МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ
НЭТИ | **Факультет прикладной математики и информатики**

Кафедра прикладной математики

Практическое задание № 1

по дисциплине «Компьютерные технологии моделирования и анализа данных»

ТЕХНОЛОГИЯ ВЫДЕЛЕНИЯ ПОЛЯ

Группа ПММ-42

Бригада САЛЯЕВ АРТЁМ

ЯНКОВСКИЙ ЕГОР

Преподаватель КОШКИНА ЮЛИЯ ИГОРЕВНА

Новосибирск, 2025

1. Задание

Реализовать расчет полей влияние с использованием технологии выделения части поля при решение обратной задачи.

2. Теоретическая часть

Пусть требуется найти распространение электромагнитного поля в горизонтально-слоистой среде, срез которой представлен на рисунке 1.

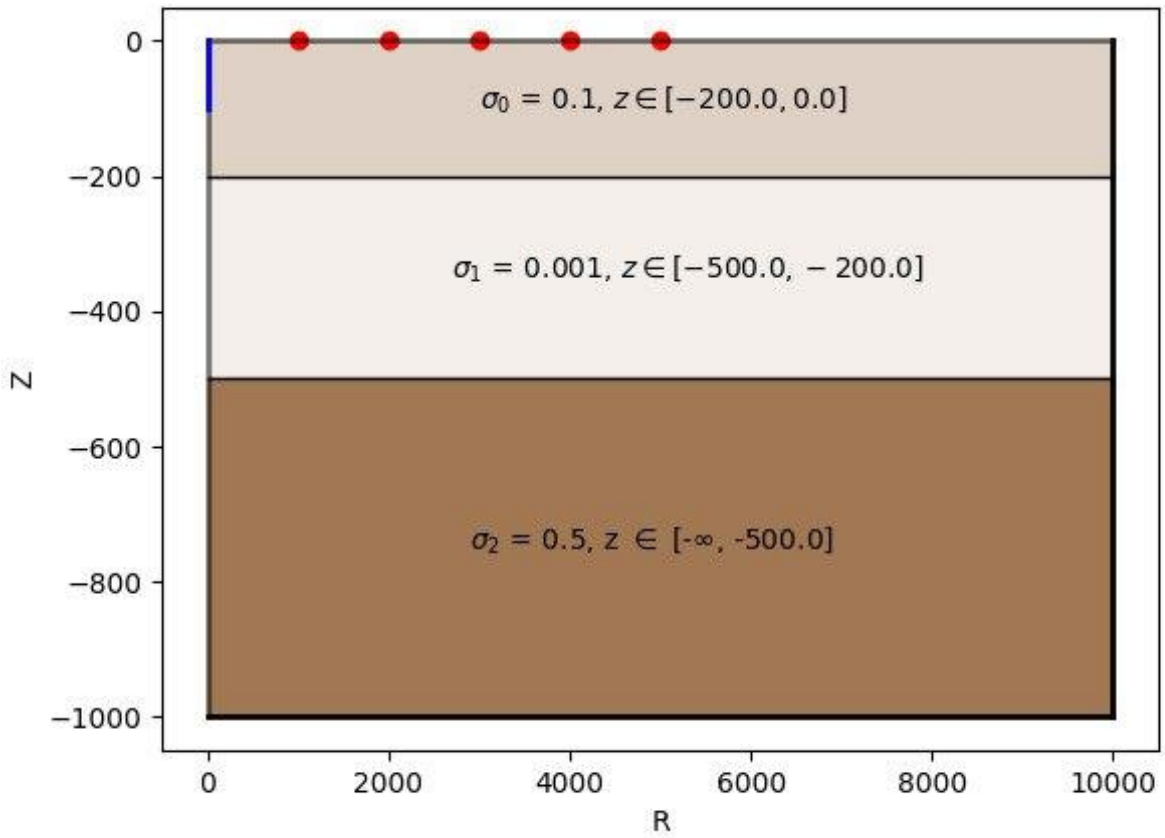

Рисунок 1. Срез горизонтально-слоистой среды.

Пусть у нас имеется источник в виде вертикальной электрической линии (на рисунке 1 отмечен синей линией сверху слева), 5 приёмников (отмечены красными точками), и 3 слоя с различными параметрами электропроводности.

Распределение электрического поля области описывается следующим эллиптическим уравнением:

$$-\mathrm{div}(\sigma grad V) = f, \qquad (1.1)$$

где σ – электропроводность слоев горизонтально-слоистой среды, V–потенциал электрического поля. Для решения задачи при данной постановке будем использовать однородные краевые условия первого и второго рода.

Для уменьшения вычислительных затрат численное решение задачи (1.1) проще искать в двумерном случае при переходе от декартовых к цилиндрическим координатам. Тогда уравнение принимает вид:

$$-\frac{1}{r}\frac{\partial}{\partial r}\left(\sigma\, r\,\frac{\partial V}{\partial r}\right)-\sigma\frac{\partial^2 V}{\partial z^2}=f, \tag{1.2}$$

с краевыми условиями:

$$V|_{\Gamma_1}=0, \tag{1.3}$$

$$\left.\frac{\partial V}{\partial n}\right|_{\Gamma_2}=0. \tag{1.4}$$

Краевые условия (1.3) – (1.4) задаются следующим образом: на верхней и левой границе однородное второго рода (на рисунке 1 они отмечены серой линией), на правой и нижней границе однородное первого рода (на рисунке 1 они отмечены чёрной линией).

Решение задачи (1.2) – (1.4) будем искать, используя метод конечных элементов. Для МКЭ переведем уравнение (1.2) в вариационную формулировку методом Галёркина:

$$\int_\Omega \sigma r\left(\frac{\partial V}{\partial r}\frac{\partial w}{\partial r}+\frac{\partial V}{\partial z}\frac{\partial w}{\partial z}\right)drdz=\int_\Omega rfw\,drdz, \tag{1.5}$$

где w – базисная функция.

