Министерство науки и высшего образования Российской Федерации
Новосибирский государственный технический университет
Кафедра прикладной математики

# Уравнения математической физики

## Практическая работа №1

Факультет: прикладной математики и информатики
Группа: ПМ-63
Студент: Кожекин М.В.
Преподаватели: Патрушев И. И.
Персова М.Г.

Новосибирск
2019

## 1. Цель работы

Разработать программу решения эллиптической краевой задачи методом конечных разностей. Протестировать программу и численно оценить порядок аппроксимации.

## 2. Задание

1. Построить прямоугольную сетку в области в соответствии с заданием. Допускается использовать фиктивные узлы для сохранения регулярной структуры.
2. Выполнить конечноразностную аппроксимацию исходного уравнения в соответствии с заданием. Получить формулы для вычисления компонент матрицы $\mathbf{A}$ и вектора правой части $\mathbf{b}$.
3. Реализовать программу решения двумерной эллиптической задачи методом конечных разностей с учетом следующих требований:
   - язык программирования C++ или Фортран;
   - предусмотреть возможность задания неравномерных сеток по пространству, учет краевых условий в соответствии с заданием;
   - матрицу хранить в диагональном формате, для решения СЛАУ использовать метод блочной релаксации [2, с. 886; 3].
4. Протестировать разработанные программы на полиномах соответствующей степени.
5. Исследовать порядок аппроксимации реализованного метода для различных задач с неполиномиальными решениями. Сравнить полученный порядок аппроксимации с теоретическим.

Вариант 6: Область имеет L-образную форму. Предусмотреть учет первых и третьих краевых условий.

## 3. Анализ

Уравнение $-div(\lambda\ grad\ u) + \gamma u = f$ для функции $u = u(x, y)$.

Первые краевые условия: $f(x, y) = value$

Третьи краевые условия: $f'^{(x,y)} + C_1 f(x, y) = C_2$

Пятиточечный оператор Лапласа на равномерной сетке имеет вид:

$$\Delta_h u_i = \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h_x^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h_y^2}$$

Пятиточечный оператор Лапласа на неравномерной сетке имеет вид:

$$\Delta_h u_i = \frac{2u_{i-1}}{h_{i-1}(h_i + h_{i-1})} - \frac{2u_i}{h_{i-1}h_i} + \frac{2u_{i+1}}{h_i(h_i + h_{i-1})}$$

Неравномерная сетка рассчитывается по формуле:

$$b - a = h_1 + h_1 k + \cdots + h_1 k^{n-1}$$

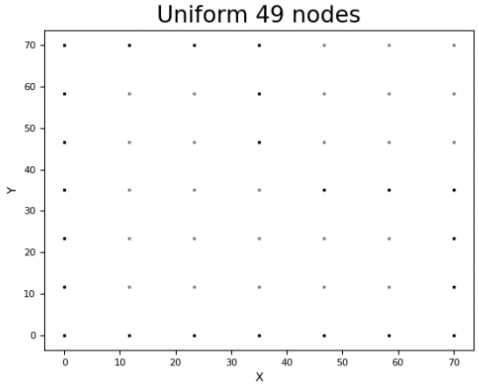
## 4. Исследования

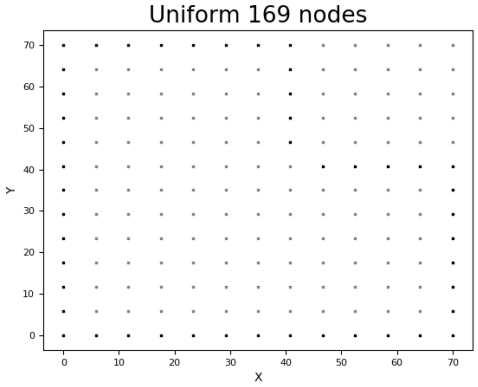
Сетка, используемая в исследовании имеет фиксированную топологию – 7x7 узлов, часть из которых фиктивные. Область [0, 70] x [0, 70]. Для неравномерной сетки коэффициент k = 1.05 Каждая сетка имеет 9/49 фиктивных узлов.

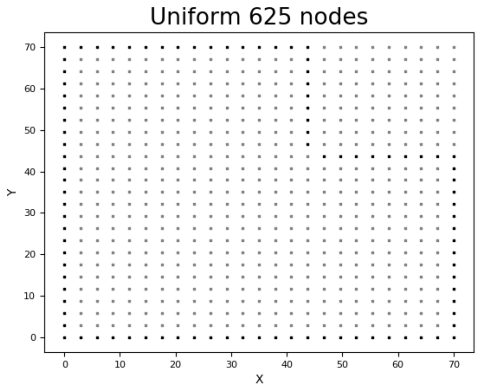
Во всех исследованиях параметры СЛАУ были одинаковые E = 1e-12, maxiter = 1000000.

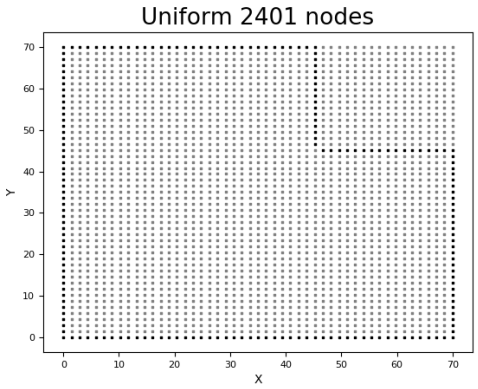
Для определения точности использовалась норма $L_2$, учитывающая основные 49 узлов.

