_array\_of\_points = [[0] \* size for \_ in range(size)]

iteration = True

while iteration:

iterationFrag += 1

iteration = False

for i in range(0, size):

for j in range(0, size):

if array\_of\_points[i][j].inElectrode:

new\_phi\_array\_of\_points[i][j] = (array\_of\_points[i + 1][j].phi

+ array\_of\_points[i - 1][j].phi

+ array\_of\_points[i][j + 1].phi

+ array\_of\_points[i][j - 1].phi) / 4

for i in range(0, size):

for j in range(0, size):

if array\_of\_points[i][j].inElectrode:

if not iteration and (np.abs(array\_of\_points[i][j].phi - new\_phi\_array\_of\_points[i][j]) > epsilon):

iteration = True

array\_of\_points[i][j].phi = new\_phi\_array\_of\_points[i][j]

print(f'Количество итераций алгоритма: {iterationFrag}')

def equipotentialPoints(LeftTopPoint, RightTopPoint, LeftBottomPoint, RightBottomPoint):

global iterationLen

points = []

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

top\_triangle\_eq1 = a + k1 \* LeftTopPoint.x + k2 \* LeftTopPoint.y - LeftTopPoint.phi

top\_triangle\_eq2 = a + k1 \* RightTopPoint.x + k2 \* RightTopPoint.y - RightTopPoint.phi

top\_triangle\_eq3 = a + k1 \* LeftBottomPoint.x + k2 \* LeftBottomPoint.y - LeftBottomPoint.phi

top\_triangle\_solve = sp.nsolve((top\_triangle\_eq1, top\_triangle\_eq2, top\_triangle\_eq3), (a, k1, k2), (1, 1, 1))

xTop = (find\_phi - top\_triangle\_solve[0] - top\_triangle\_solve[2] \* LeftTopPoint.y) / top\_triangle\_solve[1]

if LeftTopPoint.x < xTop < RightTopPoint.x:

plt.scatter(xTop, LeftTopPoint.y, c='black', s=1)

points.append((xTop, LeftTopPoint.y))

yLeft = (find\_phi - top\_triangle\_solve[0] - top\_triangle\_solve[1] \* LeftTopPoint.x) / top\_triangle\_solve[2]

if LeftBottomPoint.y < yLeft < LeftTopPoint.y:

plt.scatter(LeftTopPoint.x, yLeft, c='black', s=1)

points.append((LeftTopPoint.x, yLeft))

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

bottom\_triangle\_eq1 = a + k1 \* RightBottomPoint.x + k2 \* RightBottomPoint.y - RightBottomPoint.phi

bottom\_triangle\_eq2 = a + k1 \* RightTopPoint.x + k2 \* RightTopPoint.y - RightTopPoint.phi

bottom\_triangle\_eq3 = a + k1 \* LeftBottomPoint.x + k2 \* LeftBottomPoint.y - LeftBottomPoint.phi

bottom\_triangle\_solve = sp.nsolve((bottom\_triangle\_eq1, bottom\_triangle\_eq2, bottom\_triangle\_eq3),

(a, k1, k2), (1, 1, 1))

xBottom = (find\_phi - bottom\_triangle\_solve[0] -

bottom\_triangle\_solve[2] \* RightBottomPoint.y) / bottom\_triangle\_solve[1]

if LeftBottomPoint.x < xBottom < RightBottomPoint.x:

plt.scatter(xBottom, RightBottomPoint.y, c='black', s=1)

points.append((xBottom, RightBottomPoint.y))

yRight = (find\_phi - bottom\_triangle\_solve[0] -

bottom\_triangle\_solve[1] \* RightBottomPoint.x) / bottom\_triangle\_solve[2]

if RightBottomPoint.y < yRight < RightTopPoint.y:

plt.scatter(RightBottomPoint.x, yRight, c='black', s=1)

points.append((RightBottomPoint.x, yRight))

if len(points) == 2:

return np.sqrt((np.abs(float(points[0][0] - points[1][0])) \*\* 2) +

(np.abs(float(points[0][1] - points[1][1])) \*\* 2))

return 0

def findLength():

lenEquip = 0

for i in range(0, size):

for j in range(0, size):

if array\_of\_points[i][j].inElectrode:

if find\_phi - 0.1 < array\_of\_points[i][j].phi < find\_phi + 0.1:

lenEquip += equipotentialPoints(array\_of\_points[i][j],

array\_of\_points[i][j + 1],

array\_of\_points[i - 1][j],

array\_of\_points[i - 1][j + 1])

print(f'Количество итерация для нахождения длины: {iterationLen}')

print(f'Длина: {lenEquip}')

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

array\_init()

initElectrode()

fragmentation()

algorithm()

findLength()

plt.show()