Для ускорения вычислений будем использовать технологию выделения части поля. Полное решение задачи будет искаться из суммы решений:

$$V=V^n+V^+, \tag{1.6}$$

где V^n – решение в среде без аномального объекта (нормальное поле), V^+ - поле влияния аномального объекта. Нормальное поле буем искать из решения уравнения (1.8).

$$-\mathrm{div}(\sigma^n grad V^n) = f, \tag{1.7}$$

Поле влияния найдём из решения следующего уравнения (1.9).

$$-\mathrm{div}(\sigma^+ grad V^+) = -\mathrm{div}\big((\sigma^n - \sigma^+) grad V^n\big). \tag{1.8}$$

При решении обратной задачи за неизвестные параметры обозначим электропроводность слоёв, выделяющихся относительно всей горизонтально-слоистой среды (на рисунке 1 этот слой является σ_1).

Матрица для решения обратной задачи строится согласно формуле (1.9)

$$A_{ij} = \sum_{k=1}^{N} \omega_k^2 \frac{\partial\big(\delta\varepsilon_k(u)\big)}{\partial u_i} \frac{\partial\big(\delta\varepsilon_k(u)\big)}{\partial u_j}, \tag{1.9}$$

а вектор правой части, согласно формуле (1.10):

$$f_i = -\sum_{k=1}^{N} \omega_k^2 \, \delta\varepsilon_k(u^0) \frac{\partial\big(\delta\varepsilon_k(u)\big)}{\partial u_i}. \tag{1.10}$$

Итоговое СЛАУ будет иметь вид:

$$(A + \alpha I)\Delta u = f, \tag{1.11}$$

где Δu - направление поиска для вектора параметров среды, а α – вектор параметров регуляризации.

3. Практическая часть

Для начала сравним решения без использования технологии выделения поля и решения полученные при использовании технологии выделения поля сначала на одной и той же сетке, а затем на сетке, концентрированной к аномальному слою (сетка будет содержать в 4 раза меньше элементов). Результаты представлены в таблицах 1 и 2.

Приёмники	Решение с выделением	Решение без выделения	Погрешность решений
(100, 0, 0)	2.53659328e-02	2.48088263e-02	5.57106522e-04
(150, 0, 0)	8.42969594e-03	7.98022484e-03	4.49471099e-04
(200, 0, 0)	3.22982044e-03	2.95997562e-03	2.69844821e-04
(250, 0, 0)	1.30786549e-03	1.14632428e-03	1.61541215e-04
(300, 0, 0)	5.79638620e-04	5.37818281e-04	4.18203392e-05

Таблица 1. Сравнение решений прямой задачи с выделением поля на одинаковых сетках

Приёмники	Решение с выделением	Решение без выделения	Погрешность решений
(100, 0, 0)	2.53813057e-02	2.48088263e-02	5.72479422e-04
(150, 0, 0)	8.44420976e-03	7.98022484e-03	4.63984919e-04
(200, 0, 0)	3.24367588e-03	2.95997562e-03	2.83700261e-04
(250, 0, 0)	1.32123742e-03	1.14632428e-03	1.74913145e-04
(300, 0, 0)	5.92680757e-04	5.37818281e-04	5.48624762e-05

Таблица 2. Сравнение решений прямой задачи с выделением поля на разных сетках

Как можно заметить, использование разных сеток при выделении полей не внесло особых изменений в решение прямой задачи, что вполне может случиться, поскольку сложно утверждать, как поведет себя решение при использовании выделения поля. Однако, решение с сеткой, концентрирующейся к аномальному объекту, заняло в 2,5 раза меньше времени, чем с одинаковой сеткой.

Теперь рассмотрим решение обратной задачи. Пусть нам необходимо решить нелинейную обратную задачу и найти коэффициент электропроводности $\sigma^2 = 0.5$ См/м. Проведем сравнение решений при различных начальных приближениях, а также сравним решения с использованием технологии выделения поля и без.

Итерации	С использованием технологии выделения поля		Без использования технологии выделения поля	
	σ^1	Погрешность	σ^1	Погрешность
$\sigma^2{}_0 = 0.2$ См/м				
1	3.270508e-01	1.729492e-01	3.547918e-01	1.452082e-01
2	4.461609e-01	5.383911e-02	3.980200e-01	1.019800e-01
3	4.966253e-01	3.374678e-03	3.960718e-01	1.039282e-01
4	5.001439e-01	1.439423e-04	-	

$\sigma^2{}_0 = 0.789$ См/м				
1	5.501523e-01	5.015226e-02	4.674491e-01	3.255085e-02
2	4.922575e-01	7.742519e-03	5.002458e-01	2.457825e-04
3	5.002597e-01	2.596581e-04	5.001636e-01	1.636022e-04
$\sigma^2{}_0 = 0.00879$ См/м				
1	2.065977e-02	4.793402e-01	2.549715e-02	4.745029e-01
2	4.416969e-02	4.558303e-01	4.584110e-02	4.541589e-01
3	8.892704e-02	4.110730e-01	1.651118e-01	3.348882e-01
4	1.677063e-01	3.322937e-01	1.592971e-01	3.407029e-01
5	2.860339e-01	2.139661e-01	3.892400e-01	1.107600e-01
6	4.150735e-01	8.492646e-02	4.052378e-01	9.476220e-02
7	4.891660e-01	1.083402e-02	3.965366e-01	1.034634e-01
8	5.002934e-01	2.934260e-04	3.957860e-01	1.042140e-01
$\sigma^2{}_0 = 1^{-6}$ См/м				
1	1.827025e-03	4.981730e-01	7.254274e-03	4.927457e-01
2	6.281662e-03	4.937183e-01	1.873751e-02	4.812625e-01
3	1.557112e-02	4.844289e-01	3.926346e-02	4.607365e-01
4	3.422698e-02	4.657730e-01	6.910089e-02	4.308991e-01
5	7.028568e-02	4.297143e-01	1.268344e-01	3.731656e-01
6	1.359135e-01	3.640865e-01	1.857701e-01	3.142299e-01
7	2.413998e-01	2.586002e-01	2.394016e-01	2.605984e-01
8	3.732648e-01	1.267352e-01	3.448615e-01	1.551385e-01
9	4.727726e-01	2.722740e-02	3.997954e-01	1.002046e-01
10	4.998058e-01	1.942439e-04	3.972946e-01	1.027054e-01
11	5.000096e-01	9.574702e-06	-	-

