**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

Лабораторная работа №3

**Разностная схема для уравнения Пуассона**

Вариант 5

**Выполнил:**

Кендысь Алексей Максимович

студент 3 курса, 7 группы,

специальность

“прикладная математика”

**Преподаватель:**

Доцент кафедры вычислительной

математики ФПМИ,

А.М. Будник

Минск, 2023

**Содержание:**

Постановка задачи 2

Разностная схема 2

Теоретические сведения 2

Листинг программы 3

Результаты 7

Выводы 8

Постановка задачи

Дана задача Дирихле для уравнения Пуассона в прямоугольнике :

Найти приближённое решение разностным итерационным методом на сетке узлов с шагами . В качестве итерационного метода использовать метод Зейделя.

Разностная схема

Теоретические сведения

Рассмотрим разбиение области с величиной шага по переменной и по переменной . Количество разбиений (переменная ) и (переменная ).

Узлы – (, где , *,* , *,* внутренние точки разбиения.

Шаблон:

Разностная схема для уравнения Пуассона на сетке  и на заданном шаблоне записывается в виде (безындексная форма):

где – сеточная функция, , – вторая разностная производная по переменной . Аналогично определяется .

Запишем индексную форму данной разностной схемы:

Данная схема имеет второй порядок аппроксимации по каждой переменной, т.е. .

Если разрешим уравнение относительно центрального элемента шаблона, то получим:

Тогда метод Зейделя имеет следующий вид:

При этом вычисления останавливаем, если , где в качестве берётся .

В качестве начального приближения берётся .

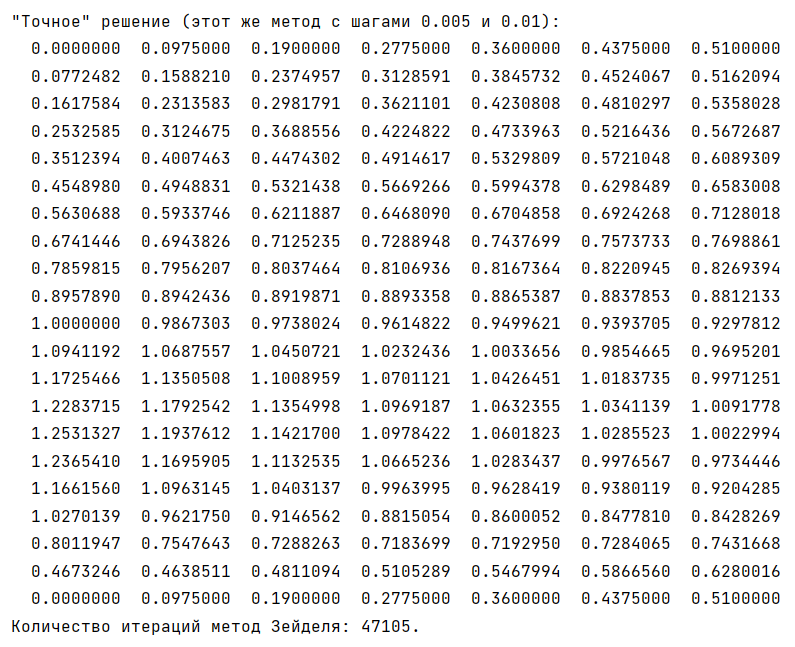
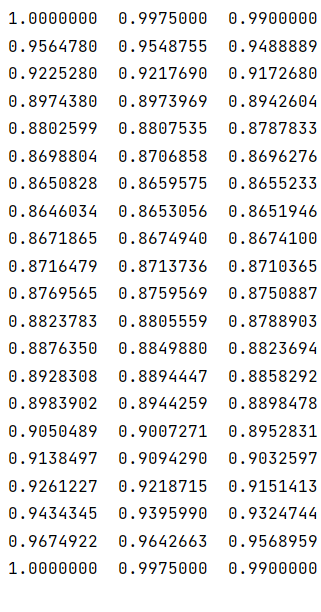
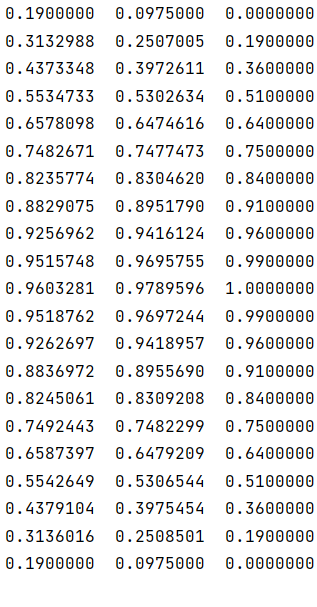
В качестве точного решения берётся приближённое решение, полученное при использовании этой же схемы с шагами .