**Равномерная сетка**

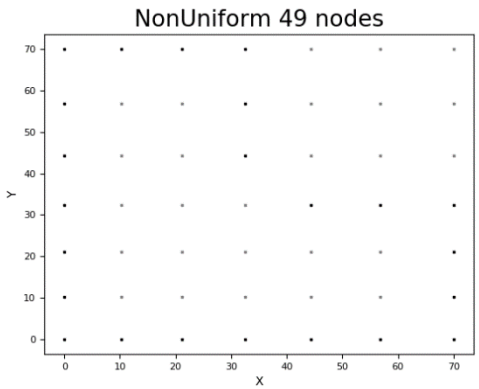


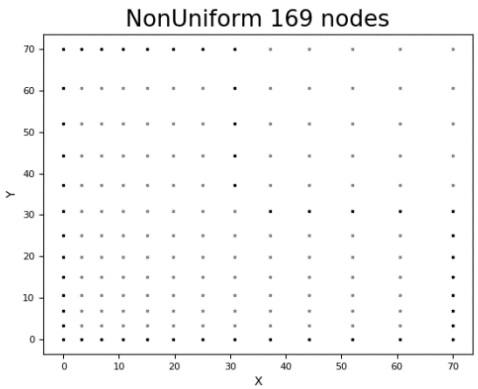



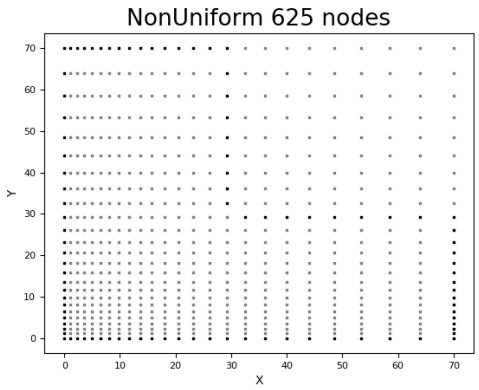



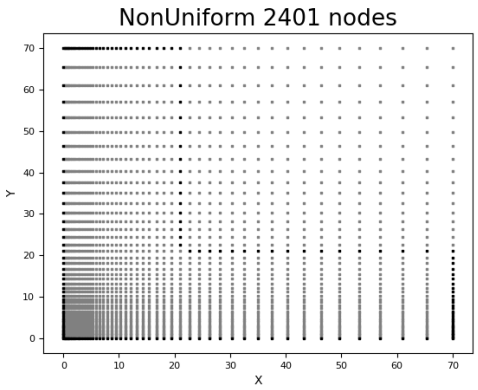




**Неравномерная сетка**






## Точность на разных функциях:

| Функция | 1 краевые | 3 краевые на одной границе |
|---|---|---|
| x + y | 4.890469e-08 | 2.846524e+00 |
| 10x + y | 2.689759e-07 | 2.846526e+00 |
| x^2 + y^2 | 2.612537e-06 | 3.795368e+00 |
| x^3 + y^3 | 1.627323e-04 | 7.464268e+01 |
| x^4 + y^4 | 3.134554e+05 | 3.117346e+05 |
| sin(x) + cos(x) | 5.520723e+02 | 5.474130e+02 |
| exp(x^2 + y^2) | 3.241467e+40 | 3.238622e+40 |

## Точность в зависимости от дробления сетки:

Функция x + y

| Число узлов | Равномерная 1 | Равномерная 3 | Неравномерная 1 | Неравномерная 3 |
|---|---|---|---|---|
| 49 | 4.890469e-08 | 2.846524e+00 | 3.903889e-08 | 2.889029e+00 |
| 169 | 4.174234e-08 | 2.898914e+00 | 2.122752e-08 | 3.113735e+00 |
| 625 | 3.317470e-08 | 2.930316e+00 | 1.099163e-08 | 3.302783e+00 |
| 2401 | 2.465322e-08 | 2.969808e+00 | 2.254498e-09 | inf |

Функция 10x + y

| Число узлов | Равномерная 1 | Равномерная 3 | Неравномерная 1 | Неравномерная 3 |
|---|---|---|---|---|
| 49 | 2.689759e-07 | 2.846526e+00 | 2.147138e-07 | 2.889031e+00 |
| 169 | 2.488475e-07 | 2.898916e+00 | 1.283241e-07 | 3.113736e+00 |
| 625 | 2.000782e-07 | 2.930318e+00 | 6.868117e-08 | 3.302784e+00 |
| 2401 | 1.484781e-07 | 2.969810e+00 | 1.464542e-08 | inf |

Функция x^2 + y^2

| Число узлов | Равномерная 1 | Равномерная 3 | Неравномерная 1 | Неравномерная 3 |
|---|---|---|---|---|
| 49 | 2.612537e-06 | 3.795368e+00 | 2.000282e-06 | 3.057305e+00 |
| 169 | 2.377901e-06 | 4.831550e-01 | 1.163375e-06 | 1.105652e+00 |
| 625 | 1.883863e-06 | 1.220962e+00 | 6.156823e-07 | 2.570099e+00 |
| 2401 | 1.405485e-06 | 2.103614e+00 | 1.311001e-07 | inf |

Функция x^3 + y^3

| Число узлов | Равномерная 1 | Равномерная 3 | Неравномерная 1 | Неравномерная 3 |
|---|---|---|---|---|
| 49 | 1.627323e-04 | 7.464268e+01 | 1.128987e+03 | 1.101616e+03 |
| 169 | 1.437524e-04 | 1.683031e+01 | 9.768367e+02 | 9.684492e+02 |
| 625 | 1.154721e-04 | 2.055716e+00 | 3.244674e+02 | 3.227378e+02 |
| 2401 | 8.672038e-05 | 1.706204e+00 | 6.875341e+01 | inf |

Функция x^4 + y^4

| Число узлов | Равномерная 1 | Равномерная 3 | Неравномерная 1 | Неравномерная 3 |
|---|---|---|---|---|
| 49 | 3.134554e+05 | 3.117346e+05 | 3.849107e+05 | 3.828598e+05 |
| 169 | 1.011097e+05 | 1.005844e+05 | 1.710433e+05 | 1.699298e+05 |
| 625 | 2.816841e+04 | 2.802627e+04 | 5.268197e+04 | 5.226888e+04 |
| 2401 | 7.383205e+03 | 7.347098e+03 | 9.902214e+03 | inf |

Функция sin(x) + cos(x)

| Число узлов | Равномерная 1 | Равномерная 3 | Неравномерная 1 | Неравномерная 3 |
|---|---|---|---|---|
| 49 | 5.520723e+02 | 5.474130e+02 | 2.329131e+02 | 2.332171e+02 |
| 169 | 6.557697e+02 | 6.501179e+02 | 2.209198e+02 | 2.206330e+02 |
| 625 | 7.026720e+00 | 6.295189e+00 | 1.767073e+01 | 1.702418e+01 |
| 2401 | 1.289721e+00 | 2.786325e+00 | 1.999323e+00 | inf |

Функция e^(x+y)

| Число узлов | Равномерная 1 | Равномерная 3 | Неравномерная 1 | Неравномерная 3 |
|---|---|---|---|---|
| 49 | 3.241467e+40 | 3.238622e+40 | 1.760230e+39 | 1.758197e+39 |
| 169 | 1.241352e+45 | 1.240916e+45 | 1.521083e+40 | 1.518537e+40 |
| 625 | 5.859998e+46 | 5.857720e+46 | 8.236691e+39 | 8.207838e+39 |
| 2401 | 1.280197e+47 | 1.279655e+47 | 7.335728e+35 | inf |

## 5. Выводы

На равномерной сетке пятиточечный оператор Лапласа имеет второй порядок погрешности.

