Санкт-Петербургский Государственный Электротехнический Университет "ЛЭТИ"

кафедра физики

Задание №2 по дисциплине

"Физические основы информационных технологий"

Название: Численное решение уравнения Лапласа

|  |  |
| --- | --- |
| Фамилия И.О.: | Иванов А. С. |
| 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

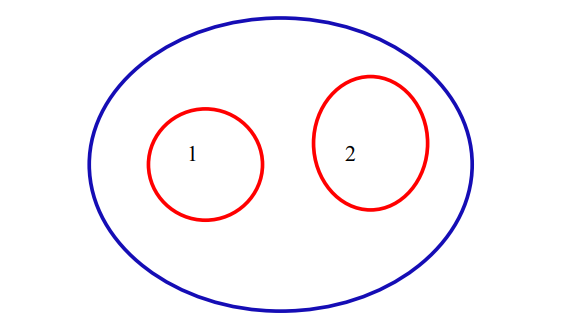


Рисунок 1. Пример электростатической системы

Исходные данные

ВАР 5



Основные теоретические положения

1. Закон Суперпозиции: Закон Суперпозиции гласит, что потенциал в точке, создаваемый несколькими заряженными объектами, равен сумме потенциалов, создаваемых каждым объектом независимо. Это позволяет учесть влияние всех заряженных электродов на потенциал в данной точке.
2. Потенциальная энергия и работа: Разница потенциалов между двумя точками в пространстве определяет работу, которую необходимо совершить, чтобы переместить заряд между этими точками. Работа равна разнице потенциальных энергий.
3. Эквипотенциальные поверхности: Эквипотенциальные линии и поверхности представляют собой места, на которых потенциал одинаков. Они всегда перпендикулярны силовым линиям электрического поля.
4. Закон Пуассона: Закон Пуассона описывает распределение электрического потенциала в пространстве, создаваемое распределением плотности заряда. Этот закон позволяет вычислять потенциал в разных точках пространства на основе распределения заряда.
5. Метод численных решений: Для нахождения эквипотенциальных линий и потенциала в сложных геометриях, когда аналитическое решение невозможно, часто используют численные методы, такие как метод конечных разностей или метод Монте-Карло.

**ПРИЛОЖЕНИЕ А**

**ФАЙЛ main.py**

import matplotlib  
matplotlib.use('TkAgg')  
import random  
import numpy as np  
import matplotlib.pyplot as plt  
  
  
class Point:  
 def \_\_init\_\_(self, x, y, f):  
 self.x = x  
 self.y = y  
 self.f = f  
  
  
class EquiPotentialCalculator:  
 def \_\_init\_\_(self, f1, f2, f, step):  
 self.f1 = f1  
 self.f2 = f2  
 self.f = f  
 self.step = step  
 self.arr\_x = []  
 self.arr\_y = []  
 self.arr\_point = []  
 self.partition = np.arange(-5, 5, step)  
  
 def areInConstraints(self, x, y):  
 res = x \*\* 2 + y \*\* 2 <= 25  
 res1 = abs(1.5 + x) \*\* 1.5 + abs(-1.5 + y) \*\* 1.5 >= 0.6  
 res2 = abs(-1.5 + x) \*\* 3 + 0.8 \* abs(1.5 + y) \*\* 3 >= 0.5  
 return res and res1 and res2  
  
 def initializePoints(self):  
 for x in self.partition:  
 innerList = []  
 for y in self.partition:  
 f = None  
 if self.areInConstraints(x, y):  
 circleConstraint = abs(x \*\* 2 + y \*\* 2 - 25)  
 constraint1 = abs(abs(1.5 + x) \*\* 1.5 + abs(-1.5 + y) \*\* 1.5 - 0.6)  
 constraint2 = abs(abs(1.5 + x) \*\* 1.5 + abs(-1.5 + y) \*\* 1.5 - 0.5)  
  
 if circleConstraint <= 1:  
 f = 0  
 elif constraint1 <= 0.5:  
 f = self.f1  
 elif constraint2 <= 0.5:  
 f = self.f2  
 else:  
 f = random.uniform(min(self.f1, self.f2), max(self.f1, self.f2))  
  
