МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Новосибирский государственный технический университет»

НГТУ НЭТИ

Кафедра прикладной математики

Практическая работа №1

по дисциплине «Уравнения математической физики»

ФПМИ

Группа	ПМ-92
Вариант	4
Студенты	Кутузов Иван
	Иванов Владислав
Преподаватели	Патрушев И. И.
	Задорожный А. Г.
Дата	06.03.2022

Новосибирск

Цель работы

Разработать программу решения эллиптической краевой задачи методом конечных разностей.

Вариант 4: Область имеет T-образную форму. Предусмотреть учет первых и третьих краевых условий.

Анализ

Метод конечных разностей [1] основан на разложении функции нескольких независимых переменных в окрестности заданной точки в ряд Тейлора:

$$u(x_1+h_1,\dots,x_n+h_n) = u(x_1,\dots,x_n)+\sum_{j=1}^{n} h_j\frac{\delta}{\delta x_j}u(x_1,\dots x_n)+\frac{1}{2}(\sum_{j=1}^{n} h_j\frac{\delta}{\delta x_j})^2 u(x_1,\dots,x_n)+$$
$$\dots+\frac{1}{k!}(\sum_{j=1}^{n} h_j\frac{\delta}{\delta x_j})^k u(x_1,\dots,x_n)+\frac{1}{(k+1)!}(\sum_{j=1}^{n} h_j\frac{\delta}{\delta x_j})^{k+1} u(\xi_1,\dots,\xi_n)$$

где h_j - произвольные приращения соответствующих аргументов $\xi_j \in [x_j, x_j+h_j]$, функция $u(x_1,\dots,x_n)$ обладает ограниченными производными до $(k+1)$-го порядка включительно.

При использовании двух слагаемых при разложении функции в ряд Тейлора производные первого порядка могут быть аппроксимированы следующими конечными разностями первого порядка:

$$\nabla_h^+ u_i = \frac{u_{i+1}-u_i}{h_i}$$
$$\nabla_h^- u_i = \frac{u_i-u_{i-1}}{h_{i-1}}$$
$$\overline{\nabla}_h u_i = \frac{u_{i+1}-u_{i-1}}{h_i+h_{i-1}}$$

где $\nabla_h^+ u_i$ - правая разность, $\nabla_h^- u_i$ - левая разность, $\overline{\nabla}_h u_i$ - двусторонняя разность первого порядка.

Через конечные разности первого порядка рекуррентно могут быть определены разности второго и более высокого порядка, аппроксимирующие различные производные. На неравномерной сетке производная второго порядка может быть получена следующим образом:

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

Если сетка равномерная, то

$$V_h u_i = \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}$$

Пусть двумерная область Ω двумерная и определена прямоугольная сетка Ω_h как совокупность точек $(x_1, y_1), \ldots, (x_n, y_1), (x_1, y_2), \ldots, (x_n, y_2), (x_1, y_m), \ldots, (x_n, y_m)$
Тогда для двумерного оператора Лапласа

$$Vu = \frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2}$$

дискретный аналог на неравномерной прямоугольной сетке может быть определен таким образом:

$$V_h u_{i,j} = \frac{2u_{i-1,j}}{h_{i-1}^x(h_i^x + h_{i-1}^x)} + \frac{2u_{i,j-1}}{h_{j-1}^y(h_j^y + h_{j-1}^y)} + \frac{2u_{i+1,j}}{h_i^x(h_i^x + h_{i-1}^x)} + \frac{2u_{i,j+1}}{h_j^y(h_j^x + h_{j-1}^y)} - (\frac{2}{h_{i-1}^x h_i^x} + \frac{2}{h_{j-1}^y h_j^y})u_{i,j}$$

На равномерной сетке пятиточечный оператор выглядит следующим образом

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

Учет краевых условий первого рода

Для узлов, расположенных на границе S_1, на которых заданы краевые условия первого рода, соответствующие разностные уравнения заменяются соотношениями точно передающими краевые условия, т.е диагональные элементами матрицы, соответствующие этим узлам заменяются на 1, а соответствующий элемент вектора правой части заменяются на значение u_g функции в этом узле.

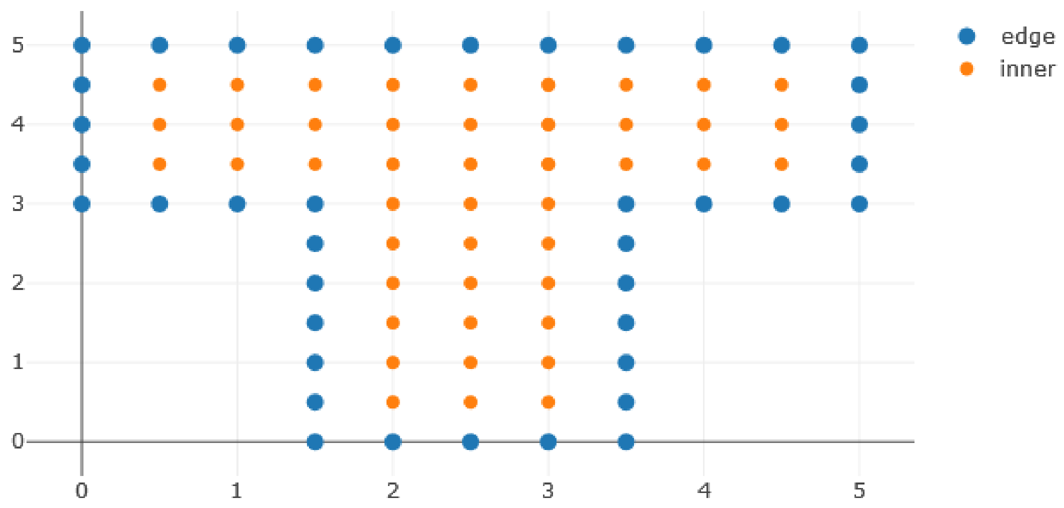
Учет краевых условий второго и третьего рода

Если расчетная область представляет собой прямоугольник со сторонами параллельными координатным осям, то направление нормали к границе S_2 и S_3, на которых заданы краевые условия второго и третьего рода, совпадает с одной из координатных линий, и тогда методы аппроксимации производной по нормали $\frac{\delta u}{\delta n}$

Тестирование

Тест

Функция: x
Правая часть: 0
Равномерная сетка, заданы условия 1-го рода на всей границе