Как видно из исследований, метод конечных разностей имеет высокую точность решения на полиномах по вторую степень. На полиномах более высокого порядка решение значительно хуже или же не работает вовсе.

Также стоит отметить, что на полиномиальных функциях третьи краевые условия снижают эффективность метода.

## 6. Текст программы

Для удобства программа была разбита на следующие модули:

head.h – заголовочный файл, в котором определяется точность вычислений

grid.h и grid.cpp – сетка

slae.h и slae.cpp – СЛАУ

fdm.h и fdm.cpp – метод конечных разностей

main.cpp – файл с исследованиями

**head.h**

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <fstream>
#include <iostream>
#include <vector>
#include <string>
#include <string>
#include <iomanip>
#include <functional>
#include <math.h>

using namespace std;


typedef std::function<double(double, double)> function2D;
```

**grid.h**

```
#pragma once
#include "head.h"


struct NODE {

	bool isFirstNode = false;
	int i, j;
	double x, y;
	int type = -9000;		// -9000	начение при инициализации
							// -1		фиктивный узел
							// 0		внутренний узел
							// n		номер границы
	int border;				// номер границы
	void setNodesData(double _x, double _y, int _i, int _j, int _type, double _coef) {
		x = _x;
		y = _y;
		i = _i;
		j = _j;
		type = _type;
		if (i % int(pow(2, _coef)) == 0 && j % int(pow(2.0, _coef)) == 0)
			isFirstNode = true;
	}
};



class GRID
{
public:
	void inputGrid();
	void buildGrid();
	void showGrid();
	void saveGridAndBorder(const string &filepathGrid, const string &filepathGridBorder);

protected:
	bool isGridUniform;
	int condType, coef;
	int width, heigth, widthLeft, widthRight, heigthLower, heigthUpper, elemCount;
	double xLeft, xRight, yLower, yUpper;
	double hx, hy, nx, ny, kx, ky, hxPrev, hyPrev;
```

```
	double dx, dy;		// hx_n = hx_1 + kx^(n-1) геометрическая прогрессия
	vector <NODE> nodes;
};
```

**grid.cpp**

```
#include "grid.h"
#include "fdm.h"



void GRID::inputGrid()
{
	string filepath;
	if (isGridUniform)
		filepath = "input/uniform_grid.txt";
	else
		filepath = "input/nonuniform_grid.txt";

	std::ifstream fin(filepath);

	fin >> xLeft >> xRight >>
		yLower >> yUpper;

	fin >> width >> heigth >>
		widthLeft >> widthRight >>
		heigthLower >> heigthUpper;

	if (!isGridUniform) {
		fin >> kx >> ky;
		nx = width - 1;
		ny = heigth - 1;
	}

	fin.close();
}




void GRID::buildGrid()
{
	//      c    d          Where:
	//    -----====         a - height
	//  | 66665xxxx !        b - width
	//  | 10005xxxx ! e      c - widthLeft
	// a| 10005xxxx !        d - widthRight
	//  | 100004443 |        e - heightUpper
	//  | 100000003 | f      f - heightLower
	//  | 100000003 |
	//  | 122222222 |
	//    ---------
	//        b
	//
	// 66665xxxx
	// 10005xxxx
	// 10005xxxx
	// 100004443
	// 100000003
	// 100000003
	// 122222222
```

```
	//

	if (isGridUniform) {
		hx = ((xRight - xLeft) / double(width - 1)) / pow(2, coef);
		hy = ((yUpper - yLower) / double(heigth - 1)) / pow(2, coef);
		cout << hx << "\t" << hy << endl;
		if (coef != 0) {
			width = (width - 1) * pow(2, coef) + 1;
			heigth = (heigth - 1) * pow(2, coef) + 1;
			widthLeft = (widthLeft)* pow(2, coef);
			widthRight = (widthRight - 1) * pow(2, coef) + 1;
			heigthLower = (heigthLower)* pow(2, coef);
			heigthUpper = (heigthUpper - 1) * pow(2, coef) + 1;
		}
	}
	else {
		if (coef != 0) {
			width = (width - 1) * pow(2, coef) + 1;
			heigth = (heigth - 1) * pow(2, coef) + 1;
			widthLeft = (widthLeft)* pow(2, coef);
			widthRight = (widthRight - 1) * pow(2, coef) + 1;
			heigthLower = (heigthLower)* pow(2, coef);
			heigthUpper = (heigthUpper - 1) * pow(2, coef) + 1;
			nx *= pow(2, coef);
			ny *= pow(2, coef);
			kx *= pow(kx, 1.0 / coef);
			ky *= pow(ky, 1.0 / coef);
		}

		hx = (xRight - xLeft) * (1 - kx) / (1 - pow(kx, nx));
		hy = (yUpper - yLower) * (1 - ky) / (1 - pow(ky, ny));
	}




	elemCount = width * heigth;
	cout << "Grid is uniform" << endl
		<< "Coef: " << coef << endl
		<< "Width: " << width << endl
		<< "Height: " << heigth << endl
		<< "Count of elements: " << elemCount << endl
		<< "hx: " << hx << endl
		<< "hy: " << hy << endl;
	nodes.resize(elemCount);


	if (isGridUniform) {


		size_t i, j, elem;
		double x, y, xPrev, yPrev;
		// Обходим все внутренние элементы нижней части "L" по пяти точкам
		for (j = 1; j < heigthLower - 1; j++)
		{
			i = 1;
			for (elem = j * width + 1; elem < (j + 1) * width - 1; elem++, i++)
			{
				x = xLeft + hx * i;
				y = yLower + hy * j;
```

