Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования "Уфимский университет науки и технологий"

Кафедра Высокопроизводительных вычислительных технологий и систем

Дисциплина: Теория разностных схем

Отчет по лабораторной работе № 3

Тема: «Решение краевых задач для эллиптических уравнений»

Группа МКН-316	Фамилия И.О.	Подпись	Дата	Оценка
Студент	Султанов М.Ф.			
Принял	Белевцов Н.С.			

Уфа 2023

Цель: получить навык численного решения краевых задач для уравнений эллиптического типа с использованием различных методов на примере задачи Дирихле для линейного двумерного неоднородного уравнения.

Практическая часть

Рассматривается задача Дирихле для линейного двумерного неоднородного эллиптического уравнения с переменными коэффициентами:

$$\frac{\partial}{\partial x}\left(a(x,y)\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(b(x,y)\frac{\partial u}{\partial y}\right)+c(x,y)u+f(x,y)=0,(x,y)\in\Omega=(0,l_x)\times(0,l_y); \quad (1)$$

$$u_\Gamma=\varphi(x,y),(x,y)\in\Gamma=\partial\Omega. \quad (2)$$

I. Задача Дирихле для уравнения Пуассона с постоянными коэффициентами

Рассматривается частный случай уравнения (1) – уравнение Пуассона с постоянными коэффициентами:

$$a(x,y)=b(x,y)=1,c(x,y)=0. \quad (3)$$

В соответствии с индивидуальным заданием рассматриваем область с $l_x=1,l_y=2$. Задано точное решение:

$$u(x,y)=\sin\left(\frac{\pi x^2}{l_x^2}\right)\sin\left(\frac{\pi y}{l_y}\right)$$

Тогда задача обретает вид:

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=-f(x,y),(x,y)\in\Omega=(0,1)\times(0,2);$$

$$u_\Gamma=0,(x,y)\in\Gamma=\partial\Omega.$$

$$f(x,y)=\frac{\pi^2}{4}(1+16x^2)\sin(\pi x^2)\sin\left(\frac{\pi y}{2}\right)-2\pi\cos(\pi x^2)\sin\left(\frac{\pi y}{2}\right)$$

Задача 1.

1) Написать вычислительную программу на языке программирования C++ решения задачи (1)-(3) с использованием конечно-разностной схемы с шаблоном «крест» на сетке с постоянными шагами h_x и h_yпо направлениям *x* и *y*, удовлетворяющих соотношению

$$\frac{h_x}{h_y}=\frac{l_x}{l_y}.$$

Для решения получающейся СЛАУ использовать метод простых итераций. При этом матрица системы не должна храниться в памяти.

2) Исследовать зависимость погрешности решения от величины шагов сетки и построить соответствующие графики. Погрешность оценивать в равномерной норме. Исследовать зависимости числа итераций от шага сетки

Решение.

Сравним результаты с точным решением. Погрешность продемонстрирована на рисунке 1. Число итераций продемонстрирована на рисунке 2.

