Санкт-Петербургский Государственный Электротехнический Университет

«ЛЭТИ»

кафедра физики

Задание №2 по дисциплине

«Физические основы информационных технологий» Название: Численное решение уравнения Лапласа.

Фамилия И.О.: Токун Г.С. группа: 1303

Преподователь: Альтмарк А.М. Итоговый балл:

Крайник срок сдачи: 05.11.2023

Санкт-Петербург 2023

# Задание

Дана электростатическая система, состоящая из трех электродов. Внешний электрод обладает потенциалом 0 В. Внутренние электроды (на рисунке отме- чены красным цветом и пронуменорованы как 1 и 2) обладают потенциалами, отличными от 0. Исходные данные нужно взять в файле FOIT\_IDZ2.xlsx. Для одной из указанных таблиц эквипотенциальных линий необходимо найти дли- ну и записать её в IDZ2.txt. Контуры электродов можно построить по фор- мулам, указанным в таблице и сравнить с соответствующим изображением в jpeg-файле. Координаты в данном задании считать безразмерными.

Помимо текстового файла IDZ2.txt в папке IDZ2 должен находиться Word- файл с отчетом, а также файл с кодом.

# Основные теоритеческие положения

Для нахождения значения потенциала значения потенциала в зависимости от удаленности требуется решить следующее уравнение:

∆*ϕ* = 0

Поле представим в виде сетки с индексами *i, j*. Тогда имеют место быть сле- дующее выражение:

=

∆*ϕ δx*

∆*ϕ δy*

*δ*2*ϕ*

*ϕi*+1*,j−ϕi,j h*

*ϕi,j*+1*−ϕi,j h*

=

*ϕi*+1*,j−ϕi,j −ϕi,j−ϕi−*1*,j*

*δx*2 =

*h h*

*h*

(1)

*δ*2*ϕ δy*2

*ϕi,j*+1*−ϕi,j ϕi,j−ϕi,j−*1

*h h*

=

*—*

*h*

(2)

∆*ϕ* = (1) + (2) = 0 *<*=*> ϕi,j* = *ϕi−*1*,j* +*ϕi*+1*,j* +*ϕi,j−*1+*ϕi,j*+1

4

(3)

Алгоритм решения уравнения Лапласа:

1. Генерируем начальные значений *ϕ* в промежутке от минимального значения потенциала электрода до максимального.
2. Для каждого узла сетки вычисляем значение потенциала по формуле (3).
3. Повторяем предыдущий шаг до тех пор, пока для каждого узла сетки не выполнится условие:

*|ϕs−*1 *− ϕs*

*| ≤ ϵ*

# Выполнение работы

1. Нарисуем электроды.