```
                        nodes[elem].setNodesData(x, y, i, j, 0, coef);
                }
        }

        // Обходим все внутренние элементы средней части части "L" по пяти точкам
        i = 1;
        j = heigthLower - 1;
        y = yLower + hy * j;
        for (elem = j * width + 1; elem < j * width + widthLeft; elem++, i++)
        {
                x = xLeft + hx * i;
                nodes[elem].setNodesData(x, y, i, j, 0, coef);
        }


        // Обходим все внутренние элементы верхней части "L" по пяти точкам
        for (j = heigthLower; j < heigth - 1; j++)
        {
                i = 1;
                for (elem = j * width + 1; elem < j * width + widthLeft - 1; elem++,
i++)
                {
                        x = xLeft + hx * i;
                        y = yLower + hy * j;
                        nodes[elem].setNodesData(x, y, i, j, 0, coef);
                }
        }

        // 1
        i = 0;
        j = 0;
        x = xLeft + hx * i;
        for (elem = 0; elem < (heigth - 1) * width; elem += width, j++)
        {
                y = yLower + hy * j;
                nodes[elem].setNodesData(x, y, i, j, condType, coef);
                nodes[elem].border = 1;
        }

        // 2
        i = 1;
        j = 0;
        y = yLower + hy * j;
        for (elem = 1; elem < width; elem++, i++)
        {
                x = xLeft + hx * i;
                nodes[elem].setNodesData(x, y, i, j, condType, coef);
                nodes[elem].border = 2;
        }

        // 3
        i = width - 1;
        j = 1;
        x = xLeft + hx * i;
        for (elem = 2 * width - 1; elem < (width * heigthLower); elem += width, j++)
        {
                y = yLower + hy * j;
                nodes[elem].setNodesData(x, y, i, j, condType, coef);
                nodes[elem].border = 3;
        }
```

```
		// 4
		i = widthLeft;
		j = heigthLower - 1;
		y = yLower + hy * j;
		for (elem = width * heigthLower - widthRight; elem < width * heigthLower - 1;
elem++, i++)
		{
			x = xLeft + hx * i;
			nodes[elem].setNodesData(x, y, i, j, condType, coef);
			nodes[elem].border = 4;
		}

		// 5
		i = widthLeft - 1;
		j = heigthLower;
		x = xLeft + hx * i;
		for (elem = width * j + widthLeft - 1; elem < elemCount; elem += width, j++)
		{
			y = yLower + hy * j;
			nodes[elem].setNodesData(x, y, i, j, condType, coef);
			nodes[elem].border = 5;
		}

		// 6
		i = 0;
		j = heigth - 1;
		y = yLower + hy * j;
		for (elem = width * j; elem < width * j + widthLeft - 1; elem++, i++)
		{
			x = xLeft + hx * i;
			nodes[elem].setNodesData(x, y, i, j, condType, coef);
			nodes[elem].border = 6;
		}

		// fictitious nodes
		for (j = heigthLower; j < heigth; j++)
		{
			i = widthLeft;
			for (elem = width * j + widthLeft; elem < width * (j + 1); elem++, i++)
			{
				x = xLeft + hx * i;
				y = yLower + hy * j;
				nodes[elem].setNodesData(x, y, i, j, -1, coef);
			}
		}
	}

	else {

		double x, y;
		size_t i, j, elem;


		// Обходим все внутренние элементы нижней части "L" по пяти точкам
		dy = hy * ky;
		y = yLower + hy;
		for (j = 1; j < heigthLower - 1; j++, dy *= ky)
		{
			i = 1;
```

```
                        dx = hx * kx;
                        x = xLeft + hx;
                        for (elem = j * width + 1; elem < (j + 1) * width - 1; elem++, i++, dx
*= kx)
                        {
                                nodes[elem].setNodesData(x, y, i, j, 0, coef);
                                x += dx;
                        }
                        y += dy;
                }



                // Обходим все внутренние элементы средней части части "L" по пяти точкам
                i = 1;
                dx = hx * kx;
                x = xLeft + hx;

                j = heigthLower - 1;
                dy = hy * pow(ky, j);
                y = yLower + hy * (1 - pow(ky, j)) / (1 - ky);

                for (elem = j * width + 1; elem < j * width + widthLeft; elem++, i++, dx *= kx)
                {
                        nodes[elem].setNodesData(x, y, i, j, 0, coef);
                        x += dx;
                }


                // Обходим все внутренние элементы верхней части "L" по пяти точкам
                dy = hy * pow(ky, heigthLower);
                y = yLower + hy * (1 - pow(ky, heigthLower)) / (1 - ky);
                for (j = heigthLower; j < heigth - 1; j++, dy *= ky)
                {
                        i = 1;
                        dx = hx * kx;
                        x = xLeft + hx;

                        for (elem = j * width + 1; elem < j * width + widthLeft - 1; elem++,
i++, dx *= kx)
                        {
                                nodes[elem].setNodesData(x, y, i, j, 0, coef);
                                x += dx;
                        }
                        y += dy;
                }



                // 1
                i = 0;
                dx = hx;
                x = xLeft;

                j = 0;
                dy = hy;
                y = yLower;

                for (elem = 0; elem < (heigth - 1) * width; elem += width, j++, dy *= ky)
                {
```

```
                        nodes[elem].setNodesData(x, y, i, j, condType, coef);
                        nodes[elem].border = 1;
                        y += dy;
                }



                // 2
                i = 1;
                dx = hx * kx;
                x = xLeft + hx;

                j = 0;
                dy = hy;
                y = yLower;

                for (elem = 1; elem < width; elem++, i++, dx *= kx)
                {
                        nodes[elem].setNodesData(x, y, i, j, condType, coef);
                        nodes[elem].border = 2;
                        x += dx;
                }



                // 3
                i = width - 1;
                dx = hx * pow(kx, i - 1);
                x = xLeft + hx * (1 - pow(kx, i)) / (1 - kx);

                j = 1;
                dy = hy * ky;
                y = yLower + hy;

                for (elem = 2 * width - 1; elem < (width * heigthLower); elem += width, j++, dy
*= ky)
                {
                        nodes[elem].setNodesData(x, y, i, j, condType, coef);
                        nodes[elem].border = 3;
                        y += dy;
                }



