Учреждение образования
Белорусский Государственный Университет
Информатики и Радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

Отчет по лабораторной работе №2:

«Моделирование параболических уравнений в частных произовдных, используя методы расщепления»

Вариант №10

Выполнил:
ст. гр. 052002
Паньков Е.В.

Проверил:
Анисимов В.Я.

Минск 2012

Задание

Используя методы переменных направлений и дробных шагов, решить двумерную начально-краевую задачу для дифференциального уравнения параболического типа для прямоугольной пластины. Вычислить погрешность в различные моменты времени.

$$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial y^2} + \sin x \sin y(\mu\cos\mu t + (a+b)\sin\mu t),$$

$$u(0, y, t) = 0,$$

$$u_x(\pi, y, t) = -\sin y \sin(\mu t),$$

$$u(x, 0, t) = 0,$$

$$u_y(x, \pi, t) = -\sin x \sin(\mu t),$$

$$u(x, y, 0) = 0.$$

Аналитическое решение: $U(x, y, t) = \sin x \sin y \sin(\mu t)$.

Метод переменных направлений

Используем разностную схему:

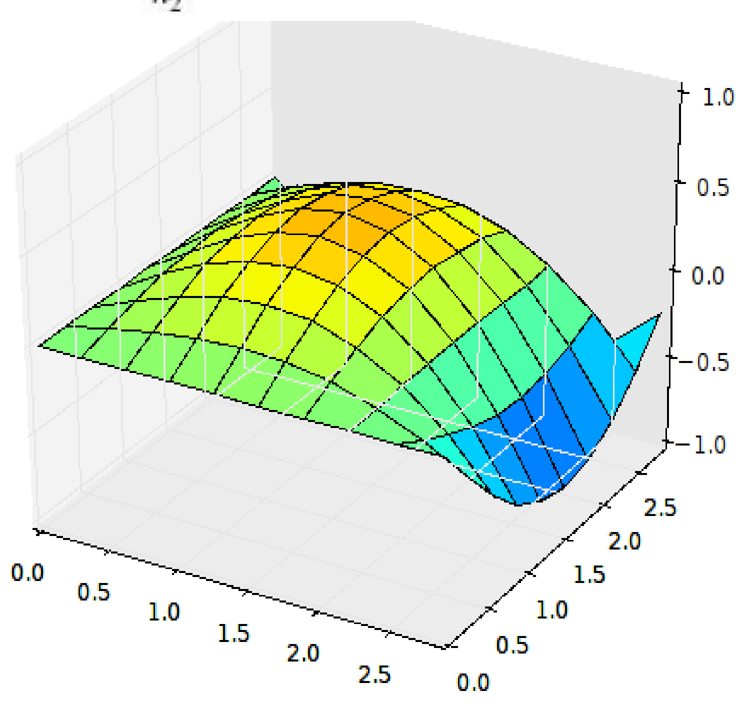
$$\frac{u_{ij}^{k+1/2} - u_{ij}^{k}}{\tau/2} = \frac{a}{h_1^2}\left(u_{i+1j}^{k+1/2} - 2u_{ij}^{k+1/2} + u_{i-1j}^{k+1/2}\right) + \frac{a}{h_2^2}\left(u_{ij+1}^{k} - 2u_{ij}^{k} + u_{ij-1}^{k}\right) + f_{ij}^{k+1/2},$$

(5.78)

$$\frac{u_{ij}^{k+1} - u_{ij}^{k+1/2}}{\tau/2} = \frac{a}{h_1^2}\left(u_{i+1j}^{k+1/2} - 2u_{ij}^{k+1/2} + u_{i-1j}^{k+1/2}\right) + \frac{a}{h_2^2}\left(u_{ij+1}^{k+1} - 2u_{ij}^{k+1} + u_{ij-1}^{k+1}\right) + f_{ij}^{k+1/2}.$$

