­БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа №3

**Разностные схемы для уравнения Пуассона**

Вариант 4

**Выполнила:**

Зуйкевич Лидия

3 курс 7 группа

**Преподаватель:**

Будник А.М.

Минск, 2023

Оглавление

[Постановка задачи 3](#_Toc118753274)

[Алгоритм решения 3](#_Toc118753275)

[Листинг программы 4](#_Toc118753276)

[Вывод программы 6](#_Toc104902787)

[Выводы 7](#_Toc104902787)

# Постановка задачи

# Дана задача Дирихле для уравнения Пуассона в прямоугольнике

Найти приближенное решение разностным методом релаксации на сетке узлов с шагами

# Алгоритм решения

Используя граничные условия, находим решение в граничных узлах:

Где - количество разбиений по х, по у соответственно; , а – приближенное решение в узле

Приближенное решение во внутренних узлах найдем итерационным методом релаксации по формуле:

(Нужно дописать, как формула получилась)

; - параметр релаксации.

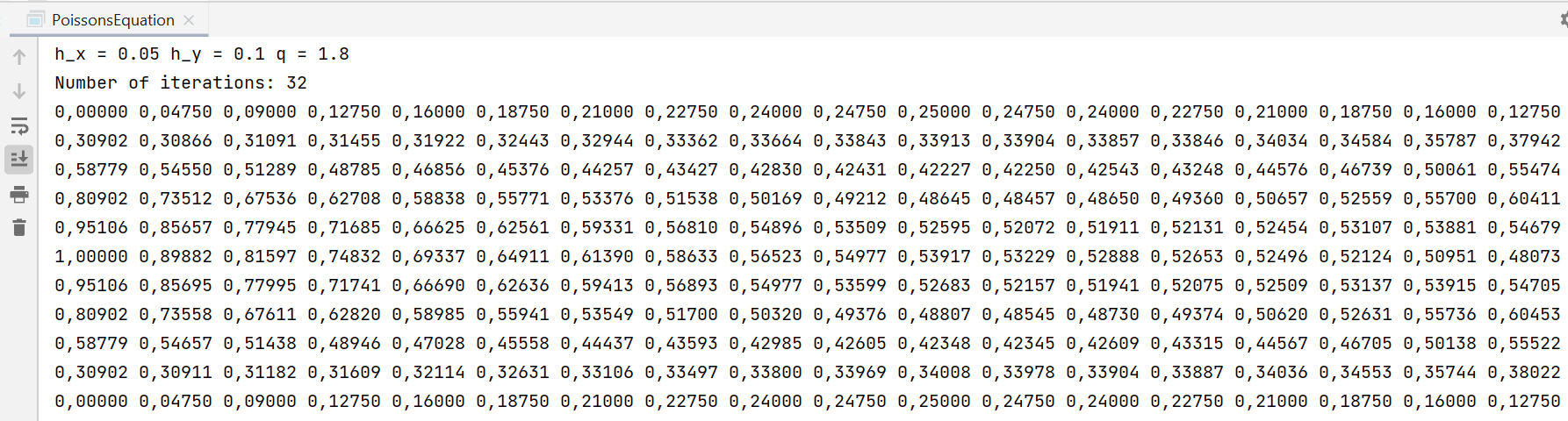
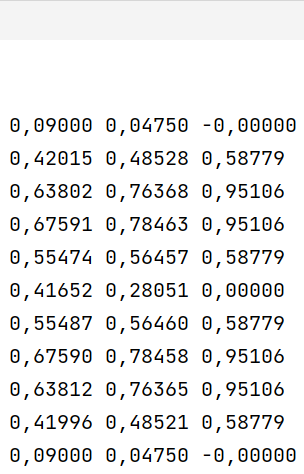
Критерий остановки: .