                // 4
                i = widthLeft;
                dx = hx * pow(kx, i);
                x = xLeft + hx * (1 - pow(kx, i)) / (1 - kx);

                j = heigthLower - 1;
                dy = hy * pow(ky, j);
                y = yLower + hy * (1 - pow(ky, j)) / (1 - ky);

                for (elem = width * heigthLower - widthRight; elem < width * heigthLower - 1;
elem++, i++, dx *= kx)
                {
                        nodes[elem].setNodesData(x, y, i, j, condType, coef);
                        nodes[elem].border = 4;
                        x += dx;
                }
```

```
		// 5
		i = widthLeft - 1;
		x = xLeft + hx * (1 - pow(kx, i)) / (1 - kx);

		j = heigthLower;
		dy = hy * pow(ky, j);
		y = yLower + hy * (1 - pow(ky, j)) / (1 - ky);

		for (elem = width * j + widthLeft - 1; elem < elemCount; elem += width, j++, dy
*= ky)
		{
			nodes[elem].setNodesData(x, y, i, j, condType, coef);
			nodes[elem].border = 5;
			y += dy;
		}



		// 6
		i = 0;
		dx = hx;
		x = xLeft;

		j = heigth - 1;
		y = yLower + hy * (1 - pow(ky, j)) / (1 - ky);

		for (elem = width * j; elem < width * j + widthLeft - 1; elem++, i++, dx *= kx)
		{
			nodes[elem].setNodesData(x, y, i, j, condType, coef);
			nodes[elem].border = 6;
			x += dx;
		}



		// fictitious nodes
		j = heigthLower;
		dy = hy * pow(ky, j);
		y = yLower + hy * (1 - pow(ky, j)) / (1 - ky);

		for (j = heigthLower; j < heigth; j++, dy *= ky)
		{
			i = widthLeft;
			dx = hx * pow(kx, i);
			x = xLeft + hx * (1 - pow(kx, i)) / (1 - kx);
			for (elem = width * j + widthLeft; elem < width * (j + 1); elem++, i++,
dx *= kx)
			{
				nodes[elem].setNodesData(x, y, i, j, -1, coef);
				x += dx;
			}
			y += dy;
		}
	}
}
```

```
// Отображние сетки на экран
void GRID::showGrid() {

	cout << "X:" << endl;
	for (size_t j = 0; j < heigth; j++)
	{
		size_t elem = j * width;
		for (size_t i = 0; i < width; i++, elem++)
		{
			cout << nodes[elem].x << " ";
		}
		cout << endl;
	}

	cout << endl << endl << "Y:" << endl;

	for (size_t j = 0; j < heigth; j++)
	{
		size_t elem = j * width;
		for (size_t i = 0; i < width; i++, elem++)
		{
			cout << nodes[elem].y << " ";
		}
		cout << endl;
	}
}



// Сохранение внутренних и внешних узлов в 2 файлах
void GRID::saveGridAndBorder(const string &filepathGrid, const string &filepathGridBorder) {

	ofstream grid(filepathGrid);
	ofstream border(filepathGridBorder);
	for (size_t i = 0; i < elemCount; i++)
	{
		if (nodes[i].type > 0)
			border << nodes[i].x << " " << nodes[i].y << endl;
		else
			grid << nodes[i].x << " " << nodes[i].y << endl;
	}

	border.close();
	grid.close();
}
```

**slae.h**

```
#pragma once
#include "head.h"

class SLAE {

public:
	void inputSLAEParameters();
	void generateInitualGuess() { x.clear(); x.resize(n, 0); }
	void writeXToFile(const string &filepath);
	void writebToFile(const string &filepath);

	void convMatrixToDense();
```

```
	void writeDenseMatrixToFile(const string &filepath);
	void writeSecondDenseMatrixToFile(const string &filepath);
	double multLine(vector<double>& line, int i, int mode);
	void mult();

	void Jacobi(double w);
	void GaussSeildel(double w);
	int calcIterative(int useJacopbNotGaussSeidel, double w);
	double findOptimalW(int mode);


protected:
	vector <vector <double>> A, A2;
	vector <double> di, au1, au2, al1, al2;
	vector <double> x, b;
	int n, m, maxiter;
	double E;


	double calcNormE(vector <double> &x);
	double calcRelativeDiscrepancy();
	double calcAii(int i);
};
```

**slae.cpp**

```
#pragma once
#include "slae.h"

////////////////////////////////////////////////////////////////////////////////////////
// matrix

// Рассчёт суммы элементов строки при генерации матрицы A(k)
double SLAE::calcAii(int i) {

	double sum = 0;

	if (i >= 1) { // Нижний треугольник
		sum += al1[i - 1];
		if (i >= m) {
			sum += al2[i - m];
		}
	}

	sum += di[i];

	if (i < n - 1) { // Верхний треугольник
		sum += au1[i];
		if (i < n - m) {
			sum += au2[i];
		}
	}

	return sum;
}


////////////////////////////////////////////////////////////////////////////////////////
// vect
```

```
void SLAE::inputSLAEParameters()
{
	ifstream fin("input/SLAE_parameters.txt");

	fin >> E >> maxiter;

	fin.close();
}

// Вывод вектора X в файл
void SLAE::writeXToFile(const string &filepath) {

	std::ofstream fout;
	fout.open(filepath);

	for (int i = 0; i < x.size(); ++i)
		fout << x[i] << endl;
	fout.close();
}

void SLAE::writebToFile(const string & filepath) {

	std::ofstream fout;
	fout.open(filepath);

	for (int i = 0; i < x.size(); ++i)
		fout << b[i] << endl;
	fout.close();
}

////////////////////////////////////////////////////////////////////////////////////////////
// slae



// Преобразование 5-ми диагональной матрицы в плотный формат
void SLAE::convMatrixToDense() {

