Санкт-Петербургский Государственный Электротехнический Университет "ЛЭТИ"

кафедра физики

Задание №2 по дисциплине

"Физические основы информационных технологий"

Название: Численное решение уравнения Лапласа

|  |  |
| --- | --- |
| Фамилия И.О.: | Новак П.И. |
| 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

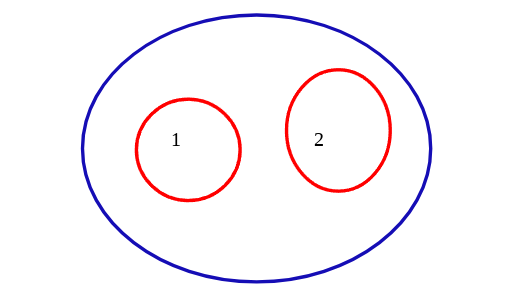


Рисунок 1. Пример электростатической системы

Исходные данные

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Вар. | Уравнение внешнего электрода | Уравнение электрода 1 | Уравнение электрода 2 | Потенциал искомой эквипотенциали, В | Потенциал на электроде 1, В | Потенциал на электроде 1, В |
| 23 | x^2 + y^2 = 25 | 0.8\*Abs[1.8 + x]^2.5 + 0.8\*Abs[y]^2.5 = 0.6 | 0.5\*Abs[-1.8 + x]^3 + 0.8\*Abs[y]^3 = 0.5 | -2 | -5 | 6 |

Основные теоретические положения

Алгоритм построения правильного распределения потенциалов магнитного поля:

* Пространство внутри внешнего электрода и между внутренними электродами разбивается сеткой
* Узлы сетки, попавшие на границы электродов, имеют потенциал соответствующего электрода. Потенциал внешнего электрода равен 0.

Остальные узлы пока имеют случайный потенциал в диапазоне Ф1 ― Ф2

* Для каждого не граничного узла высчитывается новое значение среднее арифметическое соседних узлов (сверху, снизу, слева, справа).

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.

Вывод

С помощью Python было решено уравнение Лапласа, построены эквипотенциали для электростатической системы из 3 электродов, найдена эквипотенциаль с заданным потенциалом и посчитана ее длина.

**Приложение A**

**Программа main.py**

import random  
import numpy as np  
import matplotlib.pyplot as plt  
  
potential\_blue = 0  
potential1 = -5  
potential2 = 6  
potential\_find = -2  
  
class Nest:  
 coordsY = []  
 coordsX = []  
 step = 0.1  
 size = 10  
 lines = int(size // step) + 1  
 nest = [[] for \_ in range(lines)]  
  
def electrode0(x, y):  
 return pow(x,2) + pow(y,2)-25  
  
  
def electrode1(x, y):  
 return 0.8\*pow(np.abs(1.8 + x), 2.5) + 0.8 \* pow(np.abs(y),2.5) - 0.6  
  
  
def electrode2(x, y):  
 return 0.5\*pow(np.abs(-1.8 + x),3) + 0.8 \* pow(np.abs(y),3) - 0.5  
  
def pointPosition(nest, x, y):  
 if -5 \* nest.step <= electrode0(x, y) <= 0:  
 return 0  
 if electrode0(x, y) > 0:  
 return -2  
 if electrode1(x, y) <= 0:  
 return 1  
 if electrode2(x, y) <= 0:  
 return 2  
 else:  
 return -1  
  
def fillNest(nest):  
 for x in range(0, nest.lines):  
 for y in range(0, nest.lines):  
 nest.nest[x].append(None)  
 x\_new = -5 + y \* nest.step  
 y\_new = -5 + x \* nest.step  
 position = pointPosition(nest, x\_new, y\_new)  
  
 if (position!=-1):  
 if position == 0:  
 nest.nest[x][y] = potential\_blue  
 elif position == 1:  
 nest.nest[x][y] = potential1  
 elif position == 2:  
 nest.nest[x][y] = potential2  
 elif position == -1:  
 nest.nest[x][y] = random.random() \* ((potential\_blue + potential1 + potential2) / 3)  
 else:  
 nest.nest[x][y] = None  
  