4. Выводы

1. С технологией выделения поля: Метод демонстрирует устойчивую сходимость к точному решению (~5.0e-01) во всех случаях, независимо от грубости начального приближения. В то время как без использования схемы с выделением поля решение находилось быстрее, но с весомой, относительно первого способа, погрешностью.

2. При решении обратных задач без использования схемы выделения поля, результат очень зависит от начального приближения. При тестировании были получены случаи, когда программа не могла найти правильное решение параметра электропроводности. При использовании выделения поля, даже при начальной ошибке в 5 порядков ($\sigma^2{}_0 = 1^{-6}$) он уверенно находит решение за 11 итераций.

3. С использованием технологии выделения поля значения σ на каждом шаге закономерно приближаются к истинному, что свидетельствует о хорошей обусловленности задачи после применения технологии выделения поля. Без технологии наблюдается "блуждающее" поведение: значения могут то приближаться, то удаляться от цели.

5. Текст программы

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <locale.h>
#include <Windows.h>
#include <fstream>
#include <vector>
#include <algorithm>
#include <iostream>
#include <iomanip>

using namespace std;

int maxiter = 100; // максимальное количество итераций прямая задача
int max_iter = 100; // максимальное количество итераций обратная задача
double eps = 1e-15; // величина требуемой относительной невзяки
int n = 0; // количество конечных элементов
int nv = 0; // количество узлов в сетке
int ne = 0; // количество краевых условий
int nn = 0; // количество главных условий
double r_min = 0; // наименьшая координата по r
double r_max = 0; // наибольшая координата по r
double z_min = 0; // наименьшая координата по z
double z_max = 0; // наибольшая координата по z
int nr_full = 0; // количество точек по r
int nz_full = 0; // количество точек по z
int n1 = 0; // количество 1 краевых
int n2 = 0; // количество 2 краевых

vector<vector<double>> G(4); // матрица жесткости
vector<vector<double>> M(4); // матрица массы
vector<vector<double>> localA(4); // локальная матрица
vector<double> localb(4); // локальный вектор
vector<vector<double>> grid; // матрица координат узлов
vector<vector<int>> num_elem; // матрица глобальных номеров узлов
vector<vector<int>> edge1; // 1 краевые условия (узел1, узел2, формула)
vector<vector<int>> edge2; // 2 краевые условия (узел1, узел2, формула)
vector<double> localb2(2); // для учета 2 и 3 краевых
vector<vector<double>> locala2(2); // для учета 3 краевых

vector<double> gridr;
vector<double> gridz;
vector<vector<double>> sloy;
vector<vector<double>> sloy_dop;


vector<int> ig; // строки
vector<int> jg; // столбцы
vector<double> al; // нижний треугольник
vector<double> au; // верхний треугольник
vector<double> di; // диагональ
vector<double> b; // вектор правой части
// для ЛОС
vector<double> L; // нижний треугольник
vector<double> U; // верхний треугольник
vector<double> D; // диагональ
vector<double> x0;
vector<double> z;
vector<double> r;
vector<double> y;
vector<double> p;
vector<double> t;
// обратная задача
vector<double> true_eps(5); // истинное решение
```

```
vector<double> tmp_eps(5); // решение на текущей итерации прямой задачи
vector<double> next_eps(5); // решение на следующей итерации прямой задачи
vector<double> delta_tmp_eps(5); // решение на текущей итерации прямой задачи с прира-
щением
double start_u = 0.7; // начальное приближение (u0)
double tmp_u; // решение на текущей итерации обратной задачи
double delta_u; // приращение на текущей итерации
double w = 1; // веса
double alpha = 0;
double betta = 1;
double gamma = 1e-4;
double tok = 1; // ток

vector<double> pointz; // контрольные точки по z
vector<double> nz; // количество интервалов по z
vector<double> kz; // коэффициент разрядки по z
vector<int> kz_route; // направление разрядки

vector<double> pointr; // контрольные точки по r
vector<double> nr; // количество интервалов по r
vector<double> kr; // коэффициент разрядки по r
vector<int> kr_route; // направление разрядки

ifstream input("input.txt");
vector<double> qn;
vector<double> qv;
vector<double> q_dop;
vector<vector<double>> grid_n;
vector<vector<int>> num_elem_n;
vector<vector<double>> grid_dop;
vector<vector<int>> num_elem_dop;

// подсчет в заданной точке
double result_q(double r, double z, vector<double> q, vector<vector<double>> grid,
vector<vector<int>> num_elem)
{
	double res = 0;
	// определить, какому элементу принадлежит
	for (int i = 0; i < num_elem.size(); i++)
	{
		double r0 = grid[num_elem[i][0]][0];
		double r1 = grid[num_elem[i][1]][0];
		double z0 = grid[num_elem[i][0]][1];
		double z1 = grid[num_elem[i][2]][1];
		double hr = r1 - r0;
		double hz = z1 - z0;

		if (r >= r0 && r <= r1 && z >= z0 && z <= z1)
		{// вычисление соответствующих базисных функций

			vector<double> psi(4);
			psi[0] = (r1 - r) / hr * (z1 - z) / hz;
			psi[1] = (r - r0) / hr * (z1 - z) / hz;
			psi[2] = (r1 - r) / hr * (z - z0) / hz;
			psi[3] = (r - r0) / hr * (z - z0) / hz;

			res = psi[0] * q[num_elem[i][0]] + psi[1] * q[num_elem[i][1]] +
psi[2] * q[num_elem[i][2]] + psi[3] * q[num_elem[i][3]]; // значение V
			return res;
		}
	}
	return res;
}

