Санкт-Петербургский Государственный Электротехнический Университет "ЛЭТИ"

кафедра физики

Задание №2 по дисциплине

"Физические основы информационных технологий"

Название: Численное решение уравнения Лапласа.

|  |  |
| --- | --- |
| Фамилия И.О.: | Сырцева Д.Д |
| группа: | 1303 |
| Преподаватель: | Альтмарк А.М. |
| Итоговый балл: |  |
|  |  |

Крайний срок сдачи: 05.11.23

.

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

Рис 1. Пример электростатической системы

**Исходные данные (11 вариант)**

|  |  |  |
| --- | --- | --- |
| **Уравнение внешнего электрода** | **Уравнения электрода 1** | **Уравнения электрода 2** |
| x^2 + y^2 = 25 | 0.3\*Abs[1.5 + x]^4 + 0.3\*Abs[-1.5 + y]^4 = 0.5 | Abs[-1.5 + x]^4 + 0.3\*Abs[1.5 + y]^2.5 = 0.5 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Потенциал искомой эквипотенциали, В** | **Потенциал на электроде 1, В** | **Потенциал на электроде 2,В** | **Файл с картинкой** |
| 3 | -6 | 6 | 11.jpeg |

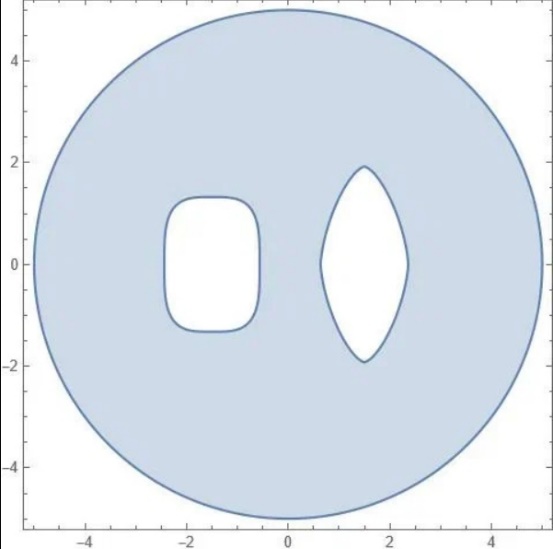


Рис.2 Контуры электродов.

**Выполнение работы**

1. Внутри внешнего электрода необходимо разбить пространство на прямоугольники.
2. Генерируем значения потенциалов в узлах. Для узлов, находящихся рядом с электродами (на границе), присвоим потенциал данного электрода. Потенциалы для узлов в пространстве между электродами зададим случайным образом (от Ф1 до Ф2).
3. После этого выполним последовательность итераций, в которых пересчитаем значения потенциалов в узлах в зависимости от окружающих точек, а именно как усредненное значение данных точек.
4. Далее найдем длину необходимой эквипотенциальной линии, и нарисуем данную линию.
5. Также нарисуем контуры электродов.

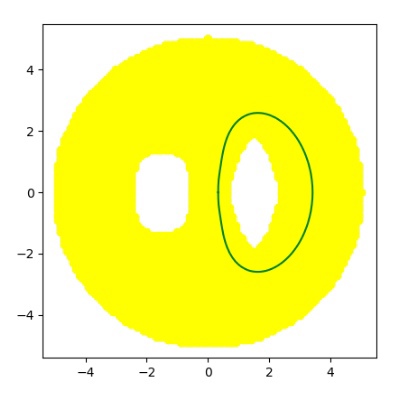


Рис.3 Полученный график.

**Вывод:**

В результате выполнения задания, была найдена длина указанной в таблице эквипотенциальной линии.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Файл **main.py**

import random

import numpy as np

import copy

import matplotlib.pyplot as plt

class Point:

