Санкт-Петербургский Государственный Электротехнический Университет "ЛЭТИ"

кафедра физики

Задание №2 по дисциплине

"Физические основы информационных технологий"

Название: Численное решение уравнения Лапласа.

|  |  |
| --- | --- |
| Фамилия И.О.: | Насонов Я.К. |
| 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

**Таблица с исходными данными**

Вариант 8

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Уравнение внешнего электрода | Уравнение первого электрода | Уравнение второго электрода | Потенциал искомой эквипотенциали | Потенциал на электроде 1  Ф1 | Потенциал на электроде 2  Ф2 |
| x^2 + y^2 = 25 | 0.8 Abs[-1.5 + x]^3.5 + 0.3 Abs[1.5 + y]^3.5 = 0.6 | Abs[1.5 + x]^4 + Abs[-1.5 + y]^4 = 0.8 | 3 | 5 | 5 |

**Основные теоретические положения**

Алгоритм построения правильного распределения потенциалов магнитного поля:

1. Пространство внутри внешнего электрода и между внутренними электродами разбивается сеткой
2. Узлы сетки, попавшие на границы электродов, имеют потенциал соответствующего электрода. Потенциал внешнего электрода равен 0.
3. Остальные узлы пока имеют случайный потенциал в диапазоне Ф1 ― Ф2
4. Для каждого не граничного узла высчитывается новое значение – среднее арифметическое соседних узлов (сверху, снизу, слева, справа).
5. Пункт 4 повторяется до момента, пока разница между соседними узлами не станет меньше заданного ε.

В итоге получается физически правильное распределение потенциалов в электромагнитном поле.

**ПРИЛОЖЕНИЕ А**

**ПРОГРАММА MAIN.PY**

import random  
from itertools import chain  
  
import matplotlib.pyplot as plt  
import numpy as np  
  
  
class Point:  
 def \_\_init\_\_(self, x, y, f):  
 self.x = x  
 self.y = y  
 self.f = f  
  
def isIn(x, y):  
 res = x \*\* 2 + y \*\* 2 <= 25  
 res1 = 0.8 \* abs(-1.5 + x) \*\* 3.5 + 0.3 \* abs(1.5 + y) \*\* 3.5 >= 0.6  
 res2 = abs(1.5 + x) \*\* 4 + abs(-1.5 + y) \*\* 4 >= 0.8  
 return res and res1 and res2  
  
  
F1 = 5  
F2 = 5  
f = 3  
step = 0.1  
start = -5  
finish = 6  
size = 4  
epsilon = 0.00001  
  
t1 = np.arange(start, finish, step)  
arr\_x = []  
arr\_y = []  
arr\_point = []  
  
  
def inicialisation():  
 for x in t1:  
 arr\_point.append([])  
 for y in t1:  
 if isIn(x, y):  
 if abs(x \*\* 2 + y \*\* 2 - 25) <= 1:  
 f = 0  
 elif abs(0.8 \* abs(-1.5 + x) \*\* 3.5 + 0.3 \* abs(1.5 + y) \*\* 3.5 - 0.6) <= 0.4:  
 f = F1  
 elif abs(abs(1.5 + x) \*\* 4 + abs(-1.5 + y) \*\* 4 - 0.8) <= 0.4:  
 f = F2  
 else:  
 f = random.random() \* (abs(F1) + abs(F2)) + min(F1, F2)  
 else:  
 f = None  
 point = Point(x, y, f)  
 arr\_point[-1].append(point)  
  
  
def iteration():  
 max\_delta = 0  
 for y in range(1, len(arr\_point) - 2):  
 for x in range(1, len(arr\_point[y]) - 2):  
 neighbours = [arr\_point[y][x - 1].f,  
 arr\_point[y][x + 1].f,  
 arr\_point[y - 1][x].f,  
 arr\_point[y + 1][x].f]  
 if any(potent is None for potent in neighbours):  
 continue  
 delta = max(neighbours) - min(neighbours)  
 if delta > max\_delta:  
 max\_delta = delta  
 arr\_point[y][x].f = sum(neighbours) / 4  
 return max\_delta  
  