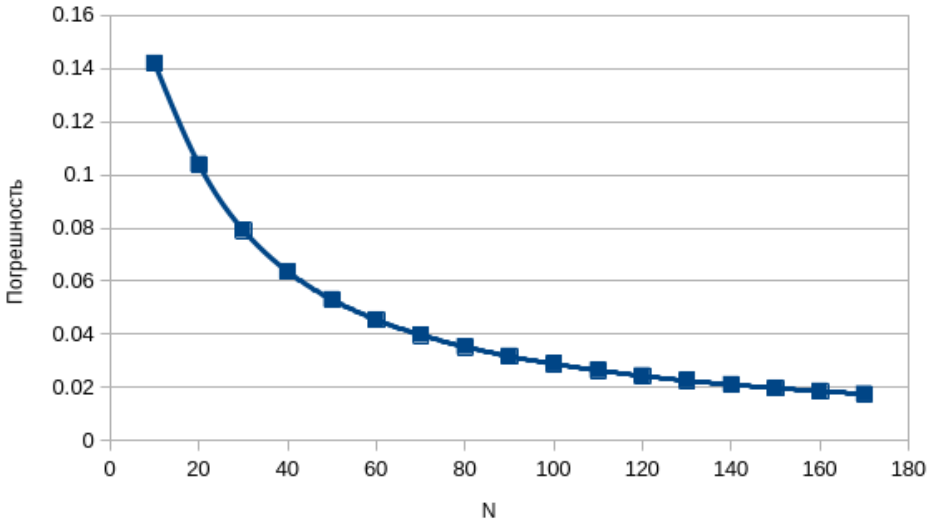

Рисунок 1: Погрешность МПИ

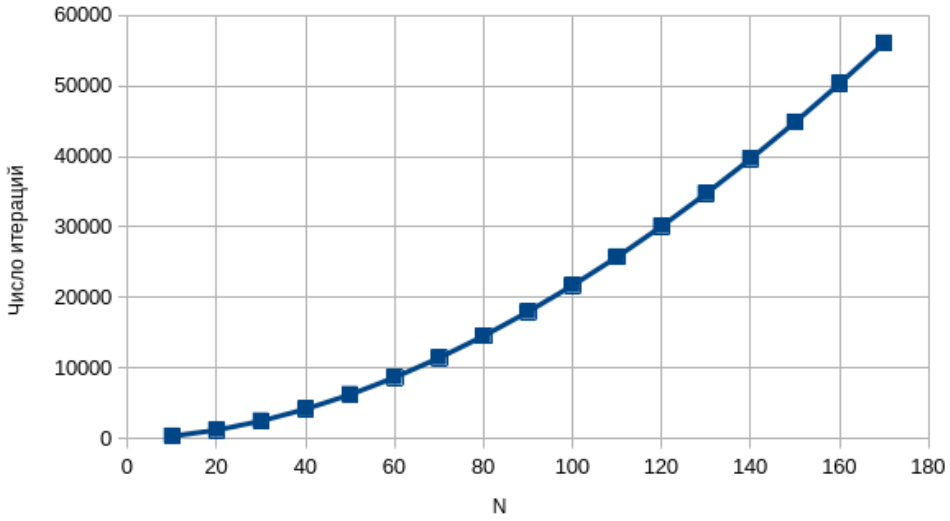

Рисунок 2: Число итераций МПИ

Задача 2

Решить задачу 1 с использованием для решения СЛАУ метод SOR. Параметр релаксации либо выбирается фиксированным, либо используется формула для оптимального значения.

Решение.

Будем использовать параметр релаксации равный $\sigma=1$, то есть будем использловать метод Гаусса-Зейделя. Сравним результаты с пточным решением и МПИ. График погрешности приведен на рисунке 3. Число итераций продемонстрирован на рисунке 4.