*i,j*

*i,j*

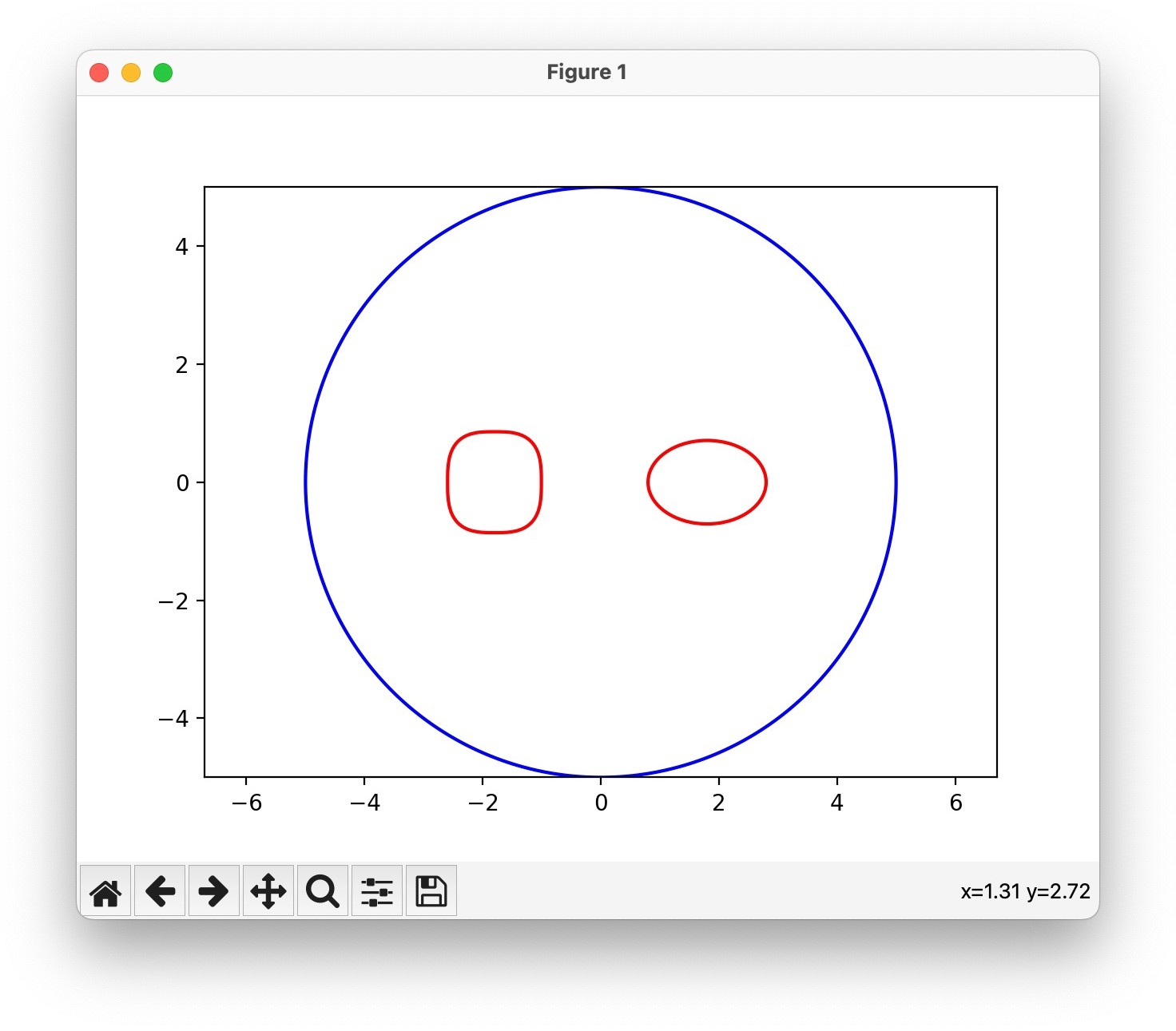


Рис. 1: Получившиеся электроды

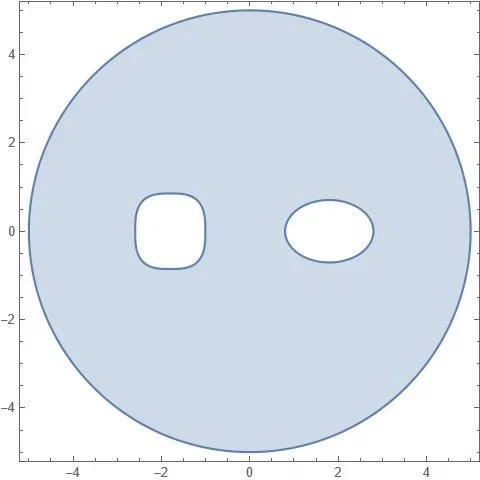


Рис. 2: Нужный результат

1. Заполняем сетку случайными значениями потенциала. Зеленые - ближе к

-6, красные ближе к 5.