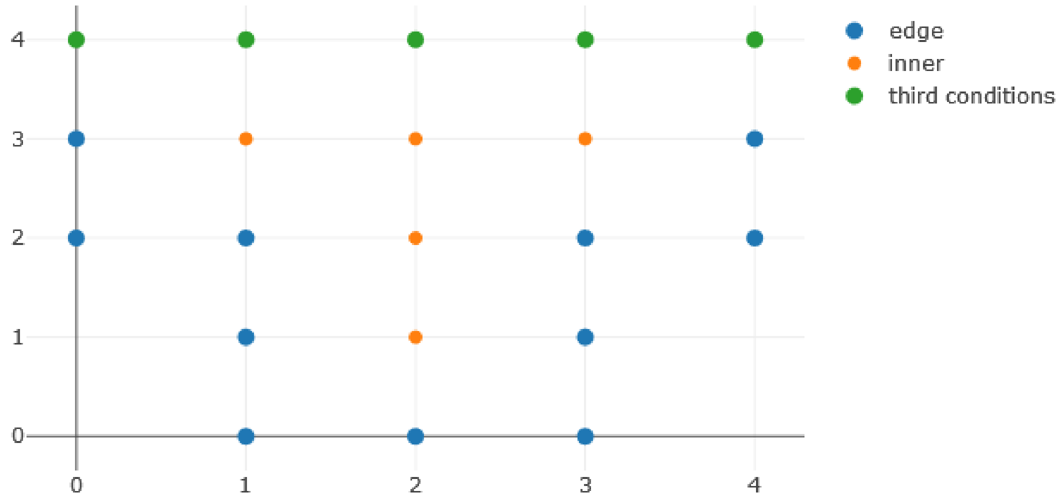
Точка	Точное	Численное	Вектор погрешности	Погрешность
0.0 3.0	0.00	0.0000000000000000	0.00e+00	8.25e-15
0.0 3.5	0.00	0.0000000000000000	0.00e+00	
0.0 4.0	0.00	0.0000000000000000	0.00e+00	
0.0 4.5	0.00	0.0000000000000000	0.00e+00	
0.0 5.0	0.00	0.0000000000000000	0.00e+00	
0.5 3.0	0.50	0.5000000000000000	0.00e+00	
0.5 3.5	0.50	0.4999999999999998	2.22e-16	
0.5 4.0	0.50	0.4999999999999997	2.78e-16	
0.5 4.5	0.50	0.4999999999999998	2.22e-16	
0.5 5.0	0.50	0.5000000000000000	0.00e+00	
1.0 3.0	1.00	1.0000000000000000	0.00e+00	
1.0 3.5	1.00	0.9999999999999994	5.55e-16	
1.0 4.0	1.00	0.9999999999999993	6.66e-16	
1.0 4.5	1.00	0.9999999999999996	4.44e-16	
1.0 5.0	1.00	1.0000000000000000	0.00e+00	
1.5 0.0	1.50	1.5000000000000000	0.00e+00	
1.5 0.5	1.50	1.5000000000000000	0.00e+00	
1.5 1.0	1.50	1.5000000000000000	0.00e+00	
1.5 1.5	1.50	1.5000000000000000	0.00e+00	

1.5 2.0	1.50	1.5000000000000000	0.00e+00	
1.5 2.5	1.50	1.5000000000000000	0.00e+00	
1.5 3.0	1.50	1.5000000000000000	0.00e+00	
1.5 3.5	1.50	1.4999999999999991	8.88e-16	
1.5 4.0	1.50	1.4999999999999989	1.11e-15	
1.5 4.5	1.50	1.4999999999999991	8.88e-16	
1.5 5.0	1.50	1.5000000000000000	0.00e+00	
2.0 0.0	2.00	2.0000000000000000	0.00e+00	
2.0 0.5	2.00	1.9999999999999996	4.44e-16	
2.0 1.0	2.00	1.9999999999999993	6.66e-16	
2.0 1.5	2.00	1.9999999999999991	8.88e-16	
2.0 2.0	2.00	1.9999999999999993	6.66e-16	
2.0 2.5	2.00	1.9999999999999996	4.44e-16	
2.0 3.0	2.00	2.0000000000000000	0.00e+00	
2.0 3.5	2.00	1.9999999999999987	1.33e-15	
2.0 4.0	2.00	1.9999999999999984	1.55e-15	
2.0 4.5	2.00	1.9999999999999987	1.33e-15	
2.0 5.0	2.00	2.0000000000000000	0.00e+00	
2.5 0.0	2.50	2.5000000000000000	0.00e+00	
2.5 0.5	2.50	2.4999999999999991	8.88e-16	
2.5 1.0	2.50	2.4999999999999991	8.88e-16	
2.5 1.5	2.50	2.4999999999999991	8.88e-16	
2.5 2.0	2.50	2.4999999999999991	8.88e-16	
2.5 2.5	2.50	2.4999999999999991	8.88e-16	
2.5 3.0	2.50	2.4999999999999991	8.88e-16	
2.5 3.5	2.50	2.4999999999999982	1.78e-15	
2.5 4.0	2.50	2.4999999999999982	1.78e-15	
2.5 4.5	2.50	2.4999999999999982	1.78e-15	
2.5 5.0	2.50	2.5000000000000000	0.00e+00	
3.0 0.0	3.00	3.0000000000000000	0.00e+00	
3.0 0.5	3.00	2.9999999999999991	8.88e-16	
3.0 1.0	3.00	2.9999999999999991	8.88e-16	
3.0 1.5	3.00	2.9999999999999991	8.88e-16	
3.0 2.0	3.00	2.9999999999999991	8.88e-16	
3.0 2.5	3.00	2.9999999999999991	8.88e-16	
3.0 3.0	3.00	3.0000000000000000	0.00e+00	
3.0 3.5	3.00	2.9999999999999982	1.78e-15	
3.0 4.0	3.00	2.9999999999999982	1.78e-15	
3.0 4.5	3.00	2.9999999999999982	1.78e-15	
3.0 5.0	3.00	3.0000000000000000	0.00e+00	
3.5 0.0	3.50	3.5000000000000000	0.00e+00	
3.5 0.5	3.50	3.5000000000000000	0.00e+00	
3.5 1.0	3.50	3.5000000000000000	0.00e+00	
3.5 1.5	3.50	3.5000000000000000	0.00e+00	
3.5 2.0	3.50	3.5000000000000000	0.00e+00	
3.5 2.5	3.50	3.5000000000000000	0.00e+00	
3.5 3.0	3.50	3.5000000000000000	0.00e+00	
3.5 3.5	3.50	3.4999999999999982	1.78e-15	
3.5 4.0	3.50	3.4999999999999982	1.78e-15	
3.5 4.5	3.50	3.4999999999999982	1.78e-15	
3.5 5.0	3.50	3.5000000000000000	0.00e+00	
4.0 3.0	4.00	4.0000000000000000	0.00e+00	
4.0 3.5	4.00	3.9999999999999982	1.78e-15	
4.0 4.0	4.00	3.9999999999999982	1.78e-15	