  
  
def checkNeighbours(nest, count):  
 for i in range(count):  
 for y in range(1, nest.lines - 1):  
 for x in range(1, nest.lines - 1):  
 x\_new = -5 + x \* nest.step  
 y\_new = -5 + y \* nest.step  
 if pointPosition(nest, x\_new, y\_new) == -1:  
 neighbours = [nest.nest[y - 1][x],  
 nest.nest[y + 1][x],  
 nest.nest[y][x - 1],  
 nest.nest[y][x + 1]]  
 neighbours = [value for value in neighbours if value is not None]  
  
 nest.nest[y][x] = sum(neighbours) / len(neighbours)  
  
  
def coordsRemake(nest, coord):  
 return -(nest.size / 2) + coord \* nest.step  
  
  
def intersection(point1, point2):  
 x1, y1, z1 = point1  
 x2, y2, z2 = point2  
 if z1 == potential\_find:  
 return point1  
 elif z2 == potential\_find:  
 return point2  
 if z1 == z2:  
 return None  
  
 level = (potential\_find - z1) / (z2 - z1)  
 if 0 <= level <= 1:  
 x = x1 + level \* (x2 - x1)  
 y = y1 + level \* (y2 - y1)  
 z = potential\_find  
 return x, y, z  
  
 return None  
  
def drawElectrode():  
 xCoords = np.linspace(-5, 5, 400)  
 yCoords = np.linspace(-5, 5, 400)  
 X, Y = np.meshgrid(xCoords, yCoords)  
  
 electr0\_z = electrode0(X, Y)  
 electr1\_z = electrode1(X, Y)  
 electr2\_z = electrode2(X, Y)  
  
 plt.contour(X, Y, electr1\_z, levels=[0], colors='g')  
 plt.contour(X, Y, electr2\_z, levels=[0], colors='g')  
 plt.contour(X, Y, electr0\_z, levels=[0], colors='g')  
  
def calcLenLineInTriangle(nest, triangle):  
 length = 0  
 coord = [intersection(triangle[0], triangle[1]), intersection(triangle[0], triangle[2]),  
 intersection(triangle[1], triangle[2])]  
 coord = [point for point in coord if point is not None]  
  
 if len(coord) == 2:  
 length += ((coord[1][0] - coord[0][0]) \*\* 2 + (coord[1][1] - coord[0][1]) \*\* 2 + (coord[1][2] - coord[0][2]) \*\* 2)\*\*0.5  
 nest.coordsX.append(coord[0][0])  
 nest.coordsX.append(coord[1][0])  
 nest.coordsY.append(coord[0][1])  
 nest.coordsY.append(coord[1][1])  
 plt.plot([nest.coordsX[-2], nest.coordsX[-1]], [nest.coordsY[-2], nest.coordsY[-1]], color="red")  
  
 return length  
  
  
def findLength(nest):  
 length = 0  
 for y in range(0, nest.lines - 1):  
 for x in range(0, nest.lines - 1):  
 if (nest.nest[y][x] is None or nest.nest[y + 1][x]  
 is None or nest.nest[y][x + 1] is None or nest.nest[y + 1][x + 1] is None):  
 continue  
  
 triangleLow = [[coordsRemake(nest, x), coordsRemake(nest, y), nest.nest[y][x]],  
 [coordsRemake(nest, x + 1), coordsRemake(nest, y), nest.nest[y][x + 1]],  
 [coordsRemake(nest, x), coordsRemake(nest, y + 1), nest.nest[y + 1][x]]]  
 triangleTop = [[coordsRemake(nest, x), coordsRemake(nest, y + 1), nest.nest[y + 1][x]],  
 [coordsRemake(nest, x + 1), coordsRemake(nest, y), nest.nest[y][x + 1]],  
 [coordsRemake(nest, x + 1), coordsRemake(nest, y + 1), nest.nest[y + 1][x + 1]]]  
  
 if max(triangleTop[0][2],  
 triangleTop[1][2],  
 triangleTop[2][2]) > potential\_find > min(triangleTop[0][2],  
 triangleTop[1][2],  
 triangleTop[2][2]):  
 length += calcLenLineInTriangle(nest, triangleTop)  
  
 if max(triangleLow[0][2],  
 triangleLow[1][2],  
 triangleLow[2][2]) > potential\_find > min(triangleLow[0][2],  
 triangleLow[1][2],  
 triangleLow[2][2]):  
 length += calcLenLineInTriangle(nest, triangleLow)  
  
 return length  
  
  
def main():  
 nest = Nest()  
 plt.grid(False)  
 plt.axis('equal')  
 drawElectrode()  
 fillNest(nest)  
 checkNeighbours(nest, 1000)  
 print(findLength(nest))  
 plt.show()  
  
main()