 point = Point(x, y, f)  
 innerList.append(point)  
 self.arr\_point.append(innerList)  
  
 def iteratePoints(self):  
 n = 0  
 while n <= 500:  
 for y in range(1, len(self.arr\_point) - 1):  
 for x in range(1, len(self.arr\_point[y]) - 1):  
 neighbors = [  
 self.arr\_point[y][x - 1].f,  
 self.arr\_point[y][x + 1].f,  
 self.arr\_point[y - 1][x].f,  
 self.arr\_point[y + 1][x].f  
 ]  
 noNoneFound = all(neighbor is not None for neighbor in neighbors)  
 if not noNoneFound:  
 continue  
 self.arr\_point[y][x].f = sum(neighbors) / 4  
 n += 1  
  
 def calculateNewPoint(self, tmp, neighbor, flag):  
 if flag == 'x':  
 new\_x = tmp.x + (abs(self.f - tmp.f) / abs(tmp.f - neighbor.f)) \* self.step  
 new\_y = tmp.y  
 else:  
 new\_x = tmp.x  
 new\_y = tmp.y + (abs(self.f - tmp.f) / abs(tmp.f - neighbor.f)) \* self.step  
 self.arr\_x.append(new\_x)  
 self.arr\_y.append(new\_y)  
  
 def processPotentialEquilibrium(self, fValue):  
 for i in range(1, len(self.arr\_point) - 1):  
 for j in range(1, len(self.arr\_point[i]) - 1):  
 current = self.arr\_point[i][j]  
 rightNeighbor = self.arr\_point[i + 1][j]  
 downNeighbor = self.arr\_point[i][j + 1]  
 if (current.f is None or rightNeighbor.f is None or downNeighbor.f is None):  
 continue  
 if (current.f <= fValue <= rightNeighbor.f or current.f >= fValue >= rightNeighbor.f):  
 self.calculateNewPoint(current, rightNeighbor, 'x')  
  
 if (current.f <= fValue <= downNeighbor.f or current.f >= fValue >= downNeighbor.f):  
 self.calculateNewPoint(current, downNeighbor, 'y')  
  
 def getResultingPoints(self):  
 points = [(x, y) for x, y in zip(self.arr\_x, self.arr\_y)]  
 sortedPoints = sorted(points, key=lambda lst: lst[0])  
 leftPoint = sortedPoints[0]  
 rightPoint = sortedPoints[-1]  
 x1, y1 = leftPoint  
 x2, y2 = rightPoint  
  
 slope = (y2 - y1) / (x2 - x1)  
  
 belowBorder = [(x1, y1)]  
 aboveBorder = [(x2, y2)]  
  
 for x, y in points:  
 border = slope \* (x - x1) + y1  
 if y < border:  
 belowBorder.append((x, y))  
 elif y > border:  
 aboveBorder.append((x, y))  
  
 belowBorder.sort()  
 aboveBorder.sort(reverse=True)  
  
 aboveBorder.append((x1, y1))  
  
 resultPoints = belowBorder + aboveBorder  
  
 xValues, yValues = zip(\*resultPoints)  
  
 plt.plot(xValues, yValues, c='red')  
  
 return resultPoints  
  
 def calculateLength(self, resultPoints):  
 length = 0  
 for i in range(1, len(resultPoints)):  
 x0, y0 = resultPoints[i - 1]  
 x1, y1 = resultPoints[i]  
 length += np.sqrt((x1 - x0) \*\* 2 + (y1 - y0) \*\* 2)  
 return length  
  
 def performPotentialEquilibrium(self):  
 self.initializePoints()  
 self.iteratePoints()  
  
 arrList = []  
 for row in self.arr\_point:  
 for el in row:  
 if el.f is not None:  
 arrList.append(el)  
  
 plt.scatter([el.x for el in arrList], [el.y for el in arrList], c='blue')  
  
 self.processPotentialEquilibrium(self.f)  
  
 length = self.calculateLength(self.getResultingPoints())  
  
 print(f"Desired length: {length}")  
 plt.show()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 calculator = EquiPotentialCalculator(f1=5, f2=5, f=4, step=0.1)  
 calculator.performPotentialEquilibrium()