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

self.x = x

self.y = y

self.f = f

def is\_in\_constraints(a, b):

constraint\_circle = a \*\* 2 + b \*\* 2 <= 25

constraint\_1 = abs(1.5 + a) \*\* 3.5 + 0.3 \* abs(b) \*\* 3.5 >= 0.8

constraint\_2 = abs(-1.5 + a) \*\* 1.5 + 0.3 \* abs(b) \*\* 1.5 >= 0.8

return constraint\_circle and constraint\_1 and constraint\_2

def initialize\_points():

for x in partition:

inner\_list = []

for y in partition:

f = None

if is\_in\_constraints(x, y):

circle\_constraint = abs(x \*\* 2 + y \*\* 2 - 25)

constraint\_1 = abs(abs(1.5 + x) \*\* 3.5 + 0.3 \* abs(y) \*\* 3.5 - 0.8)

constraint\_2 = abs(abs(-1.5 + x) \*\* 1.5 + 0.3 \* abs(y) \*\* 1.5 - 0.8)

if circle\_constraint <= 1:

f = 0

elif constraint\_1 <= 0.5:

f = f1

elif constraint\_2 <= 0.5:

f = f2

else:

f = random.uniform(min(f1, f2), max(f1, f2))

point = Point(x, y, f)

inner\_list.append(point)

arr\_point.append(inner\_list)

def perform\_iteration():

n = 0

while (n <= 500):

for y in range(1, len(arr\_point) - 1):

for x in range(1, len(arr\_point[y]) - 1):

neighbors = [

arr\_point[y][x - 1].f,

arr\_point[y][x + 1].f,

arr\_point[y - 1][x].f,

arr\_point[y + 1][x].f

]

no\_none\_found = True

for neighbor in neighbors:

if neighbor is None:

no\_none\_found = False

break

if not no\_none\_found:

continue

arr\_point[y][x].f = sum(neighbors) / 4

n += 1

def calculate\_point(tmp, neighbor, flag):

if flag == 'x':

new\_x = tmp.x + (abs(f - tmp.f) / abs(tmp.f - neighbor.f)) \* step

new\_y = tmp.y

else:

new\_x = tmp.x

new\_y = tmp.y + (abs(f - tmp.f) / abs(tmp.f - neighbor.f)) \* step

arr\_x.append(new\_x)

arr\_y.append(new\_y)

def process\_equi\_potential(f\_value):

for i in range(1, len(arr\_point) - 1):

for j in range(1, len(arr\_point[i]) - 1):

current = arr\_point[i][j]

right\_neighbor = arr\_point[i + 1][j]

down\_neighbor = arr\_point[i][j + 1]

if (current.f is None or right\_neighbor.f is None or down\_neighbor.f is None):

continue

if (current.f <= f\_value <= right\_neighbor.f or current.f >= f\_value >= right\_neighbor.f):

calculate\_point(current, right\_neighbor, 'x')

if (current.f <= f\_value <= down\_neighbor.f or current.f >= f\_value >= down\_neighbor.f):

calculate\_point(current, down\_neighbor, 'y')

def result\_points():

points = [(x, y) for x, y in zip(arr\_x, arr\_y)]

sorted\_points = sorted(points, key=lambda lst: lst[0])

left\_point = sorted\_points[0]

right\_point = sorted\_points[-1]

x1, y1 = left\_point

x2, y2 = right\_point

slope = (y2 - y1) / (x2 - x1)

below\_border = [(x1, y1)]

above\_border = [(x2, y2)]

for x, y in points:

border = slope \* (x - x1) + y1

if y < border:

below\_border.append((x, y))

elif y > border:

above\_border.append((x, y))

below\_border.sort()

above\_border.sort(reverse=True)

above\_border.append((x1, y1))

result\_points = below\_border + above\_border

x\_values, y\_values = zip(\*result\_points)

plt.plot(x\_values, y\_values, c='green')

return result\_points

def find\_length(result\_points):

length = 0

for i in range(1, len(result\_points)):

x0, y0 = result\_points[i - 1]

x1, y1 = result\_points[i]

length += np.sqrt((x1 - x0) \*\* 2 + (y1 - y0) \*\* 2)

return length

f1 = -6

f2 = 6

f = 3

step = 0.1

arr\_x = []

arr\_y = []

arr\_point = []

partition = np.arange(f1, f2 + step, step)

plt.gca().set\_aspect('equal')

initialize\_points()

perform\_iteration()

list = []

for row in arr\_point:

for el in row:

if el.f is not None:

list.append(el)

plt.scatter([el.x for el in list], [el.y for el in list], c='yellow')

process\_equi\_potential(f)

length = find\_length(result\_points())

print(f"Искомая длина: {length}")

plt.show()