Санкт-Петербургский Государственный Электротехнический Университет "ЛЭТИ"

кафедра физики

Задание №2 по дисциплине

"Физические основы информационных технологий"

Название: Численное решение уравнения Лапласа.

|  |  |
| --- | --- |
| Фамилия И.О.: | Ягодаров М. 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

2

1

Рисунок 1. Пример электростатической системы

**Вариант 25**

Исходные данные:

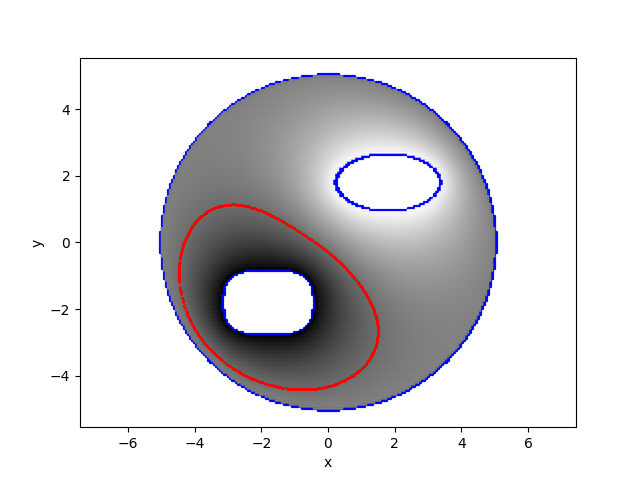
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Уравнение внешнего электрода | Уравнение электрода 1 | Уравнение электрода 2 | Потенциал искомой эквипотенциали, В | Потенциал на электроде 1, В | Потенциал на электроде 2, В |
| x^2 + y^2 = 25 | 0.3\*Abs[1.8 + x]^3 + 0.8\*Abs[1.8 + y]^3 = 0.8 | 0.3\*Abs[-1.8 + x]^2 + Abs[-1.8 + y]^2 = 0.8 | 1 | 6 | -6 |

**Выполнение работы**

Разбиваем на сетку квадрат, в который помещается внешний электрод. Каждому узлу сетке присвоено значение в зависимости от расположения, если узел на внешнем электроде или в не его, то потенциал 0, если узел находится в/на электроде 1 или 2, то потенциал соответствует электроду, иначе потенциал узла случайное значение в [-6, 6).

После этого, запускается цикл, который на каждой итерации обновляет значение потенциала в узле, как среднее значение 4-х смежных с ним, если разница между потенциалом на узле на текущей итерации и прошлой итерации меньше, чем заданная точность, то цикл останавливается.

После этого для каждого узла, в котором потенциал примерно равен искомому выполняется следующее: пробуют найти две точки на границе квадрата (квадрат это 4 узла: текущий, справа, снизу, справа по диагонали) с искомым потенциалом, если таковые нашлись, то вычисляется расстояние между точками. Суммируя полученные расстояния получаем примерную длину искомой эквипотенциали.



**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАМЫ**

Файл main.py:

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import fsolve

import random

outer\_phi = 0

first\_phi = 6

second\_phi = -6

target\_phi = 1

def outer\_electrode\_contour(x, y):

lhs = np.power(x, 2) + np.power(y, 2)

return lhs - 25

def first\_electrode\_contour(x, y):

lhs = 0.3 \* np.power(np.abs(1.8 + x), 3) + 0.8 \* np.power(np.abs(1.8 + y), 3)

return lhs - 0.8

def second\_electrode\_contour(x, y):

lhs = 0.3 \* np.power(np.abs(-1.8 + x), 2) + np.power(np.abs(-1.8 + y), 2)

return lhs - 0.8

class PointPhi:

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

self.x = x

self.y = y

self.phi = phi

self.is\_active = True

def set\_phi(self, phi):

self.phi = phi

self.is\_active = False

points\_per\_side = 200

plot\_width = 6

def plot\_for(func: callable):

for x in range(points\_per\_side):

for y in range(points\_per\_side):

func(x, y)

def create\_point(x, y):

return PointPhi(-plot\_width + 2 \* plot\_width \* x / points\_per\_side,

-plot\_width + 2 \* plot\_width \* y / points\_per\_side,

first\_phi + random.random() \* (second\_phi - first\_phi))

# Make random potential distribution on the grid

points = [

[create\_point(x, y) for y in range(points\_per\_side)]

for x in range(points\_per\_side)