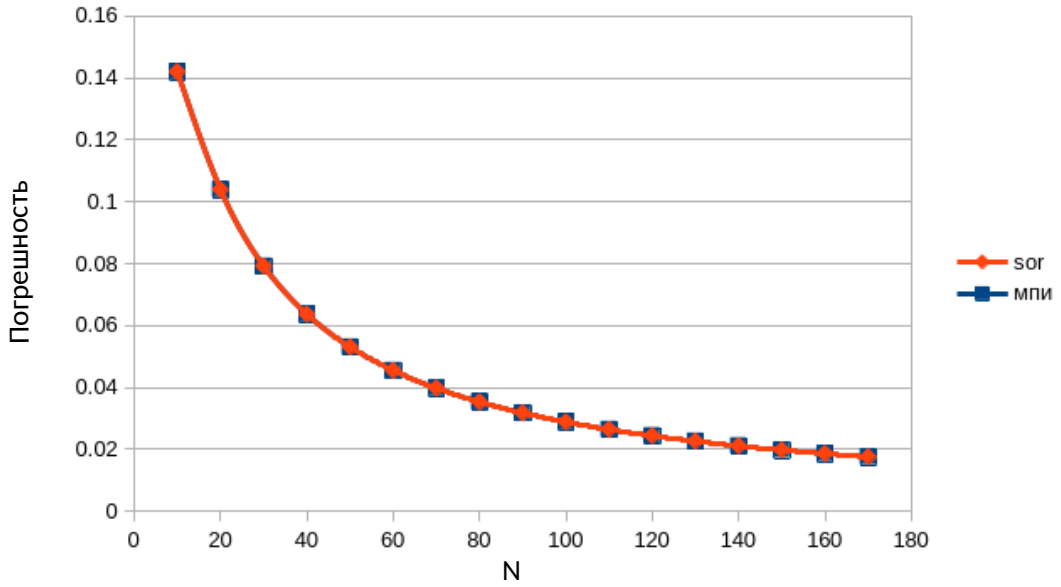

Рисунок 3: Сравнение погрешности SOR и МПИ

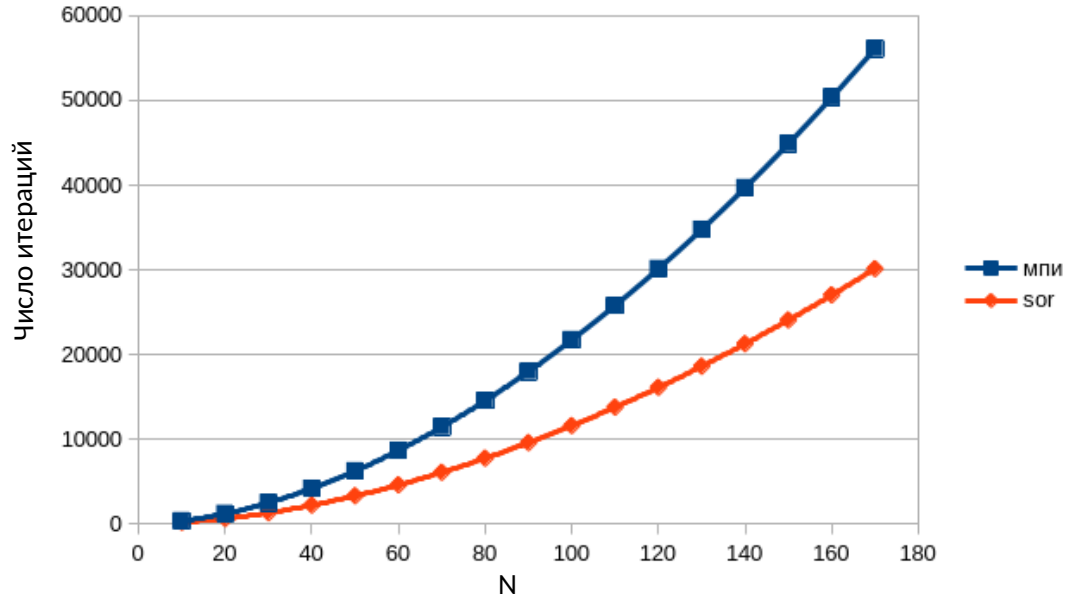

Рисунок 4: Сравнение числа итераций SOR и МПИ

Задача 3.

Решить задачу 1 с использованием для решения СЛАУ любой точный метод (Гаусса, LU-разложение, метод сопряженных градиентов с большим числом итераций). В данной задаче матрицу системы можно хранить целиком в памяти, желательно только ненулевые диагонали.

Решение

Решим задачу с помощью LU разложения. При этом для решения системы будем использовать прямой и обратный ход метода Гаусса. Результаты прогрешности приведены на Рисунке 5. Отметим, что расчет занял гораздо больше времени, чем точные методы.

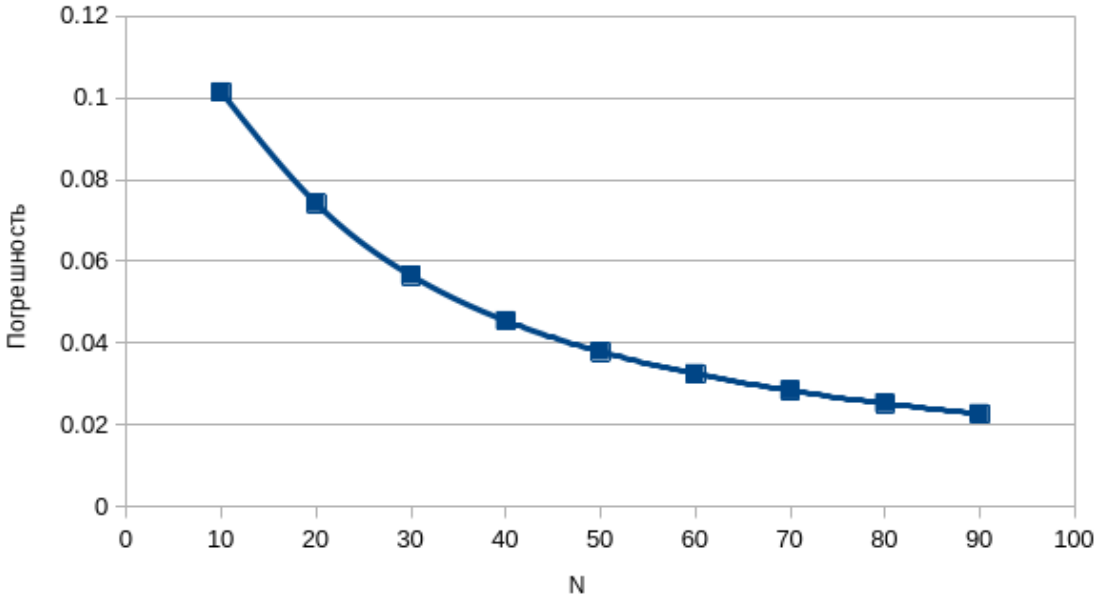

Рисунок 5: Погрешность метода Гаусса с использованием LU разложения

II. Решение задачи с переменными коэффициентами

Задача 4

1) Написать вычислительную программу на языке программирования C++ решения задачи (1)-(2) с параметрами из таблиц 1 и 2 методом переменных направлений, либо использовать другой достаточно метод решения СЛАУ (точный метод или метод сопряженных градиентов).
2) Исследовать зависимость погрешности получаемого решения от величины шага сетки, построить соответствующие графики.

В соответствии с индивидуальным варинатом рассмотрим уравнение Пуассона с переменными коэффициентами:

$$a(x,y)=1, b(x,y)=1+y^2, c(x,y)=x^2-y^2.$$

В таблице 1 приведены графики решений методом сопряженных градиентов. На рисунке 6 приведены погрешности решения от размера сетки.