4.0 4.5	4.00	3.9999999999999982	1.78e-15	
4.0 5.0	4.00	4.0000000000000000	0.00e+00	
4.5 3.0	4.50	4.5000000000000000	0.00e+00	
4.5 3.5	4.50	4.4999999999999982	1.78e-15	
4.5 4.0	4.50	4.4999999999999982	1.78e-15	
4.5 4.5	4.50	4.4999999999999982	1.78e-15	
4.5 5.0	4.50	4.5000000000000000	0.00e+00	
5.0 3.0	5.00	5.0000000000000000	0.00e+00	
5.0 3.5	5.00	5.0000000000000000	0.00e+00	
5.0 4.0	5.00	5.0000000000000000	0.00e+00	
5.0 4.5	5.00	5.0000000000000000	0.00e+00	
5.0 5.0	5.00	5.0000000000000000	0.00e+00	

Тест

Функция: y
Правая часть: 0
Метод: Равномерная сетка, третьи краевые условия на верхней границе, первые краевые по остальной границе



Точка	Точное	Численное	Вектор погрешности	Погрешность
0.0 2.0	2.00	2.0000000000000000	0.00e+00	1.20e-07
0.0 3.0	3.00	3.0000000000000000	0.00e+00	
0.0 4.0	4.00	4.0000000413701855	4.14e-08	
1.0 0.0	0.00	0.0000000000000000	0.00e+00	
1.0 1.0	1.00	1.0000000000000000	0.00e+00	
1.0 2.0	2.00	2.0000000000000000	0.00e+00	
1.0 3.0	3.00	3.0000000187978748	1.88e-08	
1.0 4.0	4.00	4.0000000507691231	5.08e-08	
2.0 0.0	0.00	0.0000000000000000	0.00e+00	
2.0 1.0	1.00	1.0000000016281505	1.63e-09	
2.0 2.0	2.00	2.0000000065126371	6.51e-09	
2.0 3.0	3.00	3.0000000244224427	2.44e-08	
2.0 4.0	4.00	4.0000000535814069	5.36e-08	
3.0 0.0	0.00	0.0000000000000000	0.00e+00	
3.0 1.0	1.00	1.0000000000000000	0.00e+00	
3.0 2.0	2.00	2.0000000000000000	0.00e+00	
3.0 3.0	3.00	3.0000000187978912	1.88e-08	
3.0 4.0	4.00	4.0000000507691311	5.08e-08	
4.0 2.0	2.00	2.0000000000000000	0.00e+00	
4.0 3.0	3.00	3.0000000000000000	0.00e+00	
4.0 4.0	4.00	4.0000000413701855	4.14e-08	

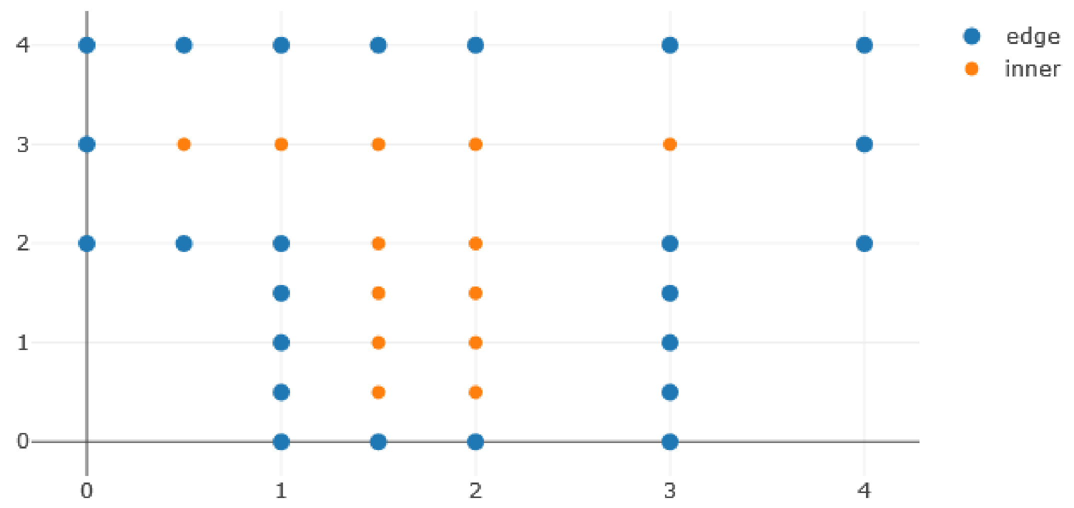
Тест

Функция: x^2
Правая часть: 0
Равномерная сетка, третьи краевые условия на верхней границе, первые краевые по остальной границе.

Точка	Точное	Численное	Вектор погрешности	Погрешность
0.0 2.0	0.00	0.0000000000000000	0.00e+00	1.26e-13
0.0 3.0	0.00	0.0000000000000000	0.00e+00	
0.0 4.0	0.00	0.0000000000000000	0.00e+00	
1.0 0.0	1.00	1.0000000000000000	0.00e+00	
1.0 1.0	1.00	1.0000000000000000	0.00e+00	
1.0 2.0	1.00	1.0000000000000000	0.00e+00	
1.0 3.0	1.00	0.9999999999999346	6.54e-14	
1.0 4.0	1.00	0.9999999999999674	3.26e-14	
2.0 0.0	4.00	4.0000000000000000	0.00e+00	
2.0 1.0	4.00	3.9999999999999658	3.42e-14	
2.0 2.0	4.00	3.9999999999999472	5.28e-14	
2.0 3.0	4.00	3.9999999999999316	6.84e-14	
2.0 4.0	4.00	3.9999999999999658	3.42e-14	
3.0 0.0	9.00	9.0000000000000000	0.00e+00	
3.0 1.0	9.00	9.0000000000000000	0.00e+00	
3.0 2.0	9.00	9.0000000000000000	0.00e+00	
3.0 3.0	9.00	8.9999999999999751	2.49e-14	
3.0 4.0	9.00	8.9999999999999876	1.24e-14	
4.0 2.0	16.00	16.0000000000000000	0.00e+00	
4.0 3.0	16.00	16.0000000000000000	0.00e+00	
4.0 4.0	16.00	16.0000000000000000	0.00e+00	