H = 0.314, dT = 0.2

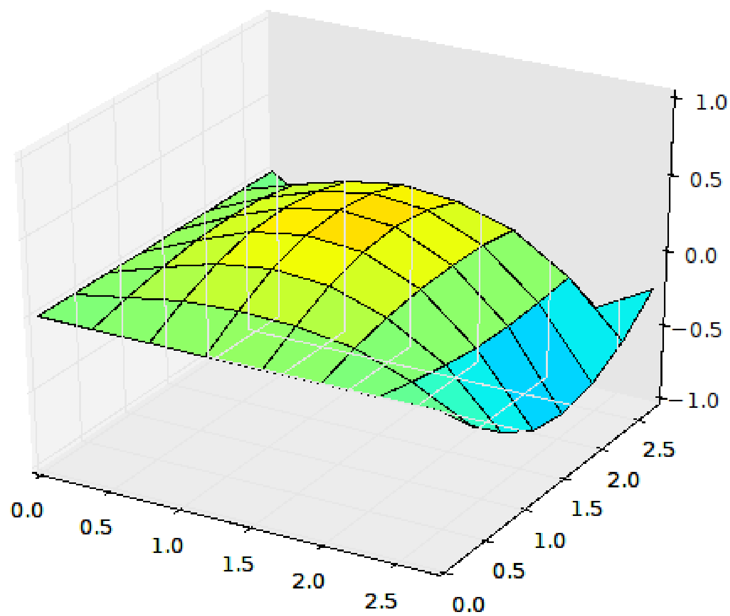
E	t
0.003	0
0.007	0.2
0.012	0.4
0.016	0.6
0.020	0.8
0.023	1
0.025	1.2



0.027	1.4
0.027	1.6
0.026	1.8
0.025	2.0
0.024	2.2
0.022	2.4

H = 0.4, dT = 0.2

E	t
0.004	0
0.008	0.2
0.013	0.4
0.017	0.6
0.023	0.8
0.028	1
0.031	1.2
0.033	1.4
0.032	1.6
0.031	1.8
0.030	2.0
0.028	2.2
0.025	2.4



H = 0.314, dT = 0.1   H = 0.4, dT = 0.1

E	t		E	t
0.002	0		0.003	0
0.004	0.2		0.006	0.2
0.007	0.4		0.010	0.4
0.010	0.6		0.012	0.6
0.019	0.8		0.020	0.8
0.020	1		0.022	1
0.025	1.2		0.028	1.2
0.028	1.4		0.030	1.4
0.028	1.6		0.029	1.6
0.027	1.8		0.027	1.8
0.026	2.0		0.026	2.0
0.022	2.2		0.024	2.2
0.020	2.4		0.023	2.4

Вывод: при использовании метода переменных направлений точность увеличивается с уменьшением шага сетки и временного шага (dT).

Метод дробных шагов

Используем разностную схему:

$$\frac{u_{ij}^{k+1/2}-u_{ij}^{k}}{\tau}=\frac{a}{h_1^2}\left(u_{i+1j}^{k+1/2}-2u_{ij}^{k+1/2}+u_{i-1j}^{k+1/2}\right)+\frac{f_{ij}^{k}}{2},$$

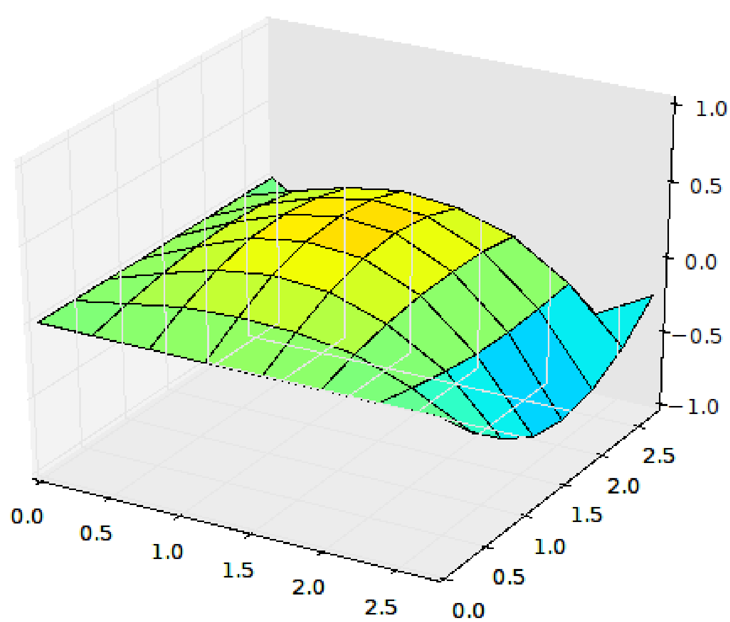
$$\frac{u_{ij}^{k+1}-u_{ij}^{k+1/2}}{\tau}=\frac{a}{h_2^2}\left(u_{i\,j+1}^{k+1}-2u_{ij}^{k+1}+u_{i\,j-1}^{k+1}\right)+\frac{f_{ij}^{k+1}}{2}.$$

H = 0.314, dT = 0.2

E	t
0.003	0
0.007	0.2
0.012	0.4
0.016	0.6
0.020	0.8
0.023	1
0.025	1.2
0.027	1.4
0.027	1.6
0.026	1.8
0.025	2.0
0.024	2.2
0.022	2.4

