**Санкт-Петербургский Государственный Электротехнический Университет "ЛЭТИ"**

**кафедра физики**

**Задание №2 по дисциплине**

**"Физические основы информационных технологий"**

**Название: Численное решение уравнения Лапласа.**

|  |  |
| --- | --- |
| Фамилия И.О.: | Хабибуллина А.М. |
| 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

**Дано:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **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** |
| 13 | x^2 + y^2 = 25 | 0.5\*Abs[-1.8 + x]^1.5 + Abs[y]^1.5 = 0.6 | 0.3\*Abs[1.8 + x]^3.5 + 0.3\*Abs[y]^3.5 = 0.7 | 1 | -5 | 6 | 13.jpeg |

**Основные теоретические положения**

Электрический потенциал (или напряжение) - это скалярная величина, которая характеризует энергию заряженной частицы в электрическом поле. Измеряется в вольтах (В).

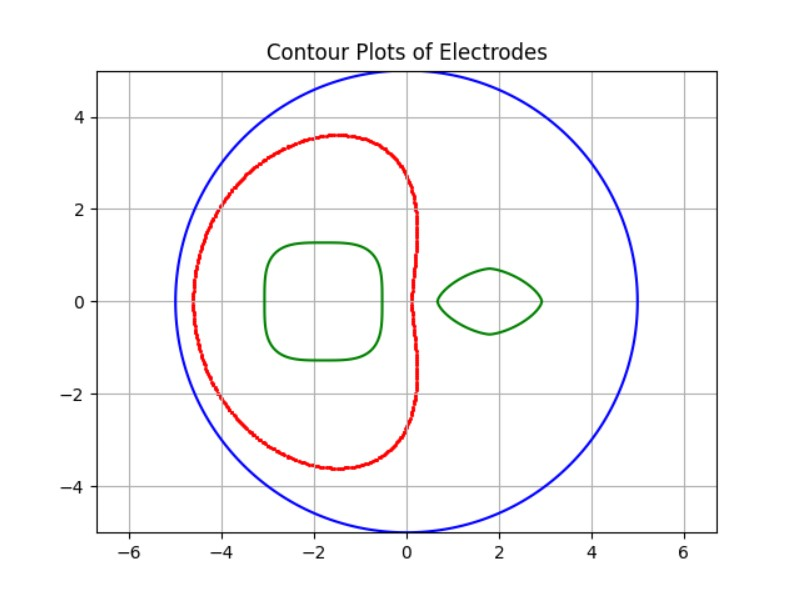
Эквипотенциальные поверхности - это поверхности, на которых потенциал имеет постоянное значение. Они перпендикулярны линиям электрической силы.

Алгоритм построения правильного распределения потенциалов с помощью метода сеток:

1. **Задание сетки:**
   * Разбейте рассматриваемую область на сетку, где каждая точка сетки представляет собой узел. Эта сетка может быть двумерной или трехмерной в зависимости от геометрии проблемы. Расстояние между узлами выбирается в зависимости от конкретных условий задачи.
2. **Начальные условия:**
   * Установите начальные условия для потенциала магнитного поля на узлах сетки. Обычно начальные условия берутся равными нулю или другому известному значению в зависимости от задачи.
3. **Расчет потенциала:**
   * На каждом узле сетки вычислите новое значение потенциала магнитного поля на основе предыдущих значений и уравнения, описывающего магнитное поле.
4. **Граничные условия:**
   * Учтите граничные условия задачи, задавая значения потенциала магнитного поля на границе области. Эти условия могут варьироваться в зависимости от конкретной задачи. Обычно на границе устанавливаются известные значения потенциала.
5. **Итерационный процесс:**
   * Повторяйте шаги 3 и 4 до тех пор, пока изменения потенциала на каждом узле сетки не станут незначительными. Это означает, что вы сходитесь к решению задачи.

**Вывод:**

Для нахождения длины эквипотенциальной линии была написана программа на Python, реализующая метод сеток. Также по исходным и найденным данным были построены контуры электродов.