//коэффициент
```

```
double sigma(int num, vector<vector<double>> grid, vector<vector<int>> num_elem)
{
	for (int i = 0; i < sloy.size(); i++)
	{
		if (grid[num_elem[num][0]][1] <= sloy[i][1] && grid[num_elem[num][2]][1]
<= sloy[i][1])
			if (grid[num_elem[num][0]][1] >= sloy[i][0] &&
grid[num_elem[num][2]][1] >= sloy[i][0])
				return sloy[i][2];
	}
	return 0;
}

double sigma_dop(int num, vector<vector<double>> grid, vector<vector<int>> num_elem)
{
	for (int i = 0; i < sloy_dop.size(); i++)
	{
		if (grid[num_elem[num][0]][1] <= sloy_dop[i][1] &&
grid[num_elem[num][2]][1] <= sloy_dop[i][1])
			if (grid[num_elem[num][0]][1] >= sloy_dop[i][0] &&
grid[num_elem[num][2]][1] >= sloy_dop[i][0])
				return sloy_dop[i][2];
	}
	return 0;
}
// подсчет матрицы жесткости
void GetLocalG(double rp, double zs, double hr, double hz)
{

	double a1 = (hz * rp) / (6 * hr);
	double a2 = (hz) / 12;
	double a3 = (hr * rp) / (6 * hz);
	double a4 = (hr * hr) / (12 * hz);

	G[0][0] = 2 * a1 + 2 * a2 + 2 * a3 + a4;
	G[0][1] = -2 * a1 - 2 * a2 + a3 + a4;
	G[0][2] = a1 + a2 - 2 * a3 - a4;
	G[0][3] = -a1 - a2 - a3 - a4;

	G[1][0] = -2 * a1 - 2 * a2 + a3 + a4;
	G[1][1] = 2 * a1 + 2 * a2 + 2 * a3 + 3 * a4;
	G[1][2] = -a1 - a2 - a3 - a4;
	G[1][3] = a1 + a2 - 2 * a3 - 3 * a4;

	G[2][0] = a1 + a2 - 2 * a3 - a4;
	G[2][1] = -a1 - a2 - a3 - a4;
	G[2][2] = 2 * a1 + 2 * a2 + 2 * a3 + a4;
	G[2][3] = -2 * a1 - 2 * a2 + a3 + a4;

	G[3][0] = -a1 - a2 - a3 - a4;
	G[3][1] = a1 + a2 - 2 * a3 - 3 * a4;
	G[3][2] = -2 * a1 - 2 * a2 + a3 + a4;
	G[3][3] = 2 * a1 + 2 * a2 + 2 * a3 + 3 * a4;
}

// расчет начального шага
double step(double min, double max, int n, double k)
{
	if (k != 1)
		return (max - min) * (1 - k) / (1 - pow(k, n));
	else
		return (max - min) / n;
}

void SigmaGeneration()
```

```
{
	ifstream input("sigma.txt");

	int n_sloy; // кол-во слоев

	input >> n_sloy;

	sloy.resize(n_sloy);

	for (int i = 0; i < n_sloy; i++)
	{
		sloy[i].resize(3);
		// нижняя граница .. верхняя граница .. sigma
		input >> sloy[i][0] >> sloy[i][1] >> sloy[i][2];
	}

	int n_sloy_dop; // кол-во слоев
	input >> n_sloy_dop;
	sloy_dop.resize(n_sloy_dop);

	for (int i = 0; i < n_sloy_dop; i++)
	{
		sloy_dop[i].resize(3);
		// нижняя граница .. верхняя граница .. sigma
		input >> sloy_dop[i][0] >> sloy_dop[i][1] >> sloy_dop[i][2];
	}
}
// сетка
void GridGeneration()
{
	ofstream node("node.txt");
	ofstream elem("elem.txt");

	int pointz_size = 0;
	int pointr_size = 0;

	// чтение r
	input >> pointr_size;
	pointr.resize(pointr_size + 1);
	for (int i = 0; i < pointr.size(); i++)
	{
		input >> pointr[i];
	}
	nr.resize(pointr_size);
	for (int i = 0; i < pointr_size; i++)
	{
		input >> nr[i];
	}
	kr.resize(pointr_size);
	for (int i = 0; i < pointr_size; i++)
	{
		input >> kr[i];
	}
	kr_route.resize(pointr_size);
	for (int i = 0; i < pointr_size; i++)
	{
		input >> kr_route[i];
	}

	// чтение z
	input >> pointz_size;
	pointz.resize(pointz_size + 1);
	for (int i = 0; i < pointz.size(); i++)
	{
		input >> pointz[i];
```

```
	}
	nz.resize(pointz_size);
	for (int i = 0; i < pointz_size; i++)
	{
		input >> nz[i];
	}
	kz.resize(pointz_size);
	for (int i = 0; i < pointz_size; i++)
	{
		input >> kz[i];
	}
	kz_route.resize(pointz_size);
	for (int i = 0; i < pointz_size; i++)
	{
		input >> kz_route[i];
	}

	// построение r
	for (int i = 0; i < pointr.size() - 1; i++)
	{
		// начальный шаг
		double hr0 = step(pointr[i], pointr[i + 1], nr[i], kr[i]);

		if (kr_route[i] == 1) // возрастает шаг
		{
			double r = pointr[i];
			while (pointr[i + 1] - r > 1e-6)
			{
				gridr.push_back(r);
				r += hr0;
				hr0 *= kr[i];
			}
		}
		else // убывает
		{
			double r = pointr[i + 1] - hr0;
			while (r - pointr[i] > 1e-6)
			{
				gridr.push_back(r);
				hr0 *= kr[i];
				r -= hr0;
			}
			gridr.push_back(pointr[i]);
		}
	}
	gridr.push_back(pointr[pointr.size() - 1]); // последнюю точку отдельно