H = 0.4, dT = 0.2

E	t
0.004	0
0.008	0.2
0.013	0.4
0.017	0.6
0.023	0.8
0.028	1
0.031	1.2
0.033	1.4



0.032	1.6
0.031	1.8
0.030	2.0
0.028	2.2

H = 0.314, dT = 0.1   H = 0.4, dT = 0.1

E	t		E	t
0.002	0		0.003	0
0.004	0.2		0.006	0.2
0.007	0.4		0.010	0.4
0.010	0.6		0.012	0.6
0.019	0.8		0.020	0.8
0.020	1		0.022	1
0.025	1.2		0.028	1.2
0.028	1.4		0.030	1.4
0.028	1.6		0.029	1.6
0.027	1.8		0.027	1.8
0.026	2.0		0.026	2.0
0.022	2.2		0.024	2.2
0.020	2.4		0.023	2.4

Вывод: при использовании метода дробных шагов точность увеличивается с уменьшением шага сетки и временного шага (dT).

Вывод: в ходе работы были изучены и реализованы на ЭВМ методы переменных направлений и дробных шагов, проверена зависимость погрешности от различных параметров. Метод переменных направлений проявил более высокую точность при численном решении.

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

import mpl_toolkits.mplot3d


USE3D = True  # False

A = 1
B = 1
MU = 1

FX = lambda x, y, t: sin(x) * sin(y) * (MU * cos(MU * t) + (A + B) *
sin(MU * t))
BORDERX0 = lambda x, y, t: 0
BORDERX1 = lambda x, y, t: -sin(y) * sin(MU * t)
BORDERY0 = lambda x, y, t: 0
BORDERY1 = lambda x, y, t: -sin(x) * sin(MU * t)
INITIAL = lambda x, y, t: 0
IDEAL = lambda x, y, t: sin(x) * sin(y) * sin(MU * t)

VMIN = -1
VMAX = 1
DT = 0.2


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

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

```
size = 3.14
res = 10
step = size * 1.0 / res
xs = np.arange(0, size, step)
ys = np.arange(0, size, step)


def check_err(matrix, t):
    e = 0
    for x in range(1, res-1):
        for y in range(1, res-1):
            e += abs(IDEAL(xs[x], ys[y], t) - matrix[x][y])

    e /= res * res
    print 'Err = %.5f' % (e / 10)

from common_15 import *


def do_step(old, t, dt, fx, type):
    symbols = [[0] * res for _ in range(res)]
    for x in range(0, res):
        for y in range(0, res):
            s = Symbol('U_%i_%i' % (x, y))
            #s.x = xs[x]
            #s.y = xs[y]
            symbols[x][y] = s

    dscheme = []

    for i in range(res):
        dscheme.append(
            symbols[0][i] - BORDERX0(0, ys[i], t)
        )
        dscheme.append(
            symbols[res-1][i] - BORDERX1(size, ys[i], t)
        )
        dscheme.append(
            symbols[i][0] - BORDERY0(xs[i], 0, t)
        )
        dscheme.append(
            symbols[i][res-1] - BORDERY1(xs[i], size, t)
        )

    for x in range(1, res - 1):
        for y in range(1, res - 1):
            if type == 1:
                dscheme.append(
```

```
                    + A / (step ** 2) * (symbols[x+1][y] - 2 *
(symbols[x][y]) + symbols[x-1][y])
                    + fx(xs[x], ys[y]) / 2
                    - (symbols[x][y] - old[x][y]) / dt
                )
            else:
                dscheme.append(
                    + B / (step ** 2) * (symbols[x][y+1] - 2 *
(symbols[x][y]) + symbols[x][y-1])
                    + fx(xs[x], ys[y]) / 2
                    - (symbols[x][y] - old[x][y]) / dt
                )

    solution = solve(dscheme)

    new = np.zeros((res, res))
    for x in range(0, res):
        for y in range(0, res):
            new[x][y] = solution[symbols[x][y]]

    return new


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


t = 0


def big_step(matrix, dt):
    matrix = do_step(matrix, t, 0.1, lambda x, y: FX(x, y, t), 1)
    matrix = do_step(matrix, t + dt / 2, 0.1, lambda x, y: FX(x, y, t
+ dt), 2)
    return matrix

fig.show()

dt = DT
ctr = 0

coords3d = np.meshgrid(xs, ys)
while True:
    print 'Step %i' % ctr
    ctr += 1
```

```
matrix = big_step(matrix, dt)
t += dt
check_err(matrix, t)

ax.clear()

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,
        vmin=VMIN,
        vmax=VMAX,
        shade=True,
    )

    ax.plot([0], [0], [VMIN])
    ax.plot([0], [0], [VMAX])
else:
    ax.pcolormesh(xs, ys, matrix)

fig.canvas.draw()

while fig.waitforbuttonpress(timeout=0.01) is not None:
    pass
```