	A.resize(n);
	for (int i = 0; i < n; ++i) {
		A[i].resize(n, 0);
		A[i][i] = di[i];
	}


	int j = 1;
	for (int i = 0; i < al1.size(); ++i, ++j) {

		A[i][j] = au1[i];
		A[j][i] = al1[i];
	}

	j = m;
	for (int i = 0; i < al2.size(); ++i, ++j) {

		A[i][j] = au2[i];
		A[j][i] = al2[i];
	}
}
```

```
// Вывод плотной матрицы в файл
void SLAE::writeDenseMatrixToFile(const string & filepath) {

	std::ofstream fout;
	fout.open(filepath);

	for (int i = 0; i < n;++i) {
		for (int j = 0; j < n; ++j)
			fout << A[i][j] << "\t";
		fout << endl;
	}

	fout.close();
}

void SLAE::writeSecondDenseMatrixToFile(const string & filepath)
{
	std::ofstream fout;
	fout.open(filepath);

	for (int i = 0; i < n; ++i) {
		for (int j = 0; j < n; ++j)
			fout << A2[i][j] << "\t";
		fout << endl;
	}

	fout.close();
}


// Умножение i-й строки матрицы на вектор
double SLAE::multLine(vector <double> &line, int i, int mode) {

	double sum = 0;
	if (mode == 1 || mode == 3) {		// Нижний треугольник

		if (i > 0) {

			sum += al1[i - 1] * line[i - 1];
			if (i > m) {
				sum += al2[i - m] * line[i - m];
			}
		}
	}


	if (mode == 2 || mode == 3) {		// Главная диагональ
										// и верхний треугольник
		sum += di[i] * line[i];
		if (i < n - 1) {

			sum += au1[i] * line[i + 1];

			if (i < n - m) {
				sum += au2[i] * line[i + m];
			}
		}
	}
```

```
	return sum;
}


// Умножение матрицы на вектор
void SLAE::mult() {

	int index;
	b.clear();
	b.resize(n, 0);
	// Нижний треугольник
	index = 1;
	for (int i = 0; i < al1.size(); ++i, ++index)
		b[index] += al1[i] * x[i];
	index = m;
	for (int i = 0; i < al2.size(); ++i, ++index)
		b[index] += al2[i] * x[i];


	// Главная диагональ
	for (int i = 0; i < di.size(); ++i)
		b[i] += di[i] * x[i];


	// Верхний треугольник
	index = 1;
	for (int i = 0; i < au1.size(); ++i, ++index)
		b[i] += au1[i] * x[index];
	index = m;
	for (int i = 0; i < au2.size(); ++i, ++index)
		b[i] += au2[i] * x[index];
}


// Метод Якоби. 0 < w < 1
// Используется общая память для x и x1
void SLAE::Jacobi(double w) {

	double sum;
	vector <double> x1;
	x1.resize(n);

	for (int i = 0; i < n; ++i) {
		sum = multLine(x, i, 3);
		//x[i] += w * (b[i] - sum) / di[i];
		x1[i] = x[i] + w * (b[i] - sum) / di[i];
	}
	x = x1;
}


// Метод Гаусса-Зейделя. 0 < w < 2
void SLAE::GaussSeildel(double w) {

	double sum;
	vector <double> x1 = x;

	for (int i = 0; i < n; ++i) {

		sum = multLine(x1, i, 1);
```

```
			sum += multLine(x, i, 2);

			x1[i] = x[i] + w * (b[i] - sum) / di[i];
		}
		x = x1;
}


// Решение СЛАУ итерационным методом
// 1  Метод Якоби
// 2  Метод Гаусса-Зейделя
int SLAE::calcIterative(int useJacopbNotGaussSeidel, double w) {
		int i = 0;
		while (i < maxiter && calcRelativeDiscrepancy() >= E) {

			if (useJacopbNotGaussSeidel)
				Jacobi(w);
			else
				GaussSeildel(w);

			++i;
		}

		return i;
}


// Поиск оптимального веса
double SLAE::findOptimalW(int mode) {

		double optimalW = 0.0, tmpW;
		int max_i, min_i = maxiter, tmp_i;
		if (mode == 1) max_i = 101;
		else max_i = 200;

		for (int i = 0; i < max_i; ++i) {

			generateInitualGuess();
			tmpW = double(i) / 100;
			tmp_i = calcIterative(mode, tmpW);
			if (tmp_i < min_i) {
				min_i = tmp_i;
				optimalW = tmpW;
			}


		}
		generateInitualGuess();
		min_i = calcIterative(mode, optimalW);

		return optimalW;
}




// Вычисление нормы в Евклидовом пространстве
double SLAE::calcNormE(vector <double> &x) {

		double normE = 0;
```

```
	for (int i = 0; i < n; i++)
		normE += x[i] * x[i];

	return sqrt(normE);
}


// Рассчёт относительной невязки
double SLAE::calcRelativeDiscrepancy() {

	vector <double> numerator, denominator = b;
	numerator.resize(n);

	mult(); // b = A*x

	for (int i = 0; i < n; ++i)
		numerator[i] = denominator[i] - b[i]; // b - A*x

	// || b - A*x || / || b ||
	double res = calcNormE(numerator) / calcNormE(denominator);
	b = denominator;
	return res;
}
```

**fdm.h**

```
#pragma once
#include "head.h"
#include "slae.h"
#include "grid.h"


// Класс МКР. L-образная область.
class FDM : public GRID, public SLAE
{
public:
	void init(function2D &_u, function2D &_f, bool isGridUniform, int _condType, int
_coef);
	void inputEquationParameters();
	void outputSLAE(const string &fileA);
	void transformGridToSLAE();
	double calcAbsResidual(const string &filepath);

	void checkAnswer();



private:
	vector <double> xExp;
	double lambda = 1, gamma = 1;	// коэффициенты диффуров
	double C1 = 5, C2 = 5;

	function2D u, f;


	double calcFirstDerivativeX(double x, double y) {
		const double h = 1e-9;
		return (-u(x + 2 * h, y) + 8 * u(x + h, y) - 8 * u(x - h, y) + u(x - 2 * h, y))
/ (12 * h);
	}
```

```
	double calcFirstDerivativeY(double x, double y) {
		const double h = 1e-9;
		return (-u(x, y + 2 * h) + 8 * u(x, y + h) - 8 * u(x, y - h) + u(x, y - 2 * h))
/ (12 * h);
	}
};
```

**fdm.cpp**

```
#include "head.h"
#include "fdm.h"




// Инициализируем модель, задавая функции u, f и тип сетки
void FDM::init(function2D & _u, function2D & _f, bool _isGridUniform, int _condType, int
_coef)
{
	u = _u;
	f = _f;
	isGridUniform = _isGridUniform;
	condType = _condType;
	coef = _coef;
}