Листинг программы

import java.util.\*;  
  
class Functions {  
 public static double f(double x, double y) {  
 return Math.*abs*((x \* x) - (y \* y));  
 }  
  
 public static double psi1(double y) {  
 return 1 - Math.*pow*(y, 2.);  
 }  
  
 public static double psi2(double y) {  
 return Math.*exp*(y) \* (1 - Math.*pow*(y, 2.));  
 }  
  
 public static double psi3(double x) {  
 return 1 - Math.*pow*(x, 2.);  
 }  
  
 public static double psi4(double x) {  
 return 1 - Math.*pow*(x, 2.);  
 }  
}  
  
class PoissonEquationSolver {  
 private final int n1;  
 private final int n2;  
 private final double h1;  
 private final double h2;  
 private final double eps;  
 private final double[] x1;  
 private final double[] x2;  
 private double[][] y;  
 private int k;  
  
 public PoissonEquationSolver(int n1, int n2, double h1, double h2, double eps, double[] x1, double[] x2) {  
 this.n1 = n1;  
 this.n2 = n2;  
 this.h1 = h1;  
 this.h2 = h2;  
 this.eps = eps;  
 this.x1 = x1;  
 this.x2 = x2;  
  
 genY();  
 }  
  
 public double[][] getY() {  
 return y;  
 }  
  
 public int getK() {  
 return k;  
 }  
  
 private void genY() {  
 double[][] res = new double[n1 + 1][n2 + 1];  
 setBoundaryValues(res);  
 setInitialValues(res);  
  
 k = 0;  
 double checkStop;  
 do {  
 checkStop = 0;  
  
 for (int i = 1; i <= n1 - 1; i++) {  
 for (int j = 1; j <= n2 - 1; j++) {  
 double old = res[i][j];  
  
 res[i][j] = ((res[i + 1][j] + res[i - 1][j]) / Math.*pow*(h1, 2.));  
 res[i][j] += ((res[i][j + 1] + res[i][j - 1]) / Math.*pow*(h2, 2.));  
 res[i][j] += Functions.*f*(x1[i], x2[j]);  
 res[i][j] /= (2. / Math.*pow*(h1, 2.)) + (2. / Math.*pow*(h2, 2.));  
  
 if (Math.*abs*(res[i][j] - old) > checkStop) {  
 checkStop = Math.*abs*(res[i][j] - old);  
 }  
 }  
 }  
  
 k++;  
 } while(checkStop > eps);  
  
 y = res;  
 }  
  
 private void setBoundaryValues(double[][] y) {  
 for (int j = 0; j <= n2; j++) {  
 y[0][j] = Functions.*psi1*(x2[j]);  
 }  
  
 for (int j = 0; j <= n2; j++) {  
 y[n1][j] = Functions.*psi2*(x2[j]);  
 }  
  
 for (int i = 0; i <= n1; i++) {  
 y[i][0] = Functions.*psi3*(x1[i]);  
 }  
  
 for (int i = 0; i <= n1; i++) {  
 y[i][n2] = Functions.*psi4*(x1[i]);  
 }  
 }  
  
 private void setInitialValues(double[][] y) {  
 for (int i = 1; i <= n1 - 1; i++) {  
 for (int j = 1; j <= n2 - 1; j++) {  
 y[i][j] = Functions.*f*(x1[i], x2[j]);  
 }  
 }  
 }  
}  
  