  
def repeat(epsilon):  
 previous\_delta = abs(F1) + abs(F2) + 10 *# верхняя граница разности, с хвостиком чтоб наверняка* n = 0  
 while True:  
 n += 1  
 new\_delta = iteration()  
 if (abs(previous\_delta - new\_delta) < epsilon):  
 break  
 previous\_delta = new\_delta  
  
  
def add(point1, point2):  
 arr\_x.append(point1.x)  
 arr\_x.append(point2.x)  
 arr\_y.append(point1.y)  
 arr\_y.append(point2.y)  
  
  
def findpoint(tmp, neighboor, flag):  
 diapason = abs(tmp.f - neighboor.f)  
 if diapason == 0: *# сработает если ищем 0* add(tmp, neighboor)  
 return True  
 s = abs(f - tmp.f)  
 if flag == 'x':  
 arr\_x.append(tmp.x + (s / diapason) \* step)  
 arr\_y.append(tmp.y)  
 else:  
 arr\_x.append(tmp.x)  
 arr\_y.append(tmp.y + (s / diapason) \* step)  
 return False  
  
  
def find(f):  
 for i in range(1, len(arr\_point) - 2):  
 for j in range(1, len(arr\_point[i]) - 2):  
 tmp = arr\_point[i][j]  
 right = arr\_point[i + 1][j]  
 down = arr\_point[i][j + 1]  
 if tmp.f is None or right.f is None or down.f is None:  
 continue  
 if tmp.f <= f <= right.f or tmp.f >= f >= right.f:  
 findpoint(tmp, right, 'x')  
  
 if tmp.f <= f <= down.f or tmp.f >= f >= down.f:  
 findpoint(tmp, down, 'y')  
  
  
def sort():  
 points = list(zip(arr\_x, arr\_y))  
 leftmost\_point, \*\_, rightmost\_point = sorted(points, key=lambda lst: lst[0])  
 x1, y1 = leftmost\_point  
 x2, y2 = rightmost\_point  
  
 border\_slope = (y2 - y1) / (x2 - x1)  
  
 points\_below\_border = [(x1, y1)]  
 points\_above\_border = [(x2, y2)]  
  
 for x, y in points:  
 border = border\_slope \* (x - x1) + y1  
 if y < border:  
 points\_below\_border.append((x, y))  
 elif y > border:  
 points\_above\_border.append((x, y))  
  
 points\_below\_border.sort()  
 points\_above\_border.sort(reverse=True)  
  
 points\_above\_border.append((x1, y1))  
  
 merged\_points = points\_below\_border + points\_above\_border  
  
 x\_lst = [x for x, \_ in merged\_points]  
 y\_lst = [y for \_, y in merged\_points]  
 plt.plot(x\_lst, y\_lst, c='black')  
  
 return merged\_points  
  
  
def findLength(merged\_points):  
 x0 = merged\_points[-1][0]  
 y0 = merged\_points[-1][1]  
 l = 0  
 for x1, y1 in merged\_points:  
 l += np.sqrt((x1 - x0) \*\* 2 + (y1 - y0) \*\* 2)  
 x0, y0 = x1, y1  
 return l  
  
  
plt.rcParams.update({'figure.figsize': (6, 6)})  
cmap = plt.get\_cmap('hsv')  
  
inicialisation()  
  
l = [elem for elem in list(chain(\*arr\_point)) if elem.f is not None]  
  
repeat(epsilon)  
  
plt.scatter([elem.x for elem in l], [elem.y for elem in l], s=size, cmap=cmap, c=[elem.f for elem in l])  
  
*# точки эквипотенциали*find(f)  
*# упорядочиваем точки и находим длину*l = findLength(sort())  
print("Длина заданной эквипотенциали = ", l)  
  
plt.show()