// Ввод параметров точности и коэффициентов уравнения из файла
void FDM::inputEquationParameters() {

	std::ifstream fin("equation_parameters.txt");

	fin >> lambda >> gamma;

	fin.close();
}


// Вывод слау в файлы
void FDM::outputSLAE(const string & fileA)
{
	ofstream foutA(fileA);

	foutA << "A = [ ";
	for (size_t i = 0; i < elemCount; i++)
	{
		for (size_t j = 0; j < elemCount; j++)
			foutA << A[i][j] << "\t";

		foutA << ";" << endl;
	}

	foutA << endl << endl << endl << "b = [ ";
	for (size_t i = 0; i < elemCount; i++)
	{
		foutA << b[i] << ";" << endl;
	}
}
```

```
// Преобразуем сетку в СЛАУ
void FDM::transformGridToSLAE()
{
	n = elemCount;
	m = width;
	di.resize(elemCount);
	al1.resize(elemCount - 1);
	al2.resize(elemCount - width);
	au1.resize(elemCount - 1);
	au2.resize(elemCount - width);
	A2.resize(elemCount);
	for (size_t i = 0; i < elemCount; i++)
	{
		A2[i].resize(elemCount);
	}

	b.resize(elemCount);
	x.resize(elemCount);



	if (isGridUniform) {

		for (size_t elem = 0; elem < elemCount; elem++)
		{
			int i = nodes[elem].i;
			int j = nodes[elem].j;
			double x = nodes[elem].x;
			double y = nodes[elem].y;

			switch (nodes[elem].type)
			{

			case -1: {    // фиктивные узлы
				di[elem] = 1;
				A2[elem][elem] = 1;
				b[elem] = 0;
			} break;


			case 0: {     // внутренние узлы
				di[elem] = -2 / (hx * hx) - 2 / (hy * hy);
				al1[elem - 1] = 1 / (hx * hx);
				au1[elem] = 1 / (hx * hx);
				al2[elem - width] = 1 / (hy * hy);
				au2[elem] = 1 / (hy * hy);

				A2[elem][elem] = -2 / (hx * hx) - 2 / (hy * hy);
				A2[elem][elem - 1] = 1 / (hx * hx);
				A2[elem][elem + 1] = 1 / (hx * hx);
				A2[elem][elem - width] = 1 / (hy * hy);
				A2[elem][elem + width] = 1 / (hy * hy);
				b[elem] = f(x, y);
			} break;


			case 1: { // 1-е краевые условия
				di[elem] = 1;
				A2[elem][elem] = 1;
```

```
                        b[elem] = u(x, y);
                } break;


                case 3: {
                        switch (nodes[elem].border)
                        {
                        case 1: {
                                au1[elem] = 1.0 / hx;
                                di[elem] = -1.0 / hx + C1;

                                A2[elem][elem + 1] = 1.0 / hx;
                                A2[elem][elem] = -1.0 / hx + C1;
                                b[elem] = calcFirstDerivativeX(x, y) + C1 * u(x, y) + C2;
                                //
                                // <---[elem][elem+1]
                                //
                        }break;

                        default: { // 1-е краевые условия
                                di[elem] = 1;
                                A2[elem][elem] = 1;
                                b[elem] = u(x, y);
                        } break;
                        }

                }break;
                }
        }

}
else {

        for (size_t elem = 0; elem < elemCount; elem++)
        {
                int i = nodes[elem].i;
                int j = nodes[elem].j;
                double x = nodes[elem].x;
                double y = nodes[elem].y;


                switch (nodes[elem].type)
                {
                case -1: {    // фиктивные узлы
                        A2[elem][elem] = 1;
                        di[elem] = 1;
                        b[elem] = 0;
                } break;

                case 0: {     // внутренние узлы

                        /* // Я памятник себе воздвиг нерукотворный
                        // Металлов твёрже он и выше пирамид...
                        hx = xLeft + i * dx;
                        hy = yLower + j * dy;
                        double hxPrev = hx - dx;
                        double hyPrev = hy - dy;
                        */

                        hx = nodes[elem + 1].x - nodes[elem].x;
```

```
                        hy = nodes[elem + width].y - nodes[elem].y;
                        hxPrev = nodes[elem].x - nodes[elem - 1].x;
                        hyPrev = nodes[elem].y - nodes[elem - width].y;

                        di[elem] = -2.0 / (hxPrev * hx) - 2.0 / (hyPrev * hy);
                        al1[elem - 1] = 2.0 / (hxPrev*(hx + hxPrev));
                        au1[elem] = 2.0 / (hx*(hx + hxPrev));
                        al2[elem - width] = 2.0 / (hyPrev*(hy + hyPrev));
                        au2[elem] = 2.0 / (hy*(hy + hyPrev));

                        A2[elem][elem] = -2.0 / (hxPrev * hx) - 2.0 / (hyPrev * hy);
                        A2[elem][elem - 1] = 2.0 / (hxPrev*(hx + hxPrev));
                        A2[elem][elem + 1] = 2.0 / (hx*(hx + hxPrev));
                        A2[elem][elem - width] = 2.0 / (hyPrev*(hy + hyPrev));
                        A2[elem][elem + width] = 2.0 / (hy*(hy + hyPrev));
                        b[elem] = f(x, y);
                    } break;

                    case 1: { // 1-е краевые условия
                        A2[elem][elem] = 1;
                        di[elem] = 1;
                        b[elem] = u(x, y);
                    } break;
                    case 3: {
                        switch (nodes[elem].border)
                        {
                        case 1: {
                            hx = nodes[elem + 1].x - nodes[elem].x;

                            au1[elem] = 1.0 / hx;
                            di[elem] = -1.0 / hx + C1;

                            A2[elem][elem + 1] = 1.0 / hx;
                            A2[elem][elem] = -1.0 / hx + C1;
                            b[elem] = calcFirstDerivativeX(x, y) + C1 * u(x, y) + C2;
                            //
                            // <---[elem][elem+1]
                            //
                        }break;

                        default: { // 1-е краевые условия
                            di[elem] = 1;
                            A2[elem][elem] = 1;
                            b[elem] = u(x, y);
                        } break;
                        }