class DirichletProblem {  
 private static final double *A* = -1.;  
 private static final double *B* = 1.;  
 private static final double *C* = -1.;  
 private static final double *D* = 1.;  
 private static final double *H1* = 0.05;  
 private static final double *H2* = 0.1;  
 private static final double *EPS* = Math.*max*(Math.*pow*(*H1*, 3.), Math.*pow*(*H2*, 3.));  
 private static final double *H1\_EXACT* = 0.005;  
 private static final double *H2\_EXACT* = 0.01;  
 private static final double *EPS\_EXACT* = Math.*max*(Math.*pow*(*H1\_EXACT*, 3.), Math.*pow*(*H2\_EXACT*, 3.));  
 private final int n1;  
 private final int n2;  
 private final int n1Exact;  
 private final int n2Exact;  
 private final double[] x1;  
 private final double[] x2;  
 private final double[] x1Exact;  
 private final double[] x2Exact;  
 private final double[][] u;  
 private int kExact;  
 private double[][] y;  
 private int k;  
 private double[][] res;  
  
 public DirichletProblem() {  
 n1 = (int) ((*B* - *A*) / *H1*);  
 n2 = (int) ((*D* - *C*) / *H2*);  
 n1Exact = (int) ((*B* - *A*) / *H1\_EXACT*);  
 n2Exact = (int) ((*D* - *C*) / *H2\_EXACT*);  
  
 x1 = getX(*H1*, n1, *A*);  
 x2 = getX(*H2*, n2, *C*);  
 x1Exact = getX(*H1\_EXACT*, n1Exact, *A*);  
 x2Exact = getX(*H2\_EXACT*, n2Exact, *C*);  
  
 u = getExact();  
 }  
  
 public void solve() {  
 PoissonEquationSolver peSolver = new PoissonEquationSolver(n1, n2, *H1*, *H2*, *EPS*, x1, x2);  
 y = peSolver.getY();  
 k = peSolver.getK();  
 res = getResidual(y);  
 }  
  
 public void out() {  
 Formatter fmt = new Formatter();  
  
 fmt.format("\n\"Точное\" решение (этот же метод с шагами 0.005 и 0.01):\n");  
 outY(fmt, u);  
 fmt.format("Количество итераций метод Зейделя: %d.\n", kExact);  
  
 fmt.format("\nПолученное решение:\n");  
 outY(fmt, y);  
 fmt.format("Количество итераций метод Зейделя: %d.\n", k);  
 fmt.format("Вектор невязок:\n");  
 outRes(fmt, res);  
  
 System.*out*.println(fmt);  
 }  
  
 private void outY(Formatter fmt, double[][] y) {  
 for (int j = 0; j <= n2; j++) {  
 for (int i = n1; i >= 0; i--) {  
 fmt.format("% 11.7f", y[i][j]);  
 }  
  
 fmt.format("\n");  
 }  
 }  
  
 private void outRes(Formatter fmt, double[][] res) {  
 for (int j = 0; j <= n2; j++) {  
 for (int i = n1; i >= 0; i--) {  
 fmt.format("% 11.3E", res[i][j]);  
 }  
  
 fmt.format("\n");  
 }  
 }  
  
 private double[][] getResidual(double[][] y) {  
 double[][] res = new double[n1 + 1][n2 + 1];  
  
 for (int i = 0; i <= n1; i++) {  
 for (int j = 0; j <= n2; j++) {  
 res[i][j] = Math.*abs*(u[i][j] - y[i][j]);  
 }  
 }  
  
 return res;  
 }  
  
 private double[][] getExact() {  
 double[][] res = new double[n1 + 1][n2 + 1];  
 double[][] fullRes;  
  