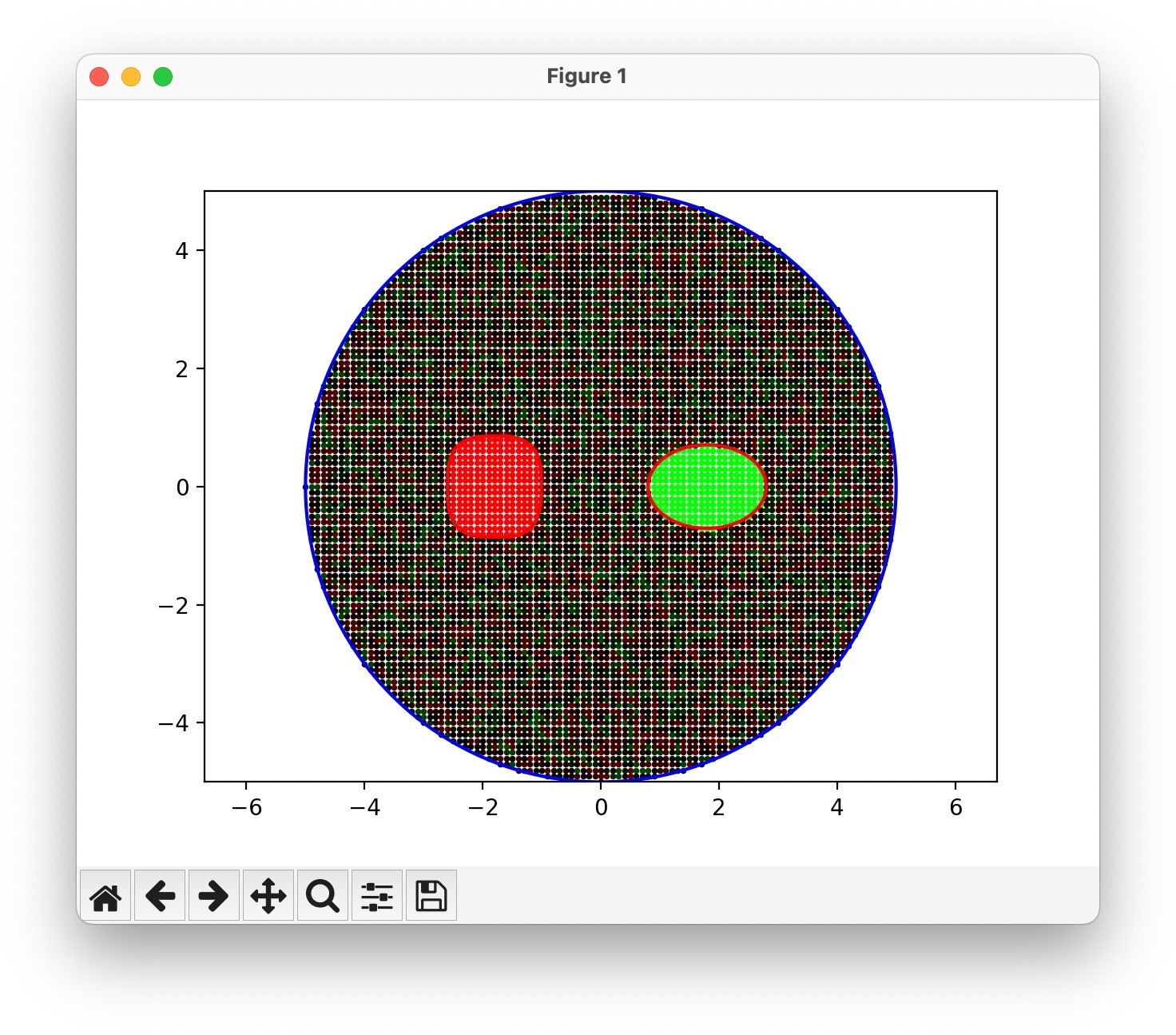


Рис. 3: Случайные значения потенциала

1. Сделаем перерасчет потенциалов по формуле 3.