]

x\_grid\_points = np.linspace(-plot\_width, plot\_width, points\_per\_side)

y\_grid\_points = np.linspace(-plot\_width, plot\_width, points\_per\_side)

contours\_x = []

contours\_y = []

outer\_ellipsis\_width = 0.5

inner\_ellipsis\_width = 0.1

# draw electrode contours and set potentials

def draw\_contours(x, y):

result = outer\_electrode\_contour(points[x][y].x, points[x][y].y)

if np.abs(result) < outer\_ellipsis\_width:

contours\_x.append(points[x][y].x)

contours\_y.append(points[x][y].y)

if result >= 0:

points[x][y].set\_phi(outer\_phi)

return

result = first\_electrode\_contour(points[x][y].x, points[x][y].y)

if np.abs(result) < inner\_ellipsis\_width:

contours\_x.append(points[x][y].x)

contours\_y.append(points[x][y].y)

if result <= 0:

points[x][y].set\_phi(first\_phi)

return

result = second\_electrode\_contour(points[x][y].x, points[x][y].y)

if np.abs(result) < inner\_ellipsis\_width:

contours\_x.append(points[x][y].x)

contours\_y.append(points[x][y].y)

if result <= 0:

points[x][y].set\_phi(second\_phi)

return

# draw\_contours()

plot\_for(draw\_contours)

plt.xlabel('x')

plt.ylabel('y')

plt.axis('equal')

plt.scatter(contours\_x, contours\_y, c='blue', s=2)

calculation\_error = 0.001

# Average potential on the grid until it is precise enough

def recalculate\_phi():

global points

new\_phi = [[0 for \_ in range(points\_per\_side)] for \_ in range(points\_per\_side)]

is\_precise = True

def average\_phi(x, y):

global points

nonlocal new\_phi

if points[x][y].is\_active:

new\_phi[x][y] = ((points[x + 1][y].phi + points[x][y + 1].phi +

points[x - 1][y].phi + points[x][y - 1].phi) / 4)

def check\_precision(x, y):

global points

nonlocal is\_precise

if points[x][y].is\_active:

if np.abs(points[x][y].phi - new\_phi[x][y]) > calculation\_error:

is\_precise = False

points[x][y].phi = new\_phi[x][y]

plot\_for(average\_phi)

plot\_for(check\_precision)

return not is\_precise

while recalculate\_phi():

...

x\_points = []

y\_points = []

colors = []

def color\_points(x, y):

global points, x\_points, y\_points, colors

def get\_color(phi):

k = (phi - first\_phi) / (second\_phi - first\_phi)

return [k, k, k]

if points[x][y].is\_active:

x\_points.append(points[x][y].x)

y\_points.append(points[x][y].y)

colors.append(get\_color(points[x][y].phi))

plot\_for(color\_points)

plt.scatter(x\_points, y\_points, c=colors, s=1)

def get\_equation\_system(a: PointPhi, b: PointPhi, c: PointPhi):

return [

f'a + k1 \* {a.x} + k2 \* {a.y} - {a.phi}',

f'a + k1 \* {b.x} + k2 \* {b.y} - {b.phi}',

f'a + k1 \* {c.x} + k2 \* {c.y} - {c.phi}'

]

def get\_length\_on\_square(lt\_point, rt\_point, lb\_point, rb\_point):

equation\_system = []

def eval\_system(variables):

(a, k1, k2) = variables

res = []

for equation in equation\_system:

res.append(eval(equation))

return res

line\_points = []

equation\_system = get\_equation\_system(lb\_point, lt\_point, rt\_point)

(a, k1, k2) = fsolve(eval\_system, [1, 1, 1])

x\_t = (target\_phi - a - k2 \* lt\_point.y) / k1

if lt\_point.x < x\_t < rt\_point.x:

plt.scatter(x\_t, lt\_point.y, c='red', s=1)

line\_points.append((x\_t, lt\_point.y))

y\_l = (target\_phi - a - k1 \* lt\_point.x) / k2

if lb\_point.y < y\_l < lt\_point.y:

plt.scatter(lt\_point.x, y\_l, c='red', s=1)

line\_points.append((lt\_point.x, y\_l))

equation\_system = get\_equation\_system(rb\_point, lb\_point, rt\_point)

(a, k1, k2) = fsolve(eval\_system, [1, 1, 1])

x\_b = (target\_phi - a - k2 \* rb\_point.y) / k1

if lb\_point.x < x\_b < rb\_point.x:

plt.scatter(x\_b, rb\_point.y, c='red', s=1)

line\_points.append((x\_b, rb\_point.y))

y\_r = (target\_phi - a - k1 \* rb\_point.x) / k2

if rb\_point.y < y\_r < rt\_point.y:

plt.scatter(rb\_point.x, y\_r, c='red', s=1)

line\_points.append((rb\_point.x, y\_r))

if len(line\_points) == 2:

return np.hypot(

line\_points[0][0] - line\_points[1][0],

line\_points[0][1] - line\_points[1][1]

)

return 0

total\_length = 0

phi\_error = 0.1

def calculate\_length(x, y):

global total\_length, points

if points[x][y].is\_active:

if np.abs(points[x][y].phi - target\_phi) < phi\_error:

p1 = points[x][y]

p2 = points[x + 1][y]

p3 = points[x][y - 1]

p4 = points[x + 1][y - 1]

total\_length += get\_length\_on\_square(p1, p2, p3, p4)

plot\_for(calculate\_length)

print(total\_length)

plt.show()