**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

**IDZ2.py**

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import fsolve

import random

e0 = 0

e1 = -5

e2 = 6

e\_find = 1

eps = 0.001

X = []

Y = []

num\_points = 200

R = 6

def electrod0(x, y):

return pow(x, 2) + pow(y, 2) - 25

def electrod1(x, y):

return 0.5\*np.abs(-1.8 + x)\*\*1.5 + np.abs(y)\*\*1.5 - 0.6

def electrod2(x, y):

return 0.3\*np.abs(1.8 + x)\*\*3.5 + 0.3\*np.abs(y)\*\*3.5 - 0.7

class Point:

def \_\_init\_\_(self, x, y, V):

self.x = x

self.y = y

self.V = V

self.take = True

points = [[Point(-R + 2 \* R \* x / num\_points,

-R + 2 \* R \* y / num\_points,

e1 + random.random() \* (e2 - e1)) for y in range(num\_points)] for x in

range(num\_points)]

# marking edges

for x in range(num\_points):

for y in range(num\_points):

res = electrod0(points[x][y].x, points[x][y].y)

if (res > 0 - 0.5 and res < 0 + 0.5):

X.append(points[x][y].x)

Y.append(points[x][y].y)

if res >= 0:

points[x][y].V = e0

points[x][y].take = False

continue

res = electrod1(points[x][y].x, points[x][y].y)

if (res > 0 - 0.1 and res < 0 + 0.1):

X.append(points[x][y].x)

Y.append(points[x][y].y)

if res <= 0:

points[x][y].V = e1

points[x][y].take = False

continue

res = electrod2(points[x][y].x, points[x][y].y)

if (res > 0 - 0.1 and res < 0 + 0.1):

X.append(points[x][y].x)

Y.append(points[x][y].y)

if res <= 0:

points[x][y].V = e2

points[x][y].take = False

continue

def iter():

V\_new = [[0 for y in range(num\_points)] for x in range(num\_points)]

for x in range(num\_points):

for y in range(num\_points):

if (points[x][y].take):

V\_new[x][y] = (points[x + 1][y].V +

points[x][y + 1].V +

points[x - 1][y].V +

points[x][y - 1].V) / 4

isLastIteration = True

for x in range(num\_points):

for y in range(num\_points):

if points[x][y].take:

if np.abs(points[x][y].V - V\_new[x][y]) > eps:

isLastIteration = False

points[x][y].V = V\_new[x][y]

if not isLastIteration:

return True

else:

return False

flag = True

while (flag):

flag = iter()

eq\_list = []

def f(variables):

(a, k1, k2) = variables

res = []

for eq in eq\_list:

res.append(eval(eq))

return res

def create\_eq(L):

return f"a + k1 \* {L.x} + k2 \* {L.y} - {L.V}"

def findlength(L1, R1, L2, R2):

global eq\_list

points = []

eq\_list = [create\_eq(L1), create\_eq(R1), create\_eq(L2)]

(a, k1, k2) = fsolve(f, [1, 1, 1])

xT = (e\_find - a - k2 \* L1.y) / k1

if (xT > L1.x and xT < R1.x):

plt.scatter(xT, L1.y, c='red', s=1)

points.append((xT, L1.y))

yL = (e\_find - a - k1 \* L1.x) / k2

if (yL > L2.y and yL < L1.y):

plt.scatter(L1.x, yL, c='red', s=1)

points.append((L1.x, yL))

eq\_list = [create\_eq(R2), create\_eq(R1), create\_eq(L2)]

(a, k1, k2) = fsolve(f, [1, 1, 1])

xB = (e\_find - a - k2 \* R2.y) / k1

if (xB > L2.x and xB < R2.x):

plt.scatter(xB, R2.y, c='red', s=1)

points.append((xB, R2.y))

yR = (e\_find - a - k1 \* R2.x) / k2

if (yR > R2.y and yR < R1.y):

plt.scatter(R2.x, yR, c='red', s=1)

points.append((R2.x, yR))

if (len(points) == 2):

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

return 0

totalLength = 0

for x in range(num\_points):

for y in range(num\_points):

if (points[x][y].take):

if (points[x][y].V < e\_find + 0.5 and

points[x][y].V > e\_find - 0.5):

totalLength += findlength(points[x][y],

points[x + 1][y],

points[x][y - 1],

points[x + 1][y - 1])

print(totalLength)

x\_range = np.linspace(-5, 5, 400)

y\_range = np.linspace(-5, 5, 400)

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

external\_electrode\_values = electrod0(X, Y)

electrode1\_values = electrod1(X, Y)

electrode2\_values = electrod2(X, Y)

plt.contour(X, Y, external\_electrode\_values, levels=[0], colors='b')

plt.contour(X, Y, electrode1\_values, levels=[0], colors='g')

plt.contour(X, Y, electrode2\_values, levels=[0], colors='g')

plt.title('Contour Plots of Electrodes')

plt.axis('equal')

plt.grid(True)

plt.show()