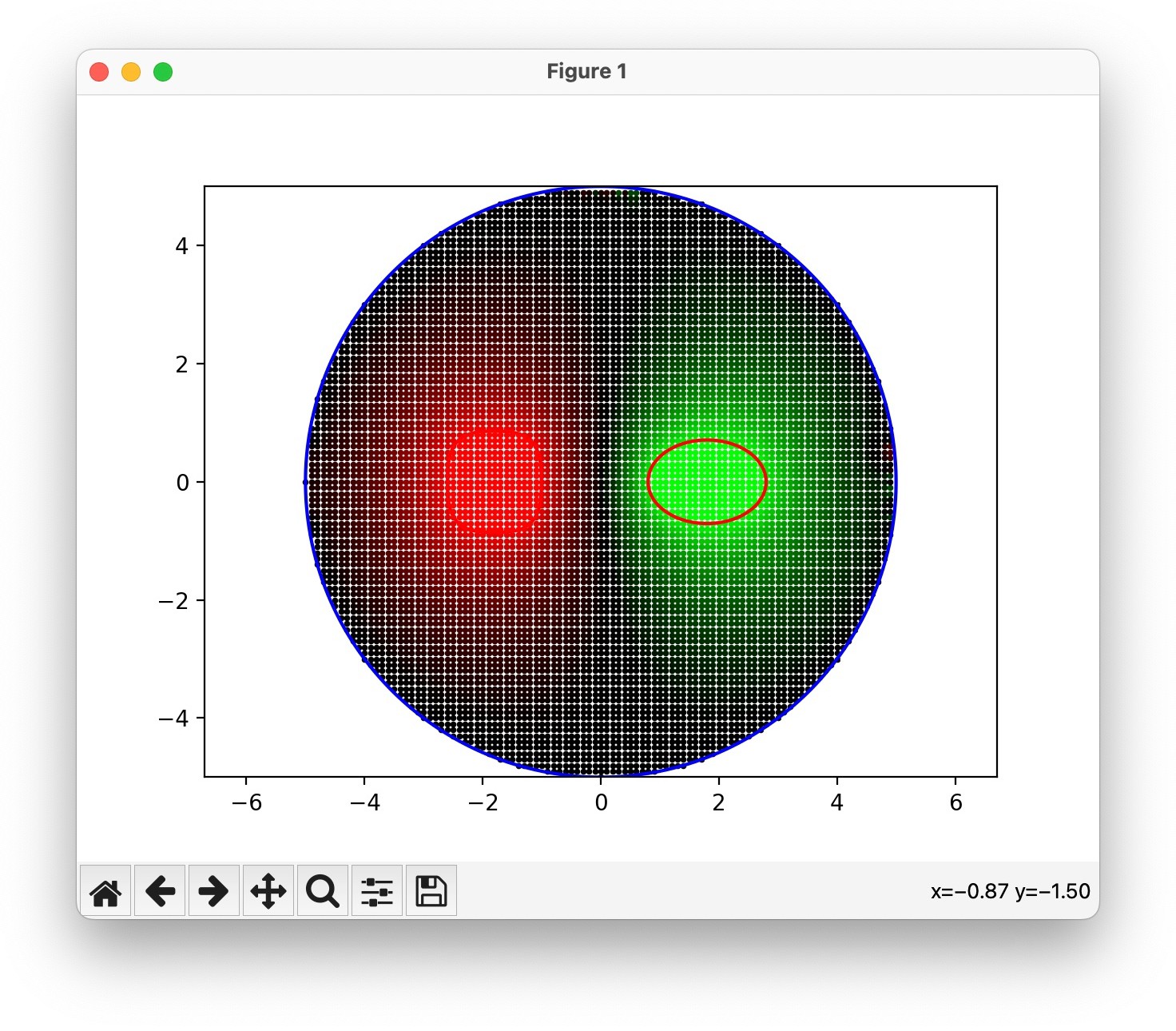


Рис. 4: Подсчитанные потенциалы

1. Нарисуем эквипотенциальную линию.