	// построение z
	for (int i = 0; i < pointz.size() - 1; i++)
	{
		// начальный шаг
		double hz0 = step(pointz[i], pointz[i + 1], nz[i], kz[i]);

		if (kz_route[i] == 1) // возрастает шаг
		{
			double z = pointz[i];
			while (pointz[i + 1] - z > 1e-6)
			{
				gridz.push_back(z);
				z += hz0;
				hz0 *= kz[i];
			}
		}
		else // убывает
		{
```

```
                double z = pointz[i + 1] - hz0;
                while (z - pointz[i] > 1e-6)
                {
                        gridz.push_back(z);
                        hz0 *= kz[i];
                        z -= hz0;
                }
                gridz.push_back(pointz[i]);
        }
    }
    gridz.push_back(pointz[pointz.size() - 1]); // последнюю точку отдельно

    sort(gridz.begin(), gridz.end());
    sort(gridr.begin(), gridr.end());
    nv = gridr.size() * gridz.size();
    grid.resize(nv);

    z_min = gridz[0];
    r_max = gridr[gridz.size() - 1];

    node << nv << endl;
    int i = 0;

    for (int iy = 0; iy < gridz.size(); iy++)
        for (int ix = 0; ix < gridr.size(); ix++)
        {
                grid[i].resize(2);
                grid[i][0] = gridr[ix];
                grid[i][1] = gridz[iy];
                node << i << " " << grid[i][0] << " " << grid[i][1] << endl;
                i++;
        }

    nr_full = gridr.size() - 1;
    nz_full = gridz.size() - 1;
    n = nr_full * nz_full; // количество конечных элементов
    num_elem.resize(n);
    int tmp_num = 0;
    for (int j = 0; j < nz_full; j++)
        for (int i = 0; i < nr_full; i++)
        {
                num_elem[tmp_num].resize(4);
                num_elem[tmp_num][0] = i + j * (nr_full + 1); // 1 вершина
                num_elem[tmp_num][1] = i + j * (nr_full + 1) + 1; // 2 вершина
                num_elem[tmp_num][2] = i + (j + 1) * (nr_full + 1); // 3 вершина
                num_elem[tmp_num][3] = i + (j + 1) * (nr_full + 1) + 1; // 4 вер-
шина
                // вывод в файл
                elem << tmp_num << " " << num_elem[tmp_num][0] << " " <<
num_elem[tmp_num][1] << " " << num_elem[tmp_num][2] << " " <<
                        num_elem[tmp_num][3] << endl;
                tmp_num++;
        }
}

// портрет матрицы
void MatrixPortrait(vector<vector<int>> num_elem)
{
    vector<vector<int>> list(nv); // вспомогательный массив по количеству узлов
    vector<int> tmp(4); // вспомогательный вектор вершин одного элемента
    for (int i = 0; i < nv; i++)
        list[i];
    int number = 0; // количество элементов для массива ig
    for (int i = 0; i < num_elem.size(); i++) //идем по конечным элементам
    {
```

```
            for (int j = 0; j < 4; j++)
                  tmp[j] = num_elem[i][j];
            reverse(tmp.begin(), tmp.end()); //сортируем по убыванию, т.к. портрет
симметричный (в нашем случае переворачиваем т.к. все изначально отсортировано)
            for (int j = 0; j < 4; j++)
                  for (int k = j + 1; k < 4; k++)
                  {
                        int flag = 1;
                        for (int p = 0; p < list[tmp[j]].size() && flag; p++)
                              if (list[tmp[j]][p] == tmp[k]) flag = 0; // если эле-
мент уже есть в списке смежности
                        if (flag)
                        {
                              list[tmp[j]].push_back(tmp[k]); // если в списке еще
нет такого элемента, то добавляем
                              number++; // увеличиваем количество для jg
                        }
                  }
      }
      for (int i = 0; i < nv; i++)
            sort(list[i].begin(), list[i].end()); //сортируем по возрастанию

      ig.resize(nv + 1);

      ig[0] = 0;
      ig[1] = 0;
      for (int i = 2; i < nv + 1; i++)
            ig[i] = ig[i - 1] + list[i - 1].size();

      for (int i = 0; i < nv; i++)
            for (int j = 0; j < list[i].size(); j++)
                  jg.push_back(list[i][j]);

      //выделение памяти для работы с векторами
      al.resize(number);
      au.resize(number);
      di.resize(nv);
      b.resize(nv);
      //ЛОС
      L.resize(number);
      U.resize(number);
      D.resize(nv);
      x0.resize(nv);
      y.resize(nv);
      z.resize(nv);
      r.resize(nv);
      p.resize(nv);
      t.resize(nv);
      locala2[0].resize(2);
      locala2[1].resize(2);

      for (int i = 0; i < 4; i++)
      {
            G[i].resize(4);
            M[i].resize(4);
            localA[i].resize(4);
      }
}

// подсчет локальной матрицы
void LocalMatrix(int num, vector<vector<double>> grid, vector<vector<int>> num_elem)
{
      //вершины прямоугольника
      int a = num_elem[num][0];
      int b = num_elem[num][1];
```

```
	int c = num_elem[num][2];
	int d = num_elem[num][3];
	double rp, zs, hr, hz;
	rp = grid[a][0];
	zs = grid[a][1];
	hr = grid[b][0] - grid[a][0];
	hz = grid[c][1] - grid[b][1];

	GetLocalG(rp, zs, hr, hz);

	for (int i = 0; i < 4; i++)
		for (int j = 0; j < 4; j++)
		{
			localA[i][j] = sigma(num, grid, num_elem) * G[i][j];
		}
}