 PoissonEquationSolver peSolver = new PoissonEquationSolver  
 (n1Exact, n2Exact, *H1\_EXACT*, *H2\_EXACT*, *EPS\_EXACT*, x1Exact, x2Exact);  
 fullRes = peSolver.getY();  
 kExact = peSolver.getK();  
  
 for (int i = 0, k = 0; i <= n1Exact; i += n1Exact / n1, k++) {  
 for (int j = 0, s = 0; j <= n2Exact; j += n2Exact / n2, s++) {  
 res[k][s] = fullRes[i][j];  
 }  
 }  
  
 return res;  
 }  
  
 private double[] getX(double h, int n, double a) {  
 double[] res = new double[n + 1];  
 for (int i = 0; i <= n; i++) {  
 res[i] = a + (i \* h);  
 }  
  
 return res;  
 }  
}  
  
public class Main {  
  
 public static void main(String[] args) {  
 DirichletProblem dp = new DirichletProblem();  
 dp.solve();  
 dp.out();  
 }  
}

Результаты

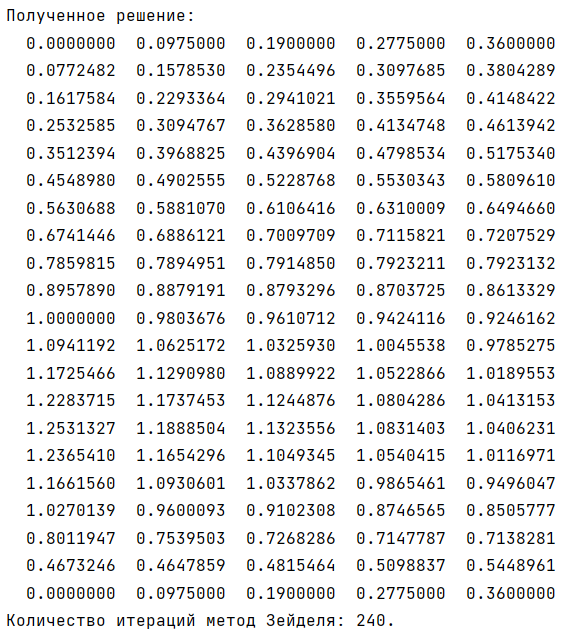
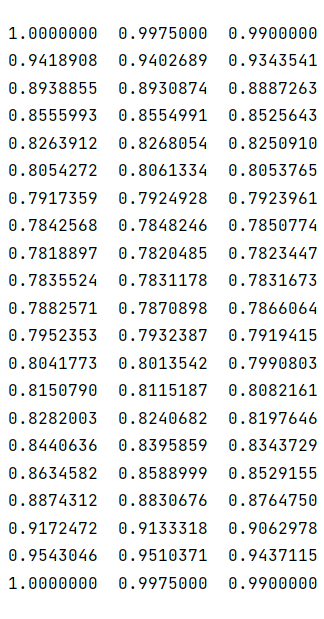
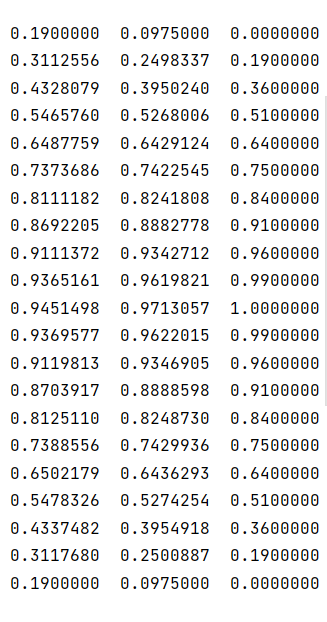
“Точное” решение (этот же метод с шагами 0.005 и 0.01):

..........

(Выведены начало, середина и конец таблицы значений)

Количество итераций метод Зейделя: 47105.