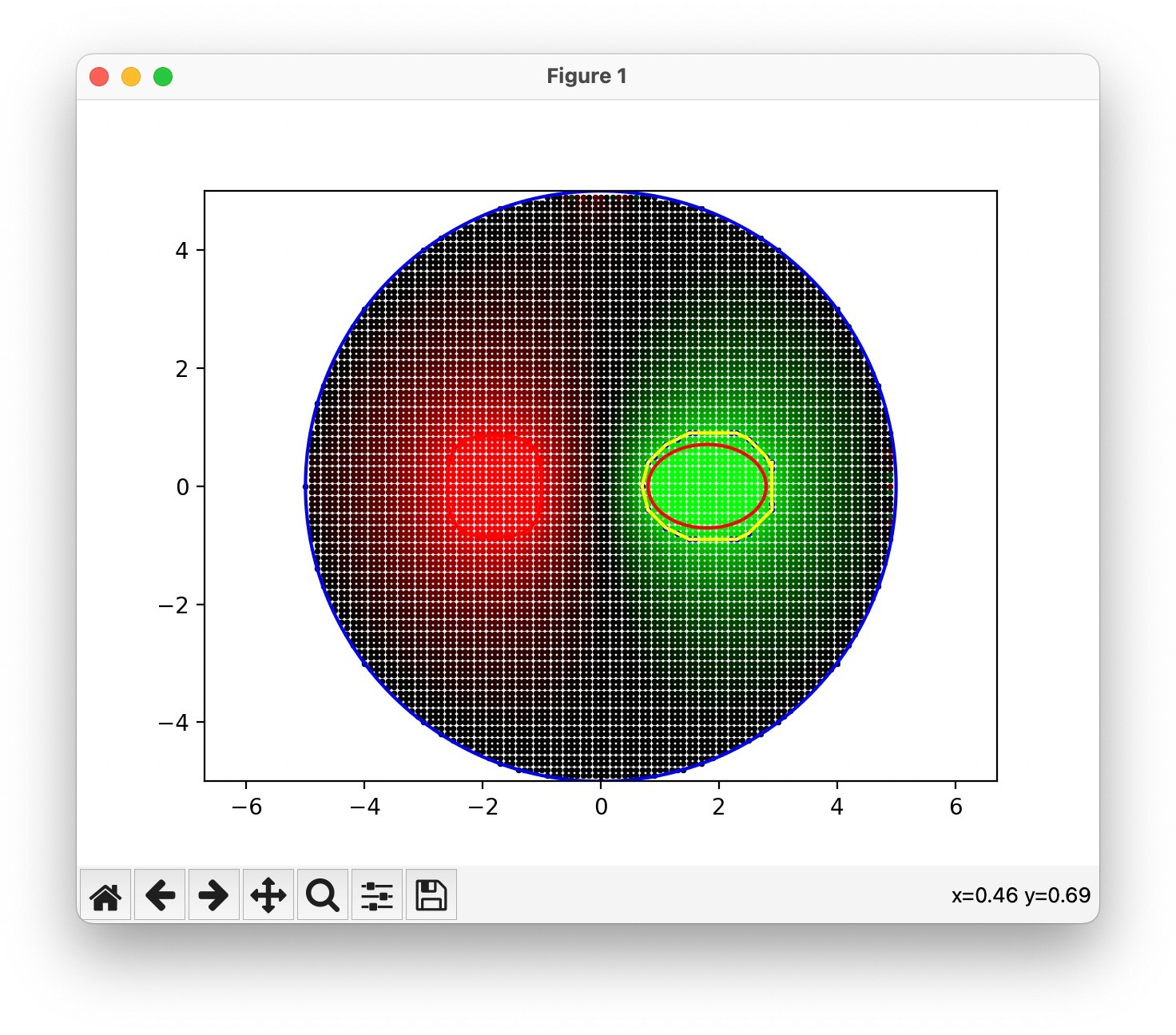


Рис. 5: Результат

# Приложение

Исходный код программы

import numpy as np  
from random import random, randint  
import matplotlib.pyplot as plt  
p\_find = 4  
p1 = -6  
p2 = 5  
eps = 0.1  
  
  
def electrode\_ext(x, y):  
 return x\*\*2 + y\*\*2 - 25  
  
  
def electrode1(x, y):  
 return abs(1.8 + x)\*\*3 + 0.8 \* abs(y)\*\*3 - 0.5  
  
  
def electrode2(x, y):  
 return 0.5 \* abs(-1.8 + x)\*\*2 + abs(y)\*\*2 - 0.5  
  
  
def makeElectrode():  
 x = np.linspace(-5, 5, 400)  
 y = np.linspace(-5, 5, 400)  
 X, Y = np.meshgrid(x, y)  
  
 coords1 = electrode1(X, Y)  
 coords2 = electrode2(X, Y)  
 coords\_ext = electrode\_ext(X, Y)  
 plt.contour(X, Y, coords1, levels=[0], colors='red')  
 plt.contour(X, Y, coords2, levels=[0], colors='red')  
 plt.contour(X, Y, coords\_ext, levels=[0], colors='blue')  
  
  
def findPos(x, y):  
 if -5 \* accuracy <= electrode\_ext(x, y) <= 0:  
 return 0 # на внешнем электроде  
 if electrode\_ext(x, y) > 0:  
 return -2 # вне области  
 if electrode1(x, y) <= 0:  
 return 1 # 1 электрод  
 if electrode2(x, y) <= 0:  
 return 2 # 2 электрод  
 else:  
 return -1 # внешний электрода  
  
  
def makeStartValues():  
 for height in range(0, countLines):  
 grid[height] = []  
 for width in range(0, countLines):  
 grid[height].append(None)  
 x = -5 + width \* accuracy  
 y = -5 + height \* accuracy  
 position = findPos(x, y)  
 if position != -1:  
 if position == 0:  
 grid[height][width] = 0  
 elif position == 1:  
 grid[height][width] = p1  
 elif position == 2:  
 grid[height][width] = p2  
 elif position == -1:  
 influence = randint(1, 10) / 10  
 grid[height][width] = (  
 (0 + influence \* p1 + (1 - influence) \* p2)) / 3  
  