Тест

Функция: $x + y$
Правая часть: 0
Неравномерная сетка, первые краевые условия на границе



Точка	Точное	Численное	Вектор погрешности	Погрешность
0.0 2.0	2.00	2.0000000000000000	0.00e+00	3.64e-14
0.0 3.0	3.00	3.0000000000000000	0.00e+00	
0.0 4.0	4.00	4.0000000000000000	0.00e+00	
0.5 2.0	2.50	2.5000000000000000	0.00e+00	
0.5 3.0	3.50	3.4999999999999951	4.88e-15	
0.5 4.0	4.50	4.5000000000000000	0.00e+00	
1.0 0.0	1.00	1.0000000000000000	0.00e+00	
1.0 0.5	1.50	1.5000000000000000	0.00e+00	
1.0 1.0	2.00	2.0000000000000000	0.00e+00	
1.0 1.5	2.50	2.5000000000000000	0.00e+00	
1.0 2.0	3.00	3.0000000000000000	0.00e+00	
1.0 3.0	4.00	3.9999999999999925	7.55e-15	
1.0 4.0	5.00	5.0000000000000000	0.00e+00	
1.5 0.0	1.50	1.5000000000000000	0.00e+00	
1.5 0.5	2.00	1.9999999999999896	1.04e-14	
1.5 1.0	2.50	2.4999999999999858	1.42e-14	
1.5 1.5	3.00	2.9999999999999867	1.33e-14	
1.5 2.0	3.50	3.4999999999999907	9.33e-15	
1.5 3.0	4.50	4.4999999999999920	7.99e-15	
1.5 4.0	5.50	5.5000000000000000	0.00e+00	
2.0 0.0	2.00	2.0000000000000000	0.00e+00	
2.0 0.5	2.50	2.4999999999999898	1.02e-14	
2.0 1.0	3.00	2.9999999999999853	1.47e-14	

2.0 1.5	3.50	3.4999999999999871	1.29e-14	
2.0 2.0	4.00	3.9999999999999911	8.88e-15	
2.0 3.0	5.00	4.9999999999999938	6.22e-15	
2.0 4.0	6.00	6.0000000000000000	0.00e+00	
3.0 0.0	3.00	3.0000000000000000	0.00e+00	
3.0 0.5	3.50	3.5000000000000000	0.00e+00	
3.0 1.0	4.00	4.0000000000000000	0.00e+00	
3.0 1.5	4.50	4.5000000000000000	0.00e+00	
3.0 2.0	5.00	5.0000000000000000	0.00e+00	
3.0 3.0	6.00	5.9999999999999982	1.78e-15	
3.0 4.0	7.00	7.0000000000000000	0.00e+00	
4.0 2.0	6.00	6.0000000000000000	0.00e+00	
4.0 3.0	7.00	7.0000000000000000	0.00e+00	
4.0 4.0	8.00	8.0000000000000000	0.00e+00	

Порядок аппроксимации

Функция	Равномерная	Неравномерная
$x + y$	1.97e-14	2.77e-14
$x^2 + y^2$	8.22e-14	1.10e-13
$x^3 + y^3$	3.75e-13	2.11e-01
$x^4 + y^4$	5.08e-01	2.47e+00
$x^5 + y^5$	7.87e+00	1.92e+01
$\sin x + \cos y$	5.99e-03	2.81e-02
$e^x + e^y$	9.17e-01	1.27e+00

Порядок сходимости

Функция $x + y$		
Количество узлов	Равномерная	Неравномерная
49	1.97e-14	2.77e-14
169	1.68e-14	1.57e-14
625	1.33e-14	8.07e-15
2401	9.92e-15	1.65e-15

Функция $x^2 + y^2$		
Количество узлов	Равномерная	Неравномерная
49	8.22e-14	1.10e-13
169	7.23e-14	6.36e-14
625	5.67e-14	3.17e-15
2401	4.22e-15	7.61e-16

Функция $x^3 + y^3$		
Количество узлов	Равномерная	Неравномерная
49	3.75e-13	2.11e-01
169	3.28e-13	1.84e-01
625	2.68e-13	6.12e-02
2401	2.07e-13	1.48e-02

Вывод

На равномерной сетке пятиточечный оператор Лапласа имеет второй порядок сходимости.

Исследования показывают, что метод конечных разностей имеет достаточную точность при работе с полиномами, степень которых меньше третьей. При работе с полиномами степени больше, чем третья, точность решения резко падает.

Наличие третьих краевых условий снижает точность решения.