                    }break;
                    }
                }
        }
}



// Абсолютная невязка
double FDM::calcAbsResidual(const string &filepath) {

        ofstream fout(filepath);
        double tmp = 0.0, normAbsRes = 0.0;
```

```
	int count = 0;
	for (size_t elem = 0; elem < elemCount; elem++)
	{
		if (nodes[elem].isFirstNode == true && nodes[elem].type != -1) {
			tmp = abs(u(nodes[elem].x, nodes[elem].y) - x[elem]);
			fout << tmp << endl;
			normAbsRes += tmp * tmp;
			count++;
		}
		else
			fout << 0 << endl;
	}

	fout.close();
	cout << "Count of main nodes:\t" << count << endl;
	return sqrt(normAbsRes);
}


void FDM::checkAnswer() {

	ifstream fin("tables/x.txt");
	ofstream fout("tables/abs_res.txt");

	xExp.resize(elemCount);
	for (size_t i = 0; i < elemCount; i++)
	{
		fin >> xExp[i];
		if (nodes[i].type == -1)
			fout << 0 << endl;
		else
			fout << (xExp[i] - u(nodes[i].x, nodes[i].y)) << endl;
	}

	fin.close();
	fout.close();
}
```

**main.cpp**

```
#include "fdm.h"


// Исследование работы МКР на функциях u,f
double performSingleTest(function2D &u, function2D&f, int index, bool isGridUniform, int
condNumber, bool useJacopbNotGaussSeidel, int coef = 0) {

	string absResidualFile = "tables/Uniform_" + to_string(index) + "_" +
to_string(condNumber) + "_" + to_string(coef) + "_AbsResidual.txt";

	FDM fdm;
	fdm.init(u, f, isGridUniform, condNumber, coef);
	fdm.inputEquationParameters();
	fdm.inputSLAEParameters();
	fdm.inputGrid();
	fdm.buildGrid();
	//fdm.showGrid();
	fdm.transformGridToSLAE();
	fdm.convMatrixToDense();
	//fdm.outputSLAE("tables/Uniform_" + to_string(index) + "_A.txt");
	fdm.generateInitualGuess();
```

```
	cout << "Count of steps: " << fdm.calcIterative(useJacopbNotGaussSeidel, 0.8) << endl
<< endl;
	return fdm.calcAbsResidual(absResidualFile);
}


void exploreConvergence(function2D &u, function2D&f, int index, bool isGridUniform, int
condNumber, bool useJacopbNotGaussSeidel) {

	ofstream fout;
	string prefix = "";
	if (!isGridUniform)
		prefix = "Non";

	fout.open("tables/Convergence" + prefix + "Uniform" + to_string(index) + "_" +
to_string(condNumber) + ".txt");
	fout << std::scientific;

	// Увеличиваем сетку в kx^coef раз
	for (size_t coef = 0; coef < 4; coef++)
		fout << performSingleTest(u, f, index, isGridUniform, condNumber,
useJacopbNotGaussSeidel, coef) << endl;

	fout.close();
}



// Отрисовка сетки
void drawGrids(bool isGridUniform) {

	string prefix = "";
	if (!isGridUniform)
		prefix = "Non";

	function2D u = { [](double x, double y) -> double { return x + y; } };
	function2D f = { [](double x, double y) -> double { return 0; } };

	for (size_t coef = 0; coef < 4; coef++)
	{
		string gridFile = "grids/" + prefix + "Uniform_" + to_string(coef) + ".txt";
		string gridBorderFile = "grids/Border" + prefix + "Uniform_" + to_string(coef)
+ ".txt";

		FDM fdm;
		fdm.init(u, f, isGridUniform, 1, coef);
		//fdm.inputEquationParameters();
		//fdm.inputSLAEParameters();
		fdm.inputGrid();
		fdm.buildGrid();
		fdm.saveGridAndBorder(gridFile, gridBorderFile);

		string runVisualisation = "python plot.py " + gridFile + " " + gridBorderFile;
		system(runVisualisation.c_str());
	}
}
```

```
int main() {

	int testsCount = 7;
	bool isGridUniform;

	vector <function2D> function_u(testsCount), function_f(testsCount);
	function_u[0] = { [](double x, double y) -> double { return x + y; } };
	function_f[0] = { [](double x, double y) -> double { return 0; } };

	function_u[1] = { [](double x, double y) -> double { return 10 * x + y; } };
	function_f[1] = { [](double x, double y) -> double { return 0; } };

	function_u[2] = { [](double x, double y) -> double { return pow(x,2) + pow(y,2); } };
	function_f[2] = { [](double x, double y) -> double { return 4; } };

	function_u[3] = { [](double x, double y) -> double { return  pow(x,3) + pow(y,3); } };
	function_f[3] = { [](double x, double y) -> double { return 6 * (x + y); } };

	function_u[4] = { [](double x, double y) -> double { return pow(x,4) + pow(y,4); } };
	function_f[4] = { [](double x, double y) -> double { return 12 * (pow(x,2) +
pow(y,2)); } };

	function_u[5] = { [](double x, double y) -> double { return sin(x) + cos(y); } };
	function_f[5] = { [](double x, double y) -> double { return -sin(x) - cos(y); } };

	function_u[6] = { [](double x, double y) -> double { return exp(x + y); } };
	function_f[6] = { [](double x, double y) -> double { return 2 * exp(x + y); } };


	// Таблица 0
	// Отрисовка сеток
	drawGrids(true);
	drawGrids(false);



	// Таблица 1
	// Точность на разных функциях
	/*std::ofstream fout1("tables/table1.txt");
	std::ofstream fout3("tables/table3.txt");
	fout1 << std::scientific;
	fout3 << std::scientific;
	for (size_t i = 0; i < testsCount; i++) {
		fout1 << performSingleTest(function_u[i], function_f[i], i, true, 1, true, 0)
<< endl;
		fout3 << performSingleTest(function_u[i], function_f[i], i, true, 3, true, 0)
<< endl;
	}
	fout1.close();
	fout3.close();*/


	// Таблица 2
	// Точность в зависимости от дробления сетки
	/*for (size_t i = 0; i < testsCount; i++) {
		exploreConvergence(function_u[i], function_f[i], i, true, 1, true);
		exploreConvergence(function_u[i], function_f[i], i, false, 1, true);
		exploreConvergence(function_u[i], function_f[i], i, true, 3, true);
		exploreConvergence(function_u[i], function_f[i], i, false, 3, true);
	}*/
```

```
        return 0;
}
```