  
def getColor(value):  
 if value >= 0:  
 green = int(value \* 255 / 5)  
 color\_hex = "#{:02x}{:02x}{:02x}".format(0, green, 0)  
 else:  
 red = int(abs(value) \* 255 / 6)  
 color\_hex = "#{:02x}{:02x}{:02x}".format(red, 0, 0)  
 return color\_hex  
  
  
def makeGrid():  
 for height in range(0, countLines):  
 for width in range(0, countLines):  
 x = -5 + width \* accuracy  
 y = -5 + height \* accuracy  
 if grid[height][width] is not None:  
 plt.scatter(x, y, color=getColor(grid[height][width]), s=3)  
 else:  
 plt.scatter(x, y, color="white", s=3)  
  
  
def setPotentialValues(quantity=1000):  
 for i in range(quantity):  
 for height in range(1, countLines - 1):  
 for width in range(1, countLines - 1):  
 x = -5 + width \* accuracy  
 y = -5 + height \* accuracy  
 if findPos(x, y) == -1:  
 neighbours = [  
 grid[height-1][width],  
 grid[height+1][width],  
 grid[height][width-1],  
 grid[height][width+1]  
 ]  
 neighbours = [\_ for \_ in neighbours if \_ is not None]  
  
 grid[height][width] = sum(neighbours) / len(neighbours)  
  
  
def coordPythagoras(x1, x2, y1, y2):  
 return ((x2 - x1)\*\*2 + (y2 - y1)\*\*2)\*\*0.5  
  
  
def calculateLen(traceX, traceY):  
 res = 0  
 for i in range(len(traceX) - 1):  
 res += coordPythagoras(traceX[i], traceX[i+1],  
 traceY[i], traceY[i+1])  
 return res  
  
  
def drawPotentialLine():  
 traceX = []  
 traceY = []  
 for width in range(0, countLines):  
 for height in range(0, countLines):  
 x = -5 + width \* accuracy  
 y = -5 + height \* accuracy  
 if (grid[height][width] is not None) and abs(grid[height][width] - p\_find) <= eps:  
 traceX.append(x)  
 traceY.append(y)  
 plt.scatter(x, y, color='blue', s=3)  
  
 traceY\_pos = []  
 traceX\_pos = []  
 traceY\_neg = []  
 traceX\_neg = []  
 for i in range(len(traceX)):  
 if traceY[i] < 0:  
 traceX\_neg.append(traceX[i])  
 traceY\_neg.append(traceY[i])  
 else:  
 traceX\_pos.append(traceX[i])  
 traceY\_pos.append(traceY[i])  
 temp = list(zip(traceX\_neg, traceY\_neg))  
 temp.sort()  
 traceX\_neg, traceY\_neg = zip(\*temp)  
 temp = list(zip(traceX\_pos, traceY\_pos))  
 temp.sort()  
 traceX\_pos, traceY\_pos = zip(\*temp)  
 length = calculateLen(traceX\_pos, traceY\_pos) + \  
 calculateLen(traceX\_neg, traceY\_neg) + coordPythagoras(  
 traceX\_neg[0], traceX\_pos[0], traceY\_neg[0], traceY\_pos[0]) + coordPythagoras(traceX\_neg[-1], traceX\_pos[-1], traceY\_neg[-1], traceY\_pos[-1])  
 print(length)  
 plt.plot(traceX\_pos, traceY\_pos, color='yellow')  
 plt.plot(traceX\_neg, traceY\_neg, color='yellow')  
 plt.plot([traceX\_neg[0], traceX\_pos[0]], [  
 traceY\_neg[0], traceY\_pos[0]], color='yellow')  
 plt.plot([traceX\_neg[-1], traceX\_pos[-1]],  
 [traceY\_neg[-1], traceY\_pos[-1]], color='yellow')  
  
  
makeElectrode()  
accuracy = 0.1  
graph\_size = 10  
countLines = int(graph\_size / accuracy)  
grid = [0] \* countLines  
makeStartValues()  
setPotentialValues()  
makeGrid()  
drawPotentialLine()  
plt.axis('equal')  
plt.grid(False)  
plt.show()