Учреждение образования
Белорусский Государственный Университет
Информатики и Радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

Отчет по лабораторной работе №2:

«Метод сеток решения задачи Дирихле для уравнения Пуассона»

Вариант №10

Выполнил:
ст. гр. 052002
Паньков Е.В.

Проверил:
Анисимов В.Я.

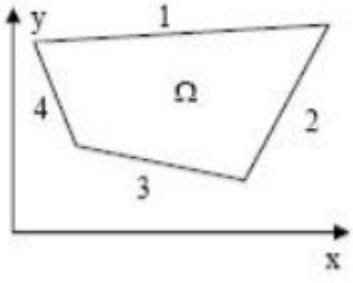
Минск 2012

Задание

Решить двухмерное стационарное уравнение Пуассона

$$\left(\frac{\partial}{\partial x}\left(g(x,y)\frac{\partial U}{\partial x}\right)+\frac{\partial^2 u}{\partial y^2}\right)=f(x,y)$$

Область представляет собой четырехугольник с координатами (8;8), (7;1), (2;2), (1;7)

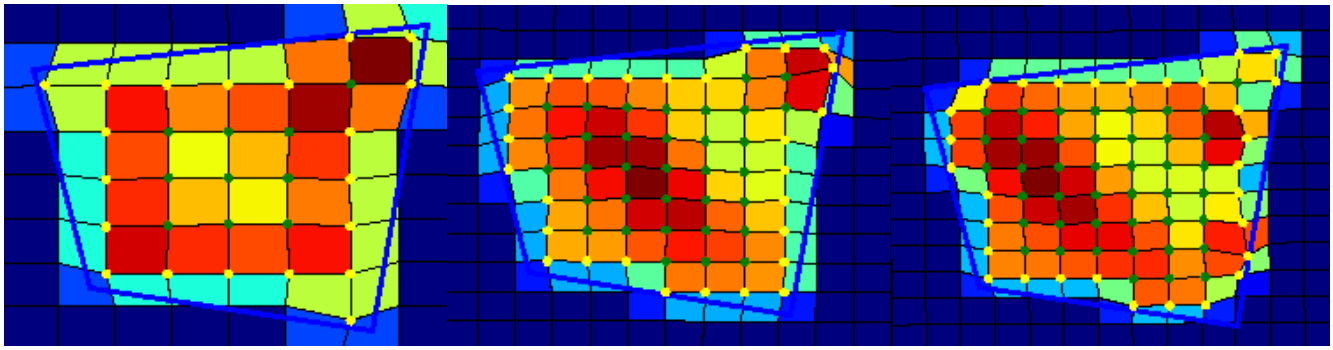


ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:
Построить решение методом сгущающихся сеток (3 раза), проанализировать сходимость и погрешность.

H — шаг сетки
E — средняя величина ошибки

Используем разностную схему:

$$-\frac{\upsilon_{k-1,m}-2\upsilon_{km}+\upsilon_{k+1,m}}{h^2}-\frac{\upsilon_{k,m-1}-2\upsilon_{km}+\upsilon_{k,m+1}}{h^2}=f_{km}.$$



H = 1.25
E ~ 0.2

H = 0.8
E ~ 0.14

H = 0.6
E ~ 0.12

Как видно из значений E, при уменьшении шага (при увеличении разрешения сетки разностной схемы) уменьшается среднее значение ошибки решения. Из этого можно заключить, что при H стремящемся к нулю, решение сойдется.

Вывод: в процессе выполнения работы было численно решено стационарноу уравнение Пуассона, используя метод сеток, разработан и реализован алгоритм решения на ЭВМ.