// подсчет локальной матрицы
void LocalMatrix_dop(int num, vector<vector<double>> grid, vector<vector<int>>
num_elem)
{
	for (int i = 0; i < 4; i++)
	{
		localb[i] = 0;
	}

	//вершины прямоугольника
	int a = num_elem[num][0];
	int b = num_elem[num][1];
	int c = num_elem[num][2];
	int d = num_elem[num][3];
	double rp, zs, hr, hz;
	rp = grid[a][0];
	zs = grid[a][1];
	hr = grid[b][0] - grid[a][0];
	hz = grid[c][1] - grid[b][1];

	vector<double> q; // правая часть задачи V_n
	q.push_back(result_q(grid[a][0], grid[a][1], qn, grid_n, num_elem_n));
	q.push_back(result_q(grid[b][0], grid[b][1], qn, grid_n, num_elem_n));
	q.push_back(result_q(grid[c][0], grid[c][1], qn, grid_n, num_elem_n));
	q.push_back(result_q(grid[d][0], grid[d][1], qn, grid_n, num_elem_n));

	GetLocalG(rp, zs, hr, hz);

	for (int i = 0; i < 4; i++)
		for (int j = 0; j < 4; j++)
		{
			localA[i][j] = sigma_dop(num, grid, num_elem) * G[i][j];
		}

	for (int i = 0; i < 4; i++)
		for (int j = 0; j < 4; j++)
		{
			localb[i] += (sigma(num, grid, num_elem) - sigma_dop(num, grid,
num_elem)) * G[i][j] * q[j];
		}
}

// добавление локальной матрицы в глобальную
void AddInGlobal(int num, vector<vector<int>> num_elem)
{
	// диагональ
	for (int i = 0; i < 4; i++)
		di[num_elem[num][i]] += localA[i][i];
```

```
	for (int i = 0; i < 4; i++)
	{
		for (int j = 0; j < i; j++)
		{
			int igg = num_elem[num][i];
			int jgg = num_elem[num][j];
			if (igg < jgg)
			{
				igg = jgg;
				jgg = num_elem[num][i];
			}
			int index = ig[igg];
			int flag = 1;
			for (; index < ig[igg + 1] && flag; index++)
				if (jg[index] == jgg) flag = 0;
			index--;
			al[index] += localA[i][j];
			au[index] += localA[j][i];
		}
	}
}

// добавление локальной матрицы в глобальную
void AddInGlobal_dop(int num, vector<vector<int>> num_elem)
{
	// диагональ
	for (int i = 0; i < 4; i++)
		di[num_elem[num][i]] += localA[i][i];

	for (int i = 0; i < 4; i++)
	{
		for (int j = 0; j < i; j++)
		{
			int igg = num_elem[num][i];
			int jgg = num_elem[num][j];
			if (igg < jgg)
			{
				igg = jgg;
				jgg = num_elem[num][i];
			}
			int index = ig[igg];
			int flag = 1;
			for (; index < ig[igg + 1] && flag; index++)
				if (jg[index] == jgg) flag = 0;
			index--;
			al[index] += localA[i][j];
			au[index] += localA[j][i];
		}
	}

	for (int i = 0; i < 4; i++)
		b[num_elem[num][i]] += localb[i];
}

// учет первых краевых условий
void First(vector<vector<double>> grid)
{
	for (int i = 0; i < grid.size(); i++) {
		// нижняя граница
		if (grid[i][1] == z_min) {
			di[i] = 1e+15;
			b[i] = 0;
		}
		// правая граница
```

```
			if (grid[i][0] == r_max) {
				di[i] = 1e+15;
				b[i] = 0;
			}
		}
}

void LU()
{
		for (int i = 0; i < ig[nv]; i++)
		{
			L[i] = al[i];
			U[i] = au[i];
		}
		for (int i = 0; i < nv; i++)
			D[i] = di[i];

		for (int i = 0; i < nv; i++)
		{
			int ia0 = ig[i];
			int ia1 = ig[i + 1];
			double sd = 0;
			for (int k = ia0; k < ia1; k++)
			{
				double s1 = 0;
				double s2 = 0;
				int j = jg[k]; // номер столбца(строки) элемента al(au)
				int ja0 = ig[j]; // номер, с которого начинается элементы
столбца(строки)
				int ja1 = ig[j + 1]; // номер, с которого начинаются элементы но-
вого столбца(строки)
				int ki = ia0;
				int kj = ja0;
				while (ki < k && kj < ja1)
					if (jg[ki] == jg[kj])
					{
						s1 += L[ki] * U[kj];
						s2 += L[kj] * U[ki];
						ki++;
						kj++;
					}
					else if (jg[ki] > jg[kj])
						kj++;
					else
						ki++;
				L[k] = L[k] - s1;
				U[k] = U[k] - s2;
				U[k] = U[k] / D[jg[k]];
				sd += U[k] * L[k];
			}
			D[i] -= sd;
		}
}

void LSolve(vector<double>& x1, vector<double> x2)
{
		x1[0] = x2[0] / D[0];
		for (int i = 1; i < nv; i++)
		{
			double s = 0;
			for (int j = ig[i]; j < ig[i + 1]; j++)
				s += L[j] * x1[jg[j]];
			x1[i] = x2[i] - s;
			x1[i] = x1[i] / D[i];
		}
```

```
}

void USolve(vector<double>& x1, vector<double> x2)
{
	for (int i = 0; i < nv; i++)
		x1[i] = x2[i];
	x1[nv - 1] = x1[nv - 1] / D[nv - 1];
	for (int i = nv - 1; i >= 1; i--)
	{
		for (int j = ig[i]; j < ig[i + 1]; j++)
			x1[jg[j]] -= U[j] * x1[i];
	}
}

