Санкт-Петербургский Государственный Электротехнический Университет "ЛЭТИ"

кафедра физики

Задание №2 по дисциплине

"Физические основы информационных технологий"

Название: Численное решение уравнения Лапласа.

|  |  |
| --- | --- |
| Фамилия И.О.: | Иевлев Е.А. |
| 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

1

2

Рисунок 1. Пример электростатической системы

**Таблица с исходными данными**

Вариант 18

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 | Abs[-1.8 + x]^2 + 0.8\*Abs[y]^2 = 0.6 | Abs[1.8 + x]^4 + 0.5\*Abs[y]^4 = 0.8 | 1 | 6 | -5 |

**Основные теоретические положения**

Алгоритм построения правильного распределения потенциалов магнитного поля (метод сеток):

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. Модель поля представлена на рисунке снизу. Красной линией выделена искомая эквипотенциаль. Длина оной 15.852496.

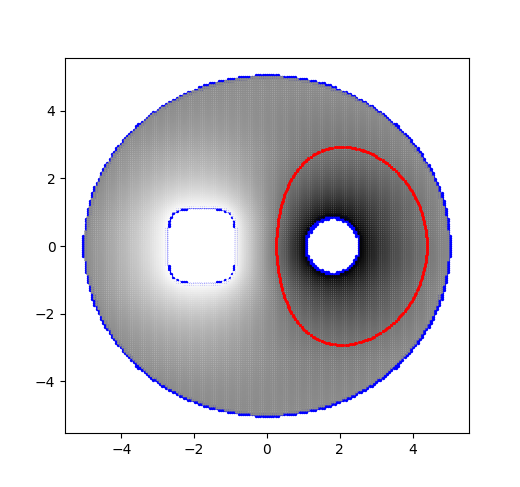


Рисунок 2. Полученная модель поля.

**Вывод**

Была написана программа, реализующая метод сеток, рисующая модель поля и рассчитывающая длину искомой эквипотенциали по заданному значению потенциала.

**ПРИЛОЖЕНИЕ А**

**ПРОГРАММА SRC\_LAB2.PY**

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import fsolve

import random

# 18'th variant

# внешний электрод x^2 + y^2 = 25

# уравнение электрода (1) abs(-1.8 + x)^2 + 0.8 \* abs(y)^2 = 0.6

# уравнение электрода (2) abs(1.8 + x)^4 + 0.5 \* abs(y)^4 = 0.8

# потенциал искомой эквипотенциали 1В

# потенциал на электроде (1) 6В

# потенциал на электроде (2) -5В

extV = 0

firstV = 6

secondV = -5

lineV = 1

def extElEquation(x, y):

result = pow(x, 2) + pow(y, 2)

return result - 25

def firstElEquation(x, y):

result = pow(np.abs(-1.8 + x), 2) + 0.8 \* pow(np.abs(y), 2)

return result - 0.6

def secondElEquation(x, y):

result = pow(np.abs(1.8 + x), 4) + 0.5 \* pow(np.abs(y), 4)

return result - 0.8

class Vpoint:

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

self.x = x

self.y = y

self.V = V

self.needed = True

pointsPerSide = 200

squareRadius = 6

points = [[Vpoint(-squareRadius + 2 \* squareRadius \* x / pointsPerSide,

-squareRadius + 2 \* squareRadius \* y / pointsPerSide,

firstV + random.random() \* (secondV - firstV)) for y in range(pointsPerSide)] for x in

range(pointsPerSide)]

visibleX = []

visibleY = []

# marking edges

for x in range(pointsPerSide):

for y in range(pointsPerSide):

result = extElEquation(points[x][y].x, points[x][y].y)

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

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

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

if result >= 0:

points[x][y].V = extV

points[x][y].needed = False

continue

result = firstElEquation(points[x][y].x, points[x][y].y)

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

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

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

if result <= 0:

points[x][y].V = firstV

points[x][y].needed = False

continue

result = secondElEquation(points[x][y].x, points[x][y].y)

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

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

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

if result <= 0:

points[x][y].V = secondV

points[x][y].needed = False

continue

plt.scatter(visibleX, visibleY, c="blue", s=2)

Vprecision = 0.001

def iterateV():

newV = [[0 for y in range(pointsPerSide)] for x in range(pointsPerSide)]

for x in range(pointsPerSide):

for y in range(pointsPerSide):

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

newV[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(pointsPerSide):

for y in range(pointsPerSide):

if points[x][y].needed:

if np.abs(points[x][y].V - newV[x][y]) > Vprecision:

isLastIteration = False

points[x][y].V = newV[x][y]

if not isLastIteration:

return True

else:

return False

a = True

while (a):

a = iterateV()

def getColor(v):

x = (v - firstV) / (secondV - firstV)

return [x, x, x]

Xx = []

Yy = []

colors = []

for x in range(pointsPerSide):

for y in range(pointsPerSide):

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

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

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

colors.append(getColor(points[x][y].V))

plt.scatter(Xx, Yy, c=colors, s=1)

eq\_list = []

def f(variables):

(a, k1, k2) = variables

res = []

for eq in eq\_list:

res.append(eval(eq))

return res

def getLengthOnSquare(LTPoint, RTPoint, LBPoint, RBPoint):

global eq\_list

points = []

eq\_list = [

"a + k1 \*" + str(LTPoint.x) + "+ k2 \* " + str(LTPoint.y) + " - " + str(LTPoint.V),

"a + k1 \*" + str(RTPoint.x) + "+ k2 \* " + str(RTPoint.y) + " - " + str(RTPoint.V),

"a + k1 \*" + str(LBPoint.x) + "+ k2 \* " + str(LBPoint.y) + " - " + str(LBPoint.V)

]

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

xT = (lineV - a - k2 \* LTPoint.y) / k1

if (xT > LTPoint.x and xT < RTPoint.x):

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

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

yL = (lineV - a - k1 \* LTPoint.x) / k2

if (yL > LBPoint.y and yL < LTPoint.y):

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

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

eq\_list = [

"a + k1 \*" + str(RBPoint.x) + "+ k2 \* " + str(RBPoint.y) + " - " + str(RBPoint.V),

"a + k1 \*" + str(RTPoint.x) + "+ k2 \* " + str(RTPoint.y) + " - " + str(RTPoint.V),

"a + k1 \*" + str(LBPoint.x) + "+ k2 \* " + str(LBPoint.y) + " - " + str(LBPoint.V)

]

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

xB = (lineV - a - k2 \* RBPoint.y) / k1

if (xB > LBPoint.x and xB < RBPoint.x):

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

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

yR = (lineV - a - k1 \* RBPoint.x) / k2

if (yR > RBPoint.y and yR < RTPoint.y):

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

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

print(len(points))

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(pointsPerSide):

for y in range(pointsPerSide):

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

if (points[x][y].V < lineV + 0.3 and

points[x][y].V > lineV - 0.3):

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

points[x + 1][y],

points[x][y - 1],

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

print(totalLength)

plt.show()