Код программы:

```
import numpy as np
from sympy import *
from sympy.solvers import solve
import math
import matplotlib
import matplotlib.path
import matplotlib.nxutils
matplotlib.use('GTK')
import matplotlib.pyplot as plt

import mpl_toolkits.mplot3d


CORNERX = [8, 7, 2, 1]
CORNERY = [8, 1, 2, 7]
CORNERS = np.array([list(x) for x in zip(CORNERX, CORNERY)])

USE3D = True#False

Qi = [0, 0, 1, 1]
Ai = [1, 1, 0.5, 0]
Bi = [-1, -1, 0.8, 0.2]

E = 2.71

FX = lambda x, y: math.sin(x + y)
#FX = lambda x, y: x*x+y*y
BORDERFX = lambda x, y: -1#math.sin(x * 2 + y * 2)
OUTER = -4
LIMIT = 2

plt.ion()
fig = plt.figure()

if USE3D:
    ax = fig.add_subplot(111, projection='3d')
else:
    ax = fig.add_subplot(111)
ax.grid()


size = 10
res = 12
res = 9
step = size * 1.0 / res
```

```
matrix = np.zeros((res, res))
pointtypes = np.zeros((res, res))
pointtypes2 = np.zeros((res, res))
symbols = [[0] * res for _ in range(res)]
symbollist = []
symbolx = {}
symboly = {}

xs = np.arange(0, size, step)
ys = np.arange(0, size, step)

poly = matplotlib.path.Path(CORNERS)
for x in range(0, res):
    for y in range(0, res):
        symbols[x][y] = Symbol('U_%i_%i' % (x, y))
        symbollist.append(symbols[x][y])
        symbolx[symbols[x][y]] = x
        symboly[symbols[x][y]] = y
        pointtypes[x][y] = 0
        if poly.contains_point((xs[x], ys[y])):
            pointtypes[x][y] = 1

for x in range(1, res-1):
    for y in range(1, res-1):
        pointtypes2[x, y] = pointtypes[x, y]
        if pointtypes[x, y] == 1:
             if pointtypes[x - 1, y] + pointtypes[x + 1, y] ==
1:  # 1,0 or 0,1
                pointtypes2[x, y] = 2
             if pointtypes[x, y - 1] + pointtypes[x, y + 1] ==
1:  # 1,0 or 0,1
                pointtypes2[x, y] = 2

pointtypes = pointtypes2

dscheme = []
dschemesymbols = []
for s in symbollist:
    x = symbolx[s]
    y = symboly[s]
    if pointtypes[x][y] == 2:
        dschemesymbols.append(s)
        dscheme.append(s - BORDERFX(xs[x], ys[y]))  # BORDER
    if pointtypes[x][y] == 1:
        dschemesymbols.append(s)

        dscheme.append(
                -(E ** (- (xs[x] ** 2))) * 2 * xs[x] * (-s +
symbols[x+1][y]) / step
```

```
            + E ** (- (xs[x] ** 2)) * (symbols[x-1][y] - 2 * s
+ symbols[x+1][y]) / (step ** 2)
              + (symbols[x][y-1] - 2 * s + symbols[x][y+1]) /
(step ** 2)
            - FX(x, y)
        )

        """
        dscheme.append(
              - (symbols[x-1][y] - 2 * s + symbols[x+1][y]) /
(step ** 2)
              - (symbols[x][y-1] - 2 * s + symbols[x][y+1]) /
(step ** 2)
            - FX(x, y)
        )
        """

print 'Solving'
solution = solve(dscheme)

print 'Mapping'

for x in range(0, res):
    for y in range(0, res):
        matrix[x][y] = OUTER

for s in dschemesymbols:
    x = symbolx[s]
    y = symboly[s]
    try:
        v = float(solution[s])
        if v > LIMIT: v = LIMIT
        if v < -LIMIT: v = -LIMIT
        matrix[x][y] = v
    except:
        print 'BAD SOLUTION: ', s, '=', solution[s]


coords3d = np.meshgrid(xs, ys)
values = np.zeros((res, res))
for x in range(0, res):
    for y in range(0, res):
        values[x][y] = matrix[y][x]

if USE3D:
    ax.plot_surface(
        coords3d[0],
        coords3d[1],
        values,
        rstride=1,
```

```
        cstride=1,
        cmap=matplotlib.cm.jet,
        shade=True,
    )
else:
    ax.pcolormesh(xs, ys, matrix)

# Draw bounds
ax.plot(
    CORNERX + [CORNERX[0]],
    CORNERY + [CORNERY[0]],
    lw=3
)

# Draw points
for x in range(0, res):
    for y in range(0, res):
        if pointtypes[x][y] > 0:
            c = 'green' if pointtypes[x][y] == 1 else 'yellow'
            if USE3D:
                ax.plot([xs[x]], [ys[y]], [matrix[x][y]], 'g.',
markersize=8.0, color=c)
            else:
                ax.plot([xs[x]], [ys[y]], 'g.', markersize=8.0,
color=c)


fig.show()
while True:
    fig.waitforbuttonpress()
```