// умножения вектора на матрицу A
void multiplication_matrix_on_vector(vector<double> a, vector<double>& b)
{
	for (int i = 0; i < nv; i++)
		b[i] = 0;
	for (int i = 0; i < nv; i++)
	{
		for (int j = ig[i]; j < ig[i + 1]; j++)
		{
			b[i] += al[j] * a[jg[j]];
			b[jg[j]] += au[j] * a[i];
		}
		b[i] += di[i] * a[i];
	}
}

// умножение двух векторов
double vectors_multiplication(vector<double> v1, vector<double> v2)
{
	double s = 0;
	for (int i = 0; i < nv; i++)
		s += v1[i] * v2[i];
	return s;
}

// норма вектора
double norma(vector<double> vector)
{
	double s = 0;
	for (int i = 0; i < nv; i++)
		s += vector[i] * vector[i];
	return(sqrt(s));
}
// d = a + b * c
void summ(vector<double> a, double b, vector<double> c, vector<double>& d)
{
	for (int i = 0; i < nv; i++)
		d[i] = a[i] + b * c[i];
}

void LOS()
{
	double alpha, betta;
	double pk_1_rk_1, pk_1_pk_1;
	// нулевое начальное приближение
	for (int i = 0; i < nv; i++)
		x0[i] = 0;
	multiplication_matrix_on_vector(x0, y); // Ax0
	summ(b, -1, y, r); // r0 = f - Ax0
	LSolve(r, r); // r0 = (f - Ax0) / L
	USolve(z, r); // z0 = r0 / U
```

```
	multiplication_matrix_on_vector(z, y); // Az0
	LSolve(p, y); // p0 = Az0 / L
	for (int k = 1; k < maxiter; k++)
	{
		pk_1_rk_1 = vectors_multiplication(p, r); // (p_(k-1),r_(k-1))
		pk_1_pk_1 = vectors_multiplication(p, p); // (p_(k-1),p_(k-1))
		alpha = pk_1_rk_1 / pk_1_pk_1; // alpha_k = (p_(k-1),r_(k-1)) / (p_(k-
1),p_(k-1))
		summ(x0, alpha, z, x0); // x_k = x_(k-1) + alpha_k * z_(k-1)
		summ(r, -alpha, p, r); // r_k = r_(k-1) - alpha_k * p_(k-1)

		USolve(t, r); // t = r_k/ U
		multiplication_matrix_on_vector(t, y); // y = A * r_k / U
		LSolve(t, y); // t = A * r_k / UL
		betta = -vectors_multiplication(p, t) / pk_1_pk_1; // betta_k = (p_k-
1,L_1*A*U_1*r_k) / (p_(k-1),p_(k-1))
		USolve(y, r); // y = r_k / U
		summ(y, betta, z, z);//  z_k = r_k / U + betta_k * z_(k-1)
		summ(t, betta, p, p);//p_k = L_1*A*U_1* r_k + betta_k * p_(k-1)

		if (vectors_multiplication(r, r) < eps) // (r_k, r_k) < e
		{
			return;
		}
	}
}

// прямая задача
vector<double> direct_task()
{
	for (int i = 0; i < num_elem_n.size(); i++)
	{
		LocalMatrix(i, grid_n, num_elem_n);
		AddInGlobal(i, num_elem_n);
	}
	//b[0] = 1;
	First(grid_n);
	for (int i = 0; i < grid_n.size(); i++) {
		if (grid_n[i][0] == 0 && grid_n[i][1] == 0) {
			b[i] = tok;
		}
	}
	LU();
	LOS();
	// меняем значения для нового слоя
	for (int i = 0; i < grid_n.size(); i++)
	{
		di[i] = 0;
		b[i] = 0;
	}
	for (int i = 0; i < al.size(); i++)
	{
		al[i] = 0;
		au[i] = 0;
	}
	ofstream q("qn.txt");

	for (int i = 0; i < grid_n.size(); i++)
		q << x0[i] << endl;
	return x0;
}
// прямая задача
vector<double> direct_task_dop()
{
	for (int i = 0; i < num_elem_dop.size(); i++)
```

```
	{
		LocalMatrix_dop(i, grid_dop, num_elem_dop);
		AddInGlobal_dop(i, num_elem_dop);
	}
	First(grid_dop);
	LU();
	LOS();
	// меняем значения для нового слоя
	for (int i = 0; i < grid_dop.size(); i++)
	{
		di[i] = 0;
		b[i] = 0;
	}
	for (int i = 0; i < al.size(); i++)
	{
		al[i] = 0;
		au[i] = 0;
	}

	ofstream q("q+.txt");

	for (int i = 0; i < grid_dop.size(); i++)
		q << x0[i] << endl;

	return x0;
}

double xA = 0;
double yA = 0;
double xB = 0;
double yB = 100;

double result_xyz_q(double x, double y, vector<double> q, vector<vector<double>> grid,
vector<vector<int>> num_elem) {
	double r1 = sqrt((x - xA) * (x - xA) + (y - yA) * (y - yA));
	double r2 = sqrt((x - xB) * (x - xB) + (y - yB) * (y - yB));
	return result_q(r1, 0, q, grid, num_elem) - result_q(r2, 0, q, grid, num_elem);
}

void result_function_q(vector<double> q, vector<vector<double>> grid, vector<vec-
tor<int>> num_elem) {
	cout << result_xyz_q(100, 0, q, grid, num_elem) << endl;
	cout << result_xyz_q(150, 0, q, grid, num_elem) << endl;
	cout << result_xyz_q(200, 0, q, grid, num_elem) << endl;
	cout << result_xyz_q(250, 0, q, grid, num_elem) << endl;
	cout << result_xyz_q(300, 0, q, grid, num_elem) << endl;