Текст программы

Figures.h

```
class Figure {
public:
	virtual bool isEdgeNode(double x, double y, double eps) = 0;
	virtual bool isInnerNode(double x, double y, double eps) = 0;

	virtual bool isLeft(double x, double y, double eps) = 0;
	virtual bool isRight(double x, double y, double eps) = 0;
	virtual bool isBottom(double x, double y, double eps) = 0;
	virtual bool isTop(double x, double y, double eps) = 0;
};

class TShapedFigure : public Figure {
private:
	double _scale;
	double _ratio;

public:
	TShapedFigure(double scale, double ratio);

	bool isEdgeNode(double x, double y, double eps) override;
	bool isInnerNode(double x, double y, double eps) override;

	bool isLeft(double x, double y, double eps) override;
	bool isRight(double x, double y, double eps) override;
	bool isBottom(double x, double y, double eps) override;
	bool isTop(double x, double y, double eps) override;
};

TShapedFigure::TShapedFigure(double scale, double ratio) {
	_scale = scale;
	_ratio = ratio;
}

bool TShapedFigure::isEdgeNode(double x, double y, double eps) {
	double connectionHeight = _scale - _ratio * _scale;
	double withOfT = _ratio * _scale;

	if (y < connectionHeight + eps && y > connectionHeight - eps) {
		return x < _scale / 2 - _ratio / 2 + eps || x > _scale / 2 + _ratio
/ 2 - eps;
	}

	if (y < connectionHeight + eps) {
		return ((y < 0.0 + eps && y > 0.0 - eps) && (x < _scale / 2 +
withOfT / 2 + eps && x > _scale / 2 - withOfT / 2 - eps)) ||
			(x < _scale / 2 - withOfT / 2 + eps && x > _scale / 2 -
withOfT / 2 - eps) ||
			(x < _scale / 2 + withOfT / 2 + eps && x > _scale / 2 +
withOfT / 2 - eps);
	}

	return (x < 0.0 + eps && x > 0.0 - eps) ||
		(x < _scale + eps && x > _scale - eps) ||
		(y < _scale + eps && y > _scale - eps);
}

bool TShapedFigure::isInnerNode(double x, double y, double eps) {
	double connectionHeight = _scale - _ratio * _scale;
	double withOfT = _ratio * _scale;

	if (y < connectionHeight + eps && y > connectionHeight - eps) {
		return x < _scale / 2 + withOfT / 2 + eps && x > _scale / 2 -
```

```
withOfT / 2 - eps;
	}

	if (y < connectionHeight + eps) {
		return x < _scale / 2 + withOfT / 2 + eps && x > _scale / 2 -
withOfT / 2 - eps && y > 0.0 - eps;
	}

	return y < _scale + eps && x > 0.0 + eps && x < _scale - eps;
}

bool TShapedFigure::isRight(double x, double y, double eps) {
	double connectionHeight = _scale - _ratio * _scale;
	double withOfT = _ratio * _scale;

	if (y < connectionHeight + eps) {
		return (x < _scale / 2 + withOfT / 2 + eps && x > _scale / 2 +
withOfT / 2 - eps);
	}

	return (x < _scale + eps && x > _scale - eps);
}

bool TShapedFigure::isLeft(double x, double y, double eps) {
	double connectionHeight = _scale - _ratio * _scale;
	double withOfT = _ratio * _scale;

	if (y < connectionHeight + eps) {
		return (x < _scale / 2 - withOfT / 2 + eps && x > _scale / 2 -
withOfT / 2 - eps);
	}

	return (x < 0.0 + eps && x > 0.0 - eps);
}

bool TShapedFigure::isBottom(double x, double y, double eps) {
	double connectionHeight = _scale - _ratio * _scale;
	double withOfT = _ratio * _scale;

	return (y < 0.0 + eps && y > 0.0 - eps) && (x < _scale / 2 + withOfT / 2 +
eps && x > _scale / 2 - withOfT / 2 - eps) ||
		(y < connectionHeight + eps && y > connectionHeight - eps);
}

bool TShapedFigure::isTop(double x, double y, double eps) {
	return (y < _scale + eps && y > _scale - eps);
}

class Square : public Figure {
private:
	double _scale;

public:
	Square(double scale);

	bool isEdgeNode(double x, double y, double eps) override;
	bool isInnerNode(double x, double y, double eps) override;

	bool isLeft(double x, double y, double eps) override;
	bool isRight(double x, double y, double eps) override;
	bool isBottom(double x, double y, double eps) override;
	bool isTop(double x, double y, double eps) override;
};

Square::Square(double scale) {
	_scale = scale;
}
```

```
bool Square::isEdgeNode(double x, double y, double eps) {
	return (x < _scale + eps && x > _scale - eps) || (x < 0.0 + eps && x > 0.0
- eps) ||
	       (y < _scale + eps && y > _scale - eps) || (y < 0.0 + eps && y > 0.0
- eps);
}

bool Square::isInnerNode(double x, double y, double eps) {
	return x < _scale + eps && y < _scale + eps && x > 0.0 - eps && y > 0.0 -
eps;
}

bool Square::isLeft(double x, double y, double eps) {
	return (x < 1.0 + eps && x > 1.0 - eps);
}

bool Square::isRight(double x, double y, double eps) {
	return (x < _scale + eps && x > _scale - eps);
}

bool Square::isBottom(double x, double y, double eps) {
	return (y < 0.0 + eps && y > 0.0 - eps);
}

bool Square::isTop(double x, double y, double eps) {
	return (y < _scale + eps && y > _scale - eps);
}
```