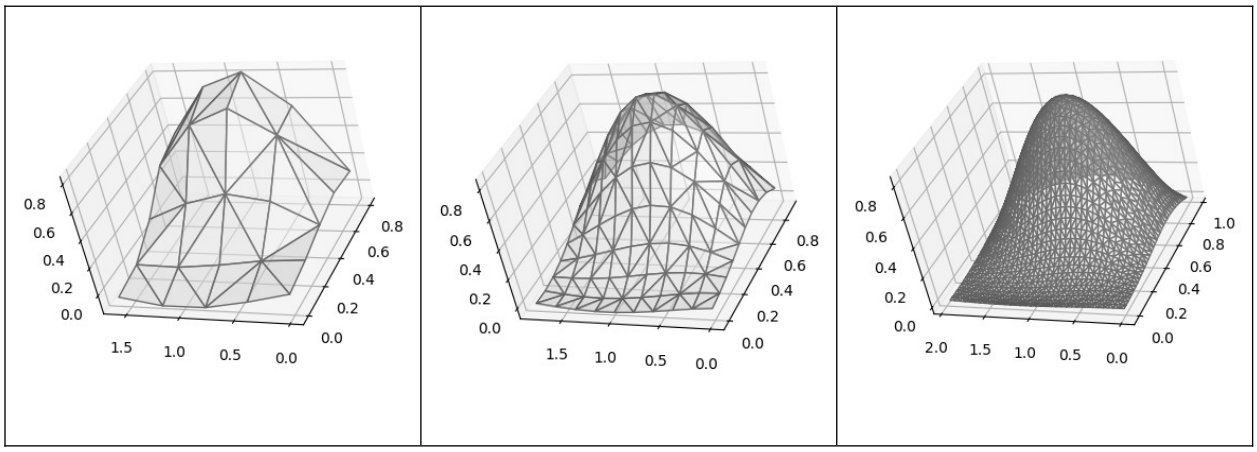

Таблица 1: Решение уравнения Пуассона с переменными коэффциентами в зависимости от размера сетки при N = 25, N = 100, N = 900

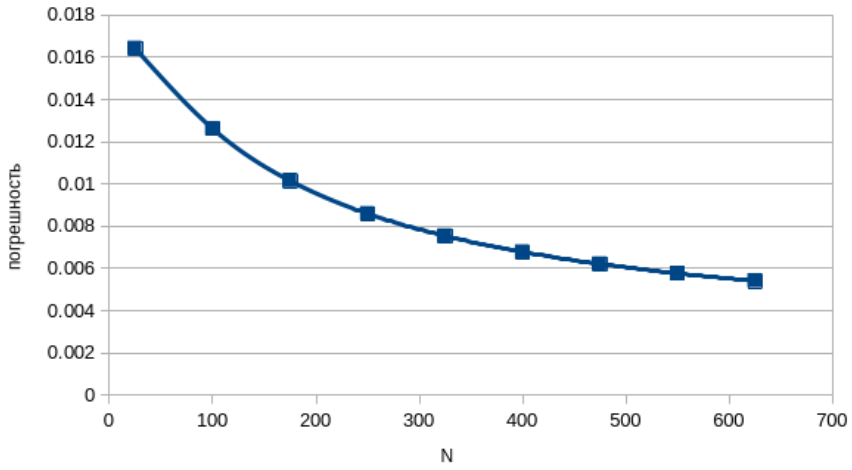

Рисунок 6: Зависимость погрешности решения уравнения Пуассона с нелинейными коэффициентами от размера сетки

Вывод

В ходе лабораторной работы был получен навык численного решения линейных и нелинейных начально-краевых задач для уравнений эллиптического типа. Погрешности итерационных методов при одинаковых параметрах оказались схожими, однако скорость сходимости различается: метод SOR сходится быстрее, а точные методы решения СЛАУ хоть и дают более точный результат, но время работы алгритмы заметно дольше.