}

void result_function_q(vector<double>& vec, vector<double> q, vector<vector<double>>
grid, vector<vector<int>> num_elem) {
	vec[0] = result_xyz_q(100, 0, q, grid, num_elem);
	vec[1] = result_xyz_q(150, 0, q, grid, num_elem);
	vec[2] = result_xyz_q(200, 0, q, grid, num_elem);
	vec[3] = result_xyz_q(250, 0, q, grid, num_elem);
	vec[4] = result_xyz_q(300, 0, q, grid, num_elem);
}

double derivative(double a, double b) {
	return (b - a) / (0.05 * tmp_u);
}

void clearAllVectors()
{
	// Основные матрицы и векторы
```

```
	G.clear();
	G.resize(4);

	M.clear();
	M.resize(4);

	localA.clear();
	localA.resize(4);

	localb.clear();
	localb.resize(4);

	grid.clear();
	num_elem.clear();
	edge1.clear();
	edge2.clear();

	localb2.clear();
	localb2.resize(2);

	locala2.clear();
	locala2.resize(2);

	// Векторы для СЛАУ
	ig.clear();
	jg.clear();
	al.clear();
	au.clear();
	di.clear();
	b.clear();

	// Векторы для ЛОС
	L.clear();
	U.clear();
	D.clear();
	x0.clear();
	z.clear();
	r.clear();
	y.clear();
	p.clear();
	t.clear();

	// Векторы для сетки
	gridr.clear();
	gridz.clear();
	pointz.clear();
	nz.clear();
	kz.clear();
	kz_route.clear();
	pointr.clear();
	nr.clear();
	kr.clear();
	kr_route.clear();
}

void field_selection()
{
	ofstream output("q.txt");
	SigmaGeneration();
	GridGeneration();
	// сохранение сетки
	for (int i = 0; i < grid.size(); i++) {

		grid_n.push_back(grid[i]);
	}
```

```
	for (int i = 0; i < num_elem.size(); i++) {

		num_elem_n.push_back(num_elem[i]);
	}
	MatrixPortrait(num_elem_n);
	qn = direct_task(); // прямая задача
	// обнуление параметров
	clearAllVectors();
	GridGeneration();
	// сохранение сетки
	for (int i = 0; i < grid.size(); i++) {

		grid_dop.push_back(grid[i]);
	}
	for (int i = 0; i < num_elem.size(); i++) {

		num_elem_dop.push_back(num_elem[i]);
	}
	MatrixPortrait(num_elem_dop);
	q_dop = direct_task_dop(); // прямая задача дополнительная
	qv.resize(qn.size());

	// итоговый результат
	for (int i = 0; i < grid_n.size(); i++)
	{
		qv[i] = result_q(grid_n[i][0], grid_n[i][1], qn, grid_n, num_elem_n) +
			result_q(grid_n[i][0], grid_n[i][1], q_dop, grid_dop,
num_elem_dop);
		//cout << qv[i] << endl;
	}

}
void field_selection_direct_task()
{
	clearAllVectors();
	nv = grid_n.size();
	MatrixPortrait(num_elem_n);
	qn = direct_task(); // прямая задача
	// обнуление параметров
	clearAllVectors();
	nv = grid_dop.size();
	MatrixPortrait(num_elem_dop);
	q_dop = direct_task_dop(); // прямая задача дополнительная
	// итоговый результат
	for (int i = 0; i < grid_n.size(); i++)
	{
		qv[i] = result_q(grid_n[i][0], grid_n[i][1], qn, grid_n, num_elem_n) +
			result_q(grid_n[i][0], grid_n[i][1], q_dop, grid_dop,
num_elem_dop);
	}
}

void inverse_problem() {
	result_function_q(true_eps, qv, grid_n, num_elem_n);
	//true_eps[0] *= 1.05; // зашумление
	tmp_u = start_u;
	for (int iter = 0; iter < max_iter; iter++) {
		double sum = 0; // левая часть уравнения
		double f = 0; // правая часть уравнения
		//tok = tmp_u;
		sloy_dop[1][2] = tmp_u;
		// приращение
		field_selection_direct_task();
		result_function_q(tmp_eps, qv, grid_n, num_elem_n);
		//tok = 1.05 * tmp_u;
```

```
			sloy_dop[1][2] = 1.05 * tmp_u;
			field_selection_direct_task();
			result_function_q(delta_tmp_eps, qv, grid_n, num_elem_n);
			// слау
			for (int i = 0; i < 5; i++) {
				sum += w * w * pow(derivative(delta_tmp_eps[i], tmp_eps[i]), 2);
			}
			sum += alpha;
			for (int i = 0; i < 5; i++) {
				f -= w * w * derivative(delta_tmp_eps[i], tmp_eps[i]) *
(true_eps[i] - tmp_eps[i]);
			}
			f -= alpha * (tmp_u - start_u);

			delta_u = f / sum;
			// итерация
			double J_prev = 0;
			double J_next = 0;
			for (int j = 0; j < 5; j++) {
				J_prev = 0;
				for (int i = 0; i < 5; i++) {
					//w = 1 / tmp_eps[i];
					J_prev += pow(w * (true_eps[i] - tmp_eps[i]), 2);
				}
				J_next = 0;
				//tok = delta_u * betta + tmp_u;
				sloy_dop[1][2] = delta_u * betta + tmp_u;
				field_selection_direct_task();
				result_function_q(next_eps, qv, grid_n, num_elem_n);
				for (int i = 0; i < 5; i++) {
					//w = 1 / next_eps[i];
					J_next += pow(w * (true_eps[i] - next_eps[i]), 2);
				}
				if (J_next - J_prev > 0) {
					betta /= 2;
				}
				else {
					break;
				}
			}
			betta = 1;
			//cout << "iter = " << iter + 1 << setprecision(15) << " tok = " << tok
<< endl;
			cout << "iter = " << iter + 1 << setprecision(15) << " sigma = " <<
sloy_dop[1][2] << endl;
			if (abs(J_prev - J_next) < eps) {
				break;
			}
			else {
				//tmp_u = tok;
				tmp_u = sloy_dop[1][2];
			}
		}
}

int main()
{
	ofstream output("q.txt");
	field_selection();
	inverse_problem();

}
```