Grids.h

```
enum class ThirdConditionsSide {
	LEFT,
	RIGHT,
	BOTTOM,
	TOP,
	NONE,
};

class RegularGrid {
private:
	double _scale;
	double _stepX;
	double _stepY;

	Figure* _figure;
	function2D _u;
	function2D _f;
	ThirdConditionsSide _side;

	double _epsBase = 1e-5;
	double _beta = 1.0;
	double _lambda = 1.0;
	double _gamma = 0.0;

public:
	RegularGrid(double scale, double stepX, double stepY, Figure* figure,
function2D f, function2D u, ThirdConditionsSide side);

	template<typename T>
	T convertToSystemOfEquations(SystemOfEquationsFactory<T>* factory);

private:
	vector<Node> nodes();
```

```
	double leftDerivativeByX(double x, double y);
	double leftDerivativeByY(double x, double y);
	double rightDerivativeByX(double x, double y);
	double rightDerivativeByY(double x, double y);
};

RegularGrid::RegularGrid(double scale, double stepX, double stepY, Figure*
figure, function2D f, function2D u, ThirdConditionsSide side) {
	_scale = scale;
	_stepX = stepX;
	_stepY = stepY;

	_figure = figure;
	_u = u;
	_f = f;
	_side = side;
}

vector<Node> RegularGrid::nodes() {
	vector<Node> nodes = vector<Node>();

	int numberOfNodesByX = int(_scale / _stepX);
	int numberOfNodesByY = int(_scale / _stepY);

	double eps = _stepX < _stepY ? _stepX * _epsBase : _stepY * _epsBase;

	for (int i = 0; i <= numberOfNodesByX; i++) {
		for (int j = 0; j <= numberOfNodesByY; j++) {
			double x = (double)i * _stepX;
			double y = (double)j * _stepY;

			if (_figure->isEdgeNode(x, y, eps)) {
				nodes.push_back(Node(NodeType::EDGE, x, y));
			}
			else {
				if (_figure->isInnerNode(x, y, eps)) {
					nodes.push_back(Node(NodeType::INNER, x, y));
				}
				else {
					nodes.push_back(Node(NodeType::DUMMY, x, y));
				}
			}
		}
	}

	return nodes;
}

double RegularGrid::leftDerivativeByX(double x, double y) {
	double h = 1e-9;
	return (_u(x, y) - _u(x - h, y)) / h;
}

double RegularGrid::leftDerivativeByY(double x, double y) {
	double h = 1e-9;
	return (_u(x, y) - _u(x, y - h)) / h;
}

double RegularGrid::rightDerivativeByX(double x, double y) {
	double h = 1e-9;
	return (_u(x + h, y) - _u(x, y)) / h;
}

double RegularGrid::rightDerivativeByY(double x, double y) {
	double h = 1e-9;
	return (_u(x, y + h) - _u(x, y)) / h;
```

```
}

template<typename T>
T RegularGrid::convertToSystemOfEquations(SystemOfEquationsFactory<T>* factory) {
	int width = int(_scale / _stepX) + 1;

	double eps = _stepX < _stepY ? _stepX * _epsBase : _stepY * _epsBase;

	vector<Node> nodes = this->nodes();
	int countOfNodes = nodes.size();

	vector<double> di = vector<double>(countOfNodes);
	vector<double> al1 = vector<double>(countOfNodes - 1);
	vector<double> al2 = vector<double>(countOfNodes - width);
	vector<double> au1 = vector<double>(countOfNodes - 1);
	vector<double> au2 = vector<double>(countOfNodes - width);
	vector<double> b = vector<double>(countOfNodes);

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

		double x = nodes[i].x;
		double y = nodes[i].y;

		switch (nodes[i].type) {
		case NodeType::EDGE: {
			switch (_side) {
			case ThirdConditionsSide::NONE: {
				di[i] = 1.0;
				b[i] = _u(x, y);
				break;
			}
			case ThirdConditionsSide::LEFT: {
				if (_figure->isLeft(x, y, eps)) {
					auto uInPoint = _u(x, y);
					auto rightByX = rightDerivativeByX(x, y);
					auto ubeta = -_lambda * rightByX / _beta +
uInPoint;

					di[i] = _lambda / _stepY + _beta;
					au2[i] = -_lambda / _stepY;
					b[i] = -_lambda * rightByX + _beta * (uInPoint -
ubeta) + _beta * ubeta;
				}
				else {
					di[i] = 1.0;
					b[i] = _u(x, y);
				}
				break;
			}
			case ThirdConditionsSide::RIGHT: {
				if (_figure->isRight(x, y, eps)) {
					auto uInPoint = _u(x, y);
					auto leftByX = leftDerivativeByX(x, y);
					auto ubeta = _lambda * leftByX / _beta +
uInPoint;

					di[i] = _lambda / _stepY + _beta;
					al2[i - width] = -_lambda / _stepY;
					b[i] = _lambda * leftByX + _beta * (uInPoint -
ubeta) + _beta * ubeta;
				}
				else {
					di[i] = 1.0;
					b[i] = _u(x, y);
				}
				break;
			}
```

```
                    case ThirdConditionsSide::BOTTOM: {
                          if (_figure->isBottom(x, y, eps)) {
                                auto uInPoint = _u(x, y);
                                auto rightByY = rightDerivativeByY(x, y);
                                auto ubeta = -_lambda * rightByY / _beta +
uInPoint;

                                di[i] = _lambda / _stepY + _beta;
                                au1[i] = -_lambda / _stepY;
                                b[i] = -_lambda * rightByY + _beta * (uInPoint -
ubeta) + _beta * ubeta;
                          }
                          else {
                                di[i] = 1.0;
                                b[i] = _u(x, y);
                          }
                          break;
                    }
                    case ThirdConditionsSide::TOP: {
                          if (_figure->isTop(x, y, eps)) {
                                auto uInPoint = _u(x, y);
                                auto leftByY = leftDerivativeByY(x, y);
                                auto ubeta = _lambda * leftByY / _beta +
uInPoint;

                                di[i] = _lambda / _stepY + _beta;
                                al1[i - 1] = -_lambda / _stepY;
                                b[i] = _lambda * leftByY + _beta * (uInPoint -
ubeta) + _beta * ubeta;
                          }
                          else {
                                di[i] = 1.0;
                                b[i] = _u(x, y);
                          }
                          break;
                    }
                    }
                    break;
              }

              case NodeType::INNER: {
                    di[i] = _lambda *(2.0 / (_stepX * _stepX) + 2.0 / (_stepY *
_stepY)) + _gamma;
                    al1[i - 1] = -_lambda / (_stepY * _stepY);
                    au1[i] = -_lambda / (_stepY * _stepY);
                    al2[i - width] = -_lambda / (_stepX * _stepX);
                    au2[i] = -_lambda / (_stepX * _stepX);
                    b[i] = _f(x, y);
                    break;
              }

              case NodeType::DUMMY: {
                    di[i] = 1.0;
                    b[i] = 0.0;
                    break;
              }
              }
       }

       return factory->createSystem(di, au1, au2, al1, al2, b);
}

class IrregularGrid {
private:
       double _scale;
       double _stepX;
       double _stepY;
```

```
	Figure* _figure;
	function2D _u;
	function2D _f;

	ThirdConditionsSide _side;

	double _epsBase = 1e-5;
	double _beta = 1.0;
	double _lambda = 1.0;
	double _gamma = 0.0;

public:
	IrregularGrid(double scale, double stepX, double stepY, Figure* figure,
function2D f, function2D u, ThirdConditionsSide side);

	template<typename T>
	T convertToSystemOfEquations(SystemOfEquationsFactory<T>* factory);

private:
	vector<Node> nodes();
	int width();

	double leftDerivativeByX(double x, double y);
	double leftDerivativeByY(double x, double y);
	double rightDerivativeByX(double x, double y);
	double rightDerivativeByY(double x, double y);
};

IrregularGrid::IrregularGrid(double scale, double stepX, double stepY, Figure*
figure, function2D f, function2D u, ThirdConditionsSide side) {
	_scale = scale;
	_stepX = stepX;
	_stepY = stepY;

	_figure = figure;

	_f = f;
	_u = u;

	_side = side;
}

template<typename T>
T IrregularGrid::convertToSystemOfEquations(SystemOfEquationsFactory<T>* factory)
{
	vector<Node> nodes = this->nodes();