Приложение

```
#define _USE_MATH_DEFINES
#include <iostream>
#include <cmath>
#include <vector>
#include <span>
using namespace std;

const double delta = 1e-8;

struct EllipticPDEMatrixGen
{
	double alpha;
	double beta;
	double gamma;

	int N;
	int M;
	EllipticPDEMatrixGen(double alpha, double beta, double gamma, int N, int
M) : alpha(alpha), beta(beta), gamma(gamma), N(N), M(M) {}

	const double &operator()(int i, int j) const
	{
		if (i == j)
		{
			return alpha;
		}
		else if (i - j == -1 && (i + 1) % M != 0)
		{
			return beta;
		}
		else if (i - j == 1 && (i % M != 0))
		{
			return beta;
		}
		else if (abs(i - j) == M)
		{
			return gamma;
		}
		else
		{
			return 0.;
```

```
		}
	}
};

void lu_decompostion(const unsigned int n, EllipticPDEMatrixGen A,
vector<double> &L, vector<double> &U)
{
	for (int i = 0; i < n; ++i)
	{
		L[i * n] = A(i, 0);
		U[i] = A(i, 0) / L[0];
	}

	for (int i = 1; i < n; i++)
	{
		for (int j = i; j < n; j++)
		{
			U[i * n + j] = A(i, j);

			for (int k = 0; k < i; k++)
			{
				U[i * n + j] -= L[i * n + k] * U[k * n + j];
			}

			L[j * n + i] = A(i, j);

			for (int k = 0; k < i; k++)
			{
				L[j * n + i] -= L[j * n + k] * U[k * n + i];
			}

			L[j * n + i] /= U[i * n + i];
		}
	}
	// for (int i = 0; i < n; ++i) {
	//	L[i * n] = A[i * n];
	//	U[i] = A[i] / L[0];
	// }

	// for (int i = 1; i < n; i++)
	//{
	//	for (int j = i; j < n; j++)
	//	{
	//		U[i * n + j] = A[i * n + j];
```

```
	//		for (int k = 0; k < i; k++)
	//		{
	//			U[i * n + j] -= L[i * n + k] * U[k * n + j];
	//		}

	//		L[j * n + i] = A[j * n + i];

	//		for (int k = 0; k < i; k++)
	//		{
	//			L[j * n + i] -= L[j * n + k] * U[k * n + i];
	//		}

	//		L[j * n + i] /= U[i * n + i];
	//	}
	//}
}

void backward_up(const unsigned int n, vector<double> U, vector<double> b, vector<double> &x)
{
	for (int i = n - 1; i >= 0; --i)
	{
		x[i] = b[i];
		for (int j = i + 1; j < n; ++j)
		{
			x[i] -= U[j + i * n] * x[j];
		}
		x[i] /= U[i + i * n];
	}
}

void backward_low(const unsigned int n, vector<double> L, vector<double> b, vector<double> &y)
{
	for (int i = 0; i < n; ++i)
	{
		y[i] = b[i];
		for (int j = 0; j < i; ++j)
		{
			y[i] -= L[i * n + j] * y[j];
		}
		y[i] /= L[i + i * n];
	}
}
```

```
void solve_lu(const unsigned int n, vector<double> L, vector<double> U,
vector<double> b, vector<double> &x)
{
	vector<double> y(n);
	backward_low(n, L, b, y);
	backward_up(n, U, y, x);
}

vector<double> Ax(double alpha, double beta, double gamma, vector<double>
U_prev, double N, double M)
{
	vector<double> U_mult(N * M, 0.);

	// умножаем левые гаммы
	for (int j = 1; j < N; ++j)
	{
		for (int i = 0; i < M; ++i)
		{
			U_mult[j * M + i] += -gamma * U_prev[(j - 1) * M + i] / alpha;
		}
	}
	// умножаем правые гаммы
	for (int j = 0; j < N - 1; ++j)
	{
		for (int i = 0; i < M; ++i)
		{
			U_mult[j * M + i] += -gamma * U_prev[(j + 1) * M + i] / alpha;
		}
	}

	// умножаем левые беты
	for (int j = 0; j < N; ++j)
	{
		for (int i = 1; i < M; ++i)
		{
			U_mult[j * M + i] += -beta * U_prev[j * M + (i - 1)] / alpha;
		}
	}

	// умножаем правые беты
	for (int j = 0; j < N; ++j)
	{
		for (int i = 0; i < M - 1; ++i)
		{
			U_mult[j * M + i] += -beta * U_prev[j * M + (i + 1)] / alpha;
```

```
            }
        }

        return U_mult;
}

int main()
{
        // Метод простых итераций с правильным параметром tau
https://zaochnik.com/spravochnik/matematika/issledovanie-slau/iteratsionnye-metody-reshenija-slau/
        vector<int> N_arr = {5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100};

        {
                // Part I
                const double lx = 1.;
                const double ly = 2.;

                auto f = [](double x, double y)
                {
                        return -0.2e1 * 0.3141592654e1 * cos(0.3141592654e1 * x * x) * sin(0.3141592654e1 * y / 0.2e1) + 0.4e1 * 0.3141592654e1 * 0.3141592654e1 * x * x * sin(0.3141592654e1 * x * x) * sin(0.3141592654e1 * y / 0.2e1) + sin(0.3141592654e1 * x * x) * 0.3141592654e1 * 0.3141592654e1 * sin(0.3141592654e1 * y / 0.2e1) / 0.4e1;
                        ;
                };

                auto phi1 = [](double x = 0, double y = 0)
                {
                        return 0;
                };

                auto phi2 = [](double x = 0, double y = 0)
                {
                        return 0;
                };

                auto phi3 = [](double x = 0, double y = 0)
                {
                        return 0;
                };

                auto phi4 = [](double x = 0, double y = 0)
```

```
		{
			return 0;
		};

		auto exact_sol = [lx, ly](double x, double y)
		{
			return sin(M_PI * x * x / lx / lx) * sin(M_PI * y / ly);
		};

		auto getF = [phi1, phi2, phi3, phi4, f](int N, int M, double hx, double
hy)
		{
			vector<double> F(N * M);
			{
				int i, j;
				// F для левой границы
				i = 0;
				j = 0;
				F[j * M + i] = -f(i * hx, j * hy) - phi1() / hx / hx - phi3() /