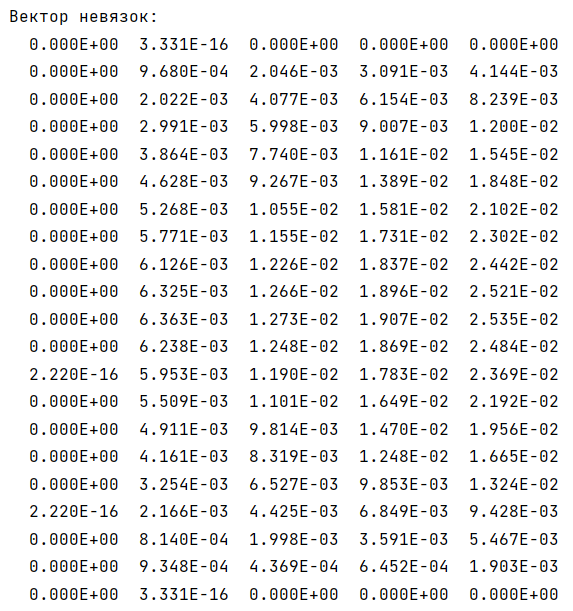
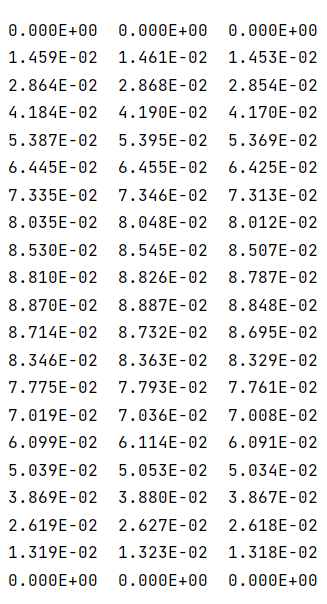
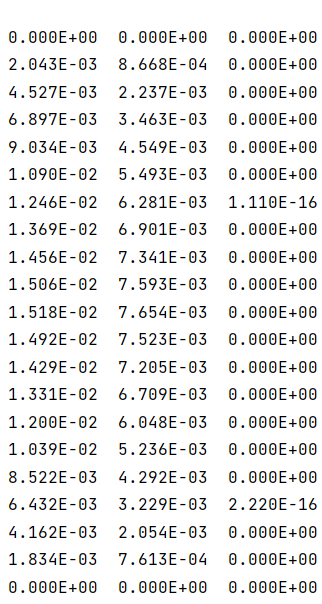
Полученное решение:

..........

(Выведены начало, середина и конец таблицы значений)

Количество итераций метода Зейделя: 240.

Невязка:

..........

(Выведены начало, середина и конец таблицы значений)

Выводы

Разностная схемы, которую мы использовали, имеет второй порядок аппроксимации по каждой переменной. Соответственно, погрешность аппроксимации составляет в нашем случае и , т.е. погрешность определяется слагаемым с и должна иметь порядок .

Значения на границе вычисляются точно из начального и граничного условия, поэтому там невязка = 0 (либо = машинному эпсилону).

Из вида вектора невязок (разница между “точным” и полученным решением) для полученного решения можем сказать, что реальная погрешность на внутренних узлах составляет примерно от до , т.е. примерно соответствует ожидаемой погрешности – . Также видно, что погрешность увеличивается при приближении к середине области разбиения. Это связано с видом шаблона разностной схемы, погрешность при вычислении каждого значения возрастает, т.к. предыдущие значения были также вычислены с погрешностью.

Также заметим, что из-за выбора итерационный процесс (метод Зейделя) не должен был повысить погрешность решения, чего и не произошло. Также видно, что для вычисления “точного” решения понадобилось достаточно большое количество итераций (k = 47105, программа работала примерно 1 минуту), т.е. метод не оптимизирован для малых значений шага, и нет смысла далее понижать шаг из-за вычислительной трудности метода.