	int countOfNodes = nodes.size();
	int w = width();

	double eps = _stepX < _stepY ? _stepX * _epsBase : _stepY * _epsBase;

	vector<double> di = vector<double>(countOfNodes);
	vector<double> al1 = vector<double>(countOfNodes - 1);
	vector<double> al2 = vector<double>(countOfNodes - w);
	vector<double> au1 = vector<double>(countOfNodes - 1);
	vector<double> au2 = vector<double>(countOfNodes - w);
	vector<double> b = vector<double>(countOfNodes);

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

		double x = nodes[i].x;
		double y = nodes[i].y;

		switch (nodes[i].type) {
		case NodeType::EDGE: {
			switch (_side) {
```

```
                    case ThirdConditionsSide::NONE: {
                          di[i] = 1.0;
                          b[i] = _u(x, y);
                          break;
                    }
                    case ThirdConditionsSide::LEFT: {
                          if (_figure->isLeft(x, y, eps)) {
                                auto uInPoint = _u(x, y);
                                auto rightByX = rightDerivativeByX(x, y);
                                auto ubeta = -_lambda * rightByX / _beta +
uInPoint;

                                di[i] = _lambda / _stepY + _beta;
                                au2[i] = -_lambda / _stepY;
                                b[i] = -_lambda * rightByX + _beta * (uInPoint -
ubeta) + _beta * ubeta;
                          }
                          else {
                                di[i] = 1.0;
                                b[i] = _u(x, y);
                          }
                          break;
                    }
                    case ThirdConditionsSide::RIGHT: {
                          if (_figure->isRight(x, y, eps)) {
                                auto uInPoint = _u(x, y);
                                auto leftByX = leftDerivativeByX(x, y);
                                auto ubeta = _lambda * leftByX / _beta +
uInPoint;

                                di[i] = _lambda / _stepY + _beta;
                                al2[i - w] = -_lambda / _stepY;
                                b[i] = _lambda * leftByX + _beta * (uInPoint -
ubeta) + _beta * ubeta;
                          }
                          else {
                                di[i] = 1.0;
                                b[i] = _u(x, y);
                          }
                          break;
                    }
                    case ThirdConditionsSide::BOTTOM: {
                          if (_figure->isBottom(x, y, eps)) {
                                auto uInPoint = _u(x, y);
                                auto rightByY = rightDerivativeByY(x, y);
                                auto ubeta = -_lambda * rightByY / _beta +
uInPoint;

                                di[i] = _lambda / _stepY + _beta;
                                au1[i] = -_lambda / _stepY;
                                b[i] = -_lambda * rightByY + _beta * (uInPoint -
ubeta) + _beta * ubeta;
                          }
                          else {
                                di[i] = 1.0;
                                b[i] = _u(x, y);
                          }
                          break;
                    }
                    case ThirdConditionsSide::TOP: {
                          if (_figure->isTop(x, y, eps)) {
                                auto uInPoint = _u(x, y);
                                auto leftByY = leftDerivativeByY(x, y);
                                auto ubeta = _lambda * leftByY / _beta +
uInPoint;

                                di[i] = _lambda / _stepY + _beta;
```

```
                                al1[i - 1] = -_lambda / _stepY;
                                b[i] = _lambda * leftByY + _beta * (uInPoint -
ubeta) + _beta * ubeta;
                            }
                            else {
                                di[i] = 1.0;
                                b[i] = _u(x, y);
                            }
                            break;
                        }
                        }
                        break;
                    }

                    case NodeType::INNER: {
                        double hx = nodes[i + w].x - nodes[i].x;
                        double hy = nodes[i + 1].y - nodes[i].y;

                        double hxPrev = nodes[i].x - nodes[i - w].x;
                        double hyPrev = nodes[i].y - nodes[i - 1].y;
                        //cout << y << "," << endl;
                        di[i] = _lambda * (2.0 / (hxPrev * hx) + 2.0 / (hyPrev * hy))
+ _gamma;
                        al1[i - 1] = -2.0 * _lambda / (hyPrev * (hy +
hyPrev));//(hxPrev * (hx + hxPrev));
                        au1[i] = -2.0 * _lambda / (hy * (hy + hyPrev));//(hx * (hx +
hxPrev));
                        al2[i - w] = -2.0 * _lambda / (hxPrev * (hx + hxPrev));
                        au2[i] = -2.0 * _lambda / (hx * (hx + hxPrev));
                        b[i] = _f(x, y);
                        break;
                    }

                    case NodeType::DUMMY: {
                        di[i] = 1.0;
                        b[i] = 0.0;
                        break;
                    }
                    }
            }

            return factory->createSystem(di, au1, au2, al1, al2, b);
}

vector<Node> IrregularGrid::nodes() {
        vector<Node> nodes = vector<Node>();

        int sumByX = int(_scale / _stepX);
        int sumByY = int(_scale / _stepY);

        int stepX = 1;
        int stepY;

        double eps = _stepX < _stepY ? _stepX * _epsBase : _stepY * _epsBase;

        for (int i = 0; i <= sumByX; i += stepX) {
                stepY = 1;

                for (int j = 0; j <= sumByY; j += stepY) {
                        double x = _stepX * (double)i;
                        double y = _stepY * (double)j;

                        if (_figure->isEdgeNode(x, y, eps)) {
                                nodes.push_back(Node(NodeType::EDGE, x, y));
                        }
                        else {
                                if (_figure->isInnerNode(x, y, eps)) {
```

```
                                        nodes.push_back(Node(NodeType::INNER, x, y));
                                }
                                else {
                                        nodes.push_back(Node(NodeType::DUMMY, x, y));
                                }
                        }

                        if (j >= sumByY / 2) {
                                stepY = 2;
                        }
                }

                if (i >= sumByX / 2) {
                        stepX = 2;
                }
        }

        return nodes;
}

double IrregularGrid::leftDerivativeByX(double x, double y) {
        double h = 1e-9;
        return (_u(x, y) - _u(x - h, y)) / h;
}

double IrregularGrid::leftDerivativeByY(double x, double y) {
        double h = 1e-9;
        return (_u(x, y) - _u(x, y - h)) / h;
}

double IrregularGrid::rightDerivativeByX(double x, double y) {
        double h = 1e-9;
        return (_u(x + h, y) - _u(x, y)) / h;
}

double IrregularGrid::rightDerivativeByY(double x, double y) {
        double h = 1e-9;
        return (_u(x, y + h) - _u(x, y)) / h;
}

int IrregularGrid::width() {
        int sumByY = int(_scale / _stepY);

        int stepY = 1;

        int w = 0;

        for (int j = 0; j <= sumByY; j += stepY, w++) {
                if (j >= sumByY / 2) {
                        stepY = 2;
                }
        }

        return w;
}
```