hy / hy;
				for (j = 1; j < N - 1; ++j)
				{
					F[j * M + i] = -f(i * hx, j * hy) - phi1() / hx / hx;
				}
				F[j * M + i] = -f(i * hx, j * hy) - phi1() / hx / hx - phi4() /
hy / hy;

				// F для верхней границы
				for (i = 1; i < M - 1; ++i)
				{
					F[j * M + i] = -f(i * hx, j * hy) - phi4() / hy / hy;
				}

				F[j * M + i] = -f(i * hx, j * hy) - phi2() / hx / hx - phi4() /
hy / hy;

				// F для правой границы
				for (j = N - 2; j > 0; --j)
				{
					F[j * M + i] = -f(i * hx, j * hy) - phi2() / hx / hx;
				}
				F[j * M + i] = -f(i * hx, j * hy) - phi2() / hx / hx - phi3() /
hy / hy;

				// F  для нижней границы
```

```
                for (i = M - 2; i > 0; --i)
                {
                        F[j * M + i] = -f(i * hx, j * hy) - phi3() / hy / hy;
                }

                // внутренняя область
                for (j = 1; j < N - 1; ++j)
                {
                        for (i = 1; i < M - 1; ++i)
                        {
                                F[j * M + i] = -f(i * hx, j * hy);
                        }
                }
        }
        return F;
};

// Метод простых итераций
for (auto N : N_arr)
{ // N число по y
        double hy = ly / N;
        double hx = hy * lx / ly;

        int M = long long(lx / hx); // M число по x

        vector<double> U(N * M, 0.); // Начальное приближение
нулевой вектор
        vector<double> U_prev(N * M);

        double alpha = -2. * (1 / hx / hx + 1 / hy / hy);
        double beta = 1 / hx / hx;
        double gamma = 1 / hy / hy;

        vector<double> F = getF(N, M, hx, hy);

        // Метод простых итераций u^(k+1) = b + A * x^(k)
https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_
%D0%B8%D1%82%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8

        vector<double> b(N * M);

        for (int k = 0; k < N * M; ++k)
        {
                b[k] = F[k] / alpha;
        }
```

```
auto Ax = [alpha, beta, gamma, N, M](vector<double> U_prev)
{
	vector<double> U_mult(N * M, 0.);

	// умножаем левые гаммы
	for (int j = 1; j < N; ++j)
	{
		for (int i = 0; i < M; ++i)
		{
			U_mult[j * M + i] += -gamma * U_prev[(j - 1) * M + i] / alpha;
		}
	}
	// умножаем правые гаммы
	for (int j = 0; j < N - 1; ++j)
	{
		for (int i = 0; i < M; ++i)
		{
			U_mult[j * M + i] += -gamma * U_prev[(j + 1) * M + i] / alpha;
		}
	}

	// умножаем левые беты
	for (int j = 0; j < N; ++j)
	{
		for (int i = 1; i < M; ++i)
		{
			U_mult[j * M + i] += -beta * U_prev[j * M + (i - 1)] / alpha;
		}
	}

	// умножаем правые беты
	for (int j = 0; j < N; ++j)
	{
		for (int i = 0; i < M - 1; ++i)
		{
			U_mult[j * M + i] += -beta * U_prev[j * M + (i + 1)] / alpha;
		}
	}

	return U_mult;
```

```
		};

		double error = 0.;
		long long iter_count = 0;
		do
		{
			iter_count++;
			copy(U.begin(), U.end(), U_prev.begin()); // копируем предыдущее значение вектора

			auto ax = Ax(U_prev);

			for (int j = 0; j < N; ++j)
			{
				for (int i = 0; i < M; ++i)
				{
					U[j * M + i] = b[j * M + i] + ax[j * M + i];
				}
			}

			error = 0.;
			for (int k = 0; k < N * M; ++k)
			{
				error = max(abs(U_prev[k] - U[k]), error);
			}

		} while (error > delta);

		error = 0.;

		for (int j = 0; j < N; ++j)
		{
			for (int i = 0; i < M; ++i)
			{
				double x = i * hx;
				double y = j * hy;
				error = max(error, abs(U[j * M + i] - sin(M_PI * x * x / lx / lx) * sin(M_PI * y / ly)));
			}
		}

		cout << "Simple iterations N = " << N << " M  = " << M << " "
			 << " error: " << error << " iter count: " << iter_count << endl;
	}
```

```
		// Метод SOR
		for (auto N : N_arr)
		{
			double hy = ly / N;
			double hx = hy * lx / ly;

			int M = int(lx / hx); // M число по x

			vector<double> U(N * M, 0.); // Начальное приближение нулевой вектор
			vector<double> U_prev(N * M);

			double alpha = -2. * (1 / hx / hx + 1 / hy / hy);
			double beta = 1 / hx / hx;
			double gamma = 1 / hy / hy;

			double omega = 1.2;

			vector<double> F = getF(N, M, hx, hy);

			double error = 0.;
			long long iter_count = 0;
			do
			{
				iter_count++;
				copy(U.begin(), U.end(), U_prev.begin()); // копируем предыдущее значение вектора
				int i, j;

				// Первый блок

				i = 0;
				j = 0;

				U[j * M + i] = (1 - omega) * U_prev[j * M + i] + omega / alpha * (F[j * M + i] - beta * U_prev[j * M + i + 1] - gamma * U_prev[(j + 1) * M + i]);

				for (i = 1; i < M - 1; ++i)
				{
					U[j * M + i] = (1 - omega) * U_prev[j * M + i] +
							omega / alpha * (F[j * M + i] - beta * U[j * M + (i - 1)] - beta * U_prev[j * M + i + 1] - gamma * U_prev[(j + 1) * M + i]);
```

```
                }

                U[j * M + i] = (1 - omega) * U_prev[j * M + i] +
                                omega / alpha * (F[j * M + i] - beta *
U[j * M + (i - 1)] - gamma * U_prev[(j + 1) * M + i]);

                // Внутренние блоки
                for (j = 1; j < N - 1; ++j)
                {
                        i = 0;