Node.h

```
enum class NodeType {
        INNER,
        EDGE,
        DUMMY,
};
```

```
struct Node {
	NodeType type;
	double x;
	double y;

	Node(NodeType type, double x, double y) {
		this->type = type;
		this->x = x;
		this->y = y;
	}
};
```

SystemOfEquations.h

```
class SystemOfEquations {
protected:
	vector<double> _di, _au1, _au2, _al1, _al2;
	vector<double> _b;

	int _maxiter = 10000;
	double _w;
	double _eps = 1e-12;
public:
	SystemOfEquations(vector<double> di, vector<double> au1, vector<double>
au2, vector<double> al1, vector<double> al2, vector<double> b);

	vector<double> solution();

protected:
	virtual vector<double> calculateXk(vector<double> x) = 0;
	double euclideanNorm(vector<double> x);
	double relativeDiscrepancy();
	vector<double> multiplyMatrixByVector(vector<double> x);
};

class JacobiMethod : public SystemOfEquations {
public:
	JacobiMethod(vector<double> di, vector<double> au1, vector<double> au2,
vector<double> al1, vector<double> al2, vector<double> b) : SystemOfEquations(di,
au1, au2, al1, al2, b) {
		_w = .5;
	};

private:
	vector<double> calculateXk(vector<double> x) override;
	double multiplyLineByVector(int line, vector<double> x);
};

class GaussSeidelMethod : public SystemOfEquations {
public:
	GaussSeidelMethod(vector<double> di, vector<double> au1, vector<double>
au2, vector<double> al1, vector<double> al2, vector<double> b) :
SystemOfEquations(di, au1, au2, al1, al2, b) {
		_w = 1.;
	};

private:
	vector<double> calculateXk(vector<double> x) override;
	double multiplyUpperLineByVector(int line, vector<double> x);
	double multiplyLowerLineByVector(int line, vector<double> x);
};

SystemOfEquations::SystemOfEquations(vector<double> di, vector<double> au1,
```

```cpp
vector<double> au2, vector<double> al1, vector<double> al2, vector<double> b) {
	_di = di;
	_au1 = au1;
	_au2 = au2;
	_al1 = al1;
	_al2 = al2;
	_b = b;
}

vector<double> SystemOfEquations::solution() {
	int i = 0;

	vector<double> xk(_b.size(), 0.);

	while (i < _maxiter && relativeDiscrepancy() >= _eps) {
		xk = calculateXk(xk);
		i++;
	}

	return xk;
}

double SystemOfEquations::relativeDiscrepancy() {
	int n = _b.size();

	vector<double> numerator(n);
	vector<double> multiplication = multiplyMatrixByVector(_b);

	for (int i = 0; i < n; i++) {
		numerator[i] = _b[i] - multiplication[i];
	}


	return euclideanNorm(numerator) / euclideanNorm(_b);
}

double SystemOfEquations::euclideanNorm(vector<double> x) {
	double sum = 0;
	int n = x.size();

	for (int i = 0; i < n; i++) {
		sum += x[i] * x[i];
	}

	return sqrt(sum);
}

vector<double> SystemOfEquations::multiplyMatrixByVector(vector<double> x) {
	int n = x.size();
	int m = _di.size() - _al2.size();

	vector<double> result(n);

	int index = 1;
	for (int i = 0; i < _al1.size(); i++, index++) {
		result[index] += _al1[i] * x[i];
	}

	index = m;
	for (int i = 0; i < _al2.size(); i++, index++) {
		result[index] += _al2[i] * x[i];
	}

	for (int i = 0; i < _di.size(); i++) {
		result[i] += _di[i] * x[i];
	}
```

```
        index = 1;
        for (int i = 0; i < _au1.size(); i++, index++) {
                result[i] += _au1[i] * x[index];
        }

        index = m;
        for (int i = 0; i < _au2.size(); i++, index++) {
                result[i] += _au2[i] * x[index];
        }

        return result;
}

vector<double> JacobiMethod::calculateXk(vector<double> x) {
        double sum;
        int n = x.size();
        vector<double> xk(n);

        for (int i = 0; i < n; i++) {
                sum = multiplyLineByVector(i, x);

                xk[i] = x[i] + _w * (_b[i] - sum) / _di[i];
        }

        return xk;
}

double JacobiMethod::multiplyLineByVector(int line, vector<double> x) {
        int n = x.size();
        int m = _di.size() - _al2.size();
        double sum = 0;

        if (line > 0) {
                sum += _al1[line - 1] * x[line - 1];

                if (line > m) {
                        sum += _al2[line - m] * x[line - m];
                }
        }

        sum += _di[line] * x[line];

        if (line < n - 1) {
                sum += _au1[line] * x[line + 1];

                if (line < n - m) {
                        sum += _au2[line] * x[line + m];
                }
        }

        return sum;
}

vector<double> GaussSeidelMethod::calculateXk(vector<double> x) {
        double sum;
        int n = x.size();
        vector<double> xk = x;

        for (int i = 0; i < n; i++) {
                sum = multiplyUpperLineByVector(i, xk);
                sum += multiplyLowerLineByVector(i, x);

                xk[i] = x[i] + _w * (_b[i] - sum) / _di[i];
        }

        return xk;
}
```

```
double GaussSeidelMethod::multiplyUpperLineByVector(int line, vector<double> x) {
	int n = x.size();
	int m = _di.size() - _al2.size();
	double sum = 0;

	if (line > 0) {
		sum += _al1[line - 1] * x[line - 1];

		if (line > m) {
			sum += _al2[line - m] * x[line - m];
		}
	}

	return sum;
}

double GaussSeidelMethod::multiplyLowerLineByVector(int line, vector<double> x) {
	int n = x.size();
	int m = _di.size() - _al2.size();
	double sum = 0;

	sum += _di[line] * x[line];

	if (line < n - 1) {
		sum += _au1[line] * x[line + 1];

		if (line < n - m) {
			sum += _au2[line] * x[line + m];
		}
	}

	return sum;
}
```

SystemOfEquationFactories.h

```
template <typename T>
class SystemOfEquationsFactory {
public:
	virtual T createSystem(vector<double> di, vector<double> au1,
vector<double> au2, vector<double> al1, vector<double> al2, vector<double> b) =
0;
};

class JacobiSystemFactory : public SystemOfEquationsFactory<JacobiMethod> {
public:
	JacobiMethod createSystem(vector<double> di, vector<double> au1,
vector<double> au2, vector<double> al1, vector<double> al2, vector<double> b)
override;
};

class GaussSeidelSystemFactory : public
SystemOfEquationsFactory<GaussSeidelMethod> {
public:
	GaussSeidelMethod createSystem(vector<double> di, vector<double> au1,
vector<double> au2, vector<double> al1, vector<double> al2, vector<double> b)
override;
};

JacobiMethod JacobiSystemFactory::createSystem(vector<double> di, vector<double>
au1, vector<double> au2, vector<double> al1, vector<double> al2, vector<double>
b) {
	return JacobiMethod(di, au1, au2, al1, al2, b);
```

```
}

GaussSeidelMethod GaussSeidelSystemFactory::createSystem(vector<double> di,
vector<double> au1, vector<double> au2, vector<double> al1, vector<double> al2,
vector<double> b) {
	return GaussSeidelMethod(di, au1, au2, al1, al2, b);
}
```