                        U[j * M + i] = (1 - omega) * U_prev[j * M + i] +
                                        omega / alpha * (F[j * M + i] -
gamma * U[(j - 1) * M + i] - beta * U_prev[j * M + i + 1] - gamma * U_prev[(j +
1) * M + i]);

                        for (i = 1; i < M - 1; ++i)
                        {
                                U[j * M + i] = (1 - omega) * U_prev[j * M +
i] +
                                                omega / alpha * (F[j *
M + i] - gamma * U[(j - 1) * M + i] - beta * U[j * M + (i - 1)] - beta * U_prev[j *
M + i + 1] - gamma * U_prev[(j + 1) * M + i]);
                        }

                        U[j * M + i] = (1 - omega) * U_prev[j * M + i] +
                                        omega / alpha * (F[j * M + i] -
gamma * U[(j - 1) * M + i] - beta * U[j * M + (i - 1)] - gamma * U_prev[(j + 1) *
M + i]);
                }

                // Последний блок
                i = 0;
                U[j * M + i] = (1 - omega) * U_prev[j * M + i] +
                                omega / alpha * (F[j * M + i] -
gamma * U[(j - 1) * M + i] - beta * U_prev[j * M + i + 1]);

                for (i = 1; i < M - 1; ++i)
                {
                        U[j * M + i] = (1 - omega) * U_prev[j * M + i] +
                                        omega / alpha * (F[j * M + i] -
gamma * U[(j - 1) * M + i] - beta * U[j * M + (i - 1)] - beta * U_prev[j * M + i +
1]);
                }

                U[j * M + i] = (1 - omega) * U_prev[j * M + i] +
```

```
                                                omega / alpha * (F[j * M + i] -
gamma * U[(j - 1) * M + i] - beta * U[j * M + (i - 1)]);

                                error = 0.;
                                for (int k = 0; k < N * M; ++k)
                                {
                                        error = max(abs(U_prev[k] - U[k]), error);
                                }

                        } while (error > delta);

                        error = 0.;

                        for (int j = 0; j < N; ++j)
                        {
                                for (int i = 0; i < M; ++i)
                                {
                                        double x = i * hx;
                                        double y = j * hy;
                                        error = max(error, abs(U[j * M + i] - sin(M_PI * x
* x / lx / lx) * sin(M_PI * y / ly)));
                                }
                        }

                        cout << "SOR method N = " << N << " M  = " << M << " "
                                << " error: " << error << " iter count: " << iter_count <<
endl;
                }

                // Прямой метод:
                {
                        for (auto N : N_arr)
                        {
                                double hy = ly / N;
                                double hx = hy * lx / ly;

                                int M = long long(lx / hx); // M число по x

                                vector<double> U(N * M, 0.); // Начальное
приближение нулевой вектор

                                double alpha = -2. * (1 / hx / hx + 1 / hy / hy);
                                double beta = 1 / hx / hx;
                                double gamma = 1 / hy / hy;
```

```
                        vector<double> F = getF(N, M, hx, hy);

                        EllipticPDEMatrixGen pm(alpha, beta, gamma, N, M);

                        vector<double> Lower(M * N * M * N);
                        vector<double> Upper(M * N * M * N);

                        lu_decompostion(N * M, pm, Lower, Upper);

                        solve_lu(N * M, Lower, Upper, F, U);

                        double error = 0.;

                        for (int j = 0; j < N; ++j)
                        {
                                for (int i = 0; i < M; ++i)
                                {
                                        double x = i * hx;
                                        double y = j * hy;
                                        error = max(error, abs(U[j * M + i] -
sin(M_PI * x * x / lx / lx) * sin(M_PI * y / ly)));
                                }
                        }

                        cout << "Gauss N = " << N << " M  = " << M << " "
                                << " error: " << error << endl;
                }
        }
    }
}

#define _USE_MATH_DEFINES
#include <iostream>
#include <cmath>
#include <vector>
#include <span>
#include <fstream>
#include <format>
#include <Eigen/Sparse>
#include <Eigen/Dense>

using namespace Eigen;
using namespace std;
```

```
const double delta = 1e-8;

int main() {
	// Метод простых итераций с правильным параметром tau
https://zaochnik.com/spravochnik/matematika/issledovanie-slau/iteratsionnye-metody-reshenija-slau/
	vector<int> N_arr = { 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170 };
	//vector<int> N_arr = { 5 };
	// Part I
	const double lx = 1.;
	const double ly = 2.;

	auto f = [](double x, double y) {
		return -0.2e1 * 0.3141592654e1 * cos(0.3141592654e1 * x * x) * sin(0.3141592654e1 * y / 0.2e1) + 0.4e1 * 0.3141592654e1 * 0.3141592654e1 * x * x * sin(0.3141592654e1 * x * x) * sin(0.3141592654e1 * y / 0.2e1) + sin(0.3141592654e1 * x * x) * 0.3141592654e1 * 0.3141592654e1 * sin(0.3141592654e1 * y / 0.2e1) / 0.4e1;;
	};

	auto phi1 = [](double x = 0, double y = 0) {
		return 0;
	};

	auto phi2 = [](double x = 0, double y = 0) {
		return 0;
	};

	auto phi3 = [](double x = 0, double y = 0) {
		return 0;
	};

	auto phi4 = [](double x = 0, double y = 0) {
		return 0;
	};

	auto b = [](double x, double y) {
		return 1 + y * y;
	};

	auto by = [](double x, double y) {
		return 2 * y;
	};
```

```
	auto a = [](double x, double y) {
		return 1;
	};

	auto ax = [](double x, double y) {
		return 0;
	};

	auto c = [](double x, double y) {
		return x * x - y * y;
	};

	auto getF = [phi1, phi2, phi3, phi4, f](int N, int M, double hx, double hy) {
		VectorXd F(N * M);
		{
			int i, j;
			// F для левой границы
			i = 0;
			j = 0;
			F[j * M + i] = -f(i * hx, j * hy) - phi1() / hx / hx - phi3() / hy /
hy;
			for (j = 1; j < N - 1; ++j) {
				F[j * M + i] = -f(i * hx, j * hy) - phi1() / hx / hx;
			}
			F[j * M + i] = -f(i * hx, j * hy) - phi1() / hx / hx - phi4() / hy /
hy;

			// F для верхней границы
			for (i = 1; i < M - 1; ++i) {
				F[j * M + i] = -f(i * hx, j * hy) - phi4() / hy / hy;
			}

			F[j * M + i] = -f(i * hx, j * hy) - phi2() / hx / hx - phi4() / hy /
hy;

			// F для правой границы
			for (j = N - 2; j > 0; --j) {
				F[j * M + i] = -f(i * hx, j * hy) - phi2() / hx / hx;
			}
			F[j * M + i] = -f(i * hx, j * hy) - phi2() / hx / hx - phi3() / hy /
hy;

			// F  для нижней границы
			for (i = M - 2; i > 0; --i) {
				F[j * M + i] = -f(i * hx, j * hy) - phi3() / hy / hy;
```

```
                }

                // внутренняя область
                for (j = 1; j < N - 1; ++j) {
                        for (i = 1; i < M - 1; ++i) {
                                F[j * M + i] = -f(i * hx, j * hy);
                        }
                }
        }
        return F;
};

std::fstream output;

for (auto N : N_arr) {
        int M = N;
        double hx = lx / N;
        double hy = ly / M;
        cout << "Process: " << N << " " << M << endl;
        std::vector<Eigen::Triplet<double>> triplets;
        // заполним левые гаммы
        for (int j = 1; j < M; ++j) {
                for (int i = 0; i < N; ++i) {
                        double value = -by(i * hx, j * hy) / 2. / hy + b(i * hx, j *
hy) / hy / hy;
                        triplets.push_back(Eigen::Triplet<double>(j * N + i, (j -
1) * N + i, value));
                }
        }

        // правые гаммы
        for (int j = 0; j < M - 1; ++j) {
                for (int i = 0; i < N; ++i) {
                        double value = by(i * hx, j * hy) / 2. / hy + b(i * hx, j *
hy) / hy / hy;
                        triplets.push_back(Eigen::Triplet<double>(j * N + i, (j +
1) * N + i, value));
                }
        }

        // альфы
        for (int j = 0; j < M; ++j) {
                for (int i = 0; i < N; ++i) {
                        double value = -2. * (a(i * hx, j * hy) / hx / hx + b(i * hx, j
* hy) / hy / hy) + c(i * hx, j * hy);
```

```
                        triplets.push_back(Eigen::Triplet<double>(j * N + i, j * N
+ i, value));
                }
        }

        // левые беты
        for (int j = 0; j < M; ++j) {
                for (int i = 1; i < N; ++i) {
                        double value = -ax(i * hx, j * hy) / 2 / hx + a(i * hx, j *
hy) / hx / hx;
                        triplets.push_back(Eigen::Triplet<double>(j * N + i, j * N
+ i - 1, value));
                }
        }

        // правые беты
        for (int j = 0; j < M; ++j) {
                for (int i = 0; i < N-1; ++i) {
                        double value = ax(i * hx, j * hy) / 2 / hx + a(i * hx, j * hy)
/ hx / hx;
                        triplets.push_back(Eigen::Triplet<double>(j * N + i, j * N
+ i + 1, value));
                }
        }

        Eigen::SparseMatrix<double> A(N * M, N * M);
        A.setFromTriplets(triplets.begin(), triplets.end());

        ConjugateGradient<SparseMatrix<double>, Lower | Upper> cg;
        cg.compute(A);
        auto b = getF(N, M, hx, hy);
        auto U = cg.solve(b);


        std::string path = std::format("ex_{}_{}.txt", 4, N * M);
        output.open(path, std::ios::out);
        if (!output)
        {
                std::cout << "File not created!";
        }
        else
        {
                for (int j = 0; j < M; ++j) {
                        for (int i = 0; i < N; ++i) {
```

```
                                        output << i * hx << " " << j * hy << " " << U[j * N
+ i] << "\n";
                                }
                        }
                        output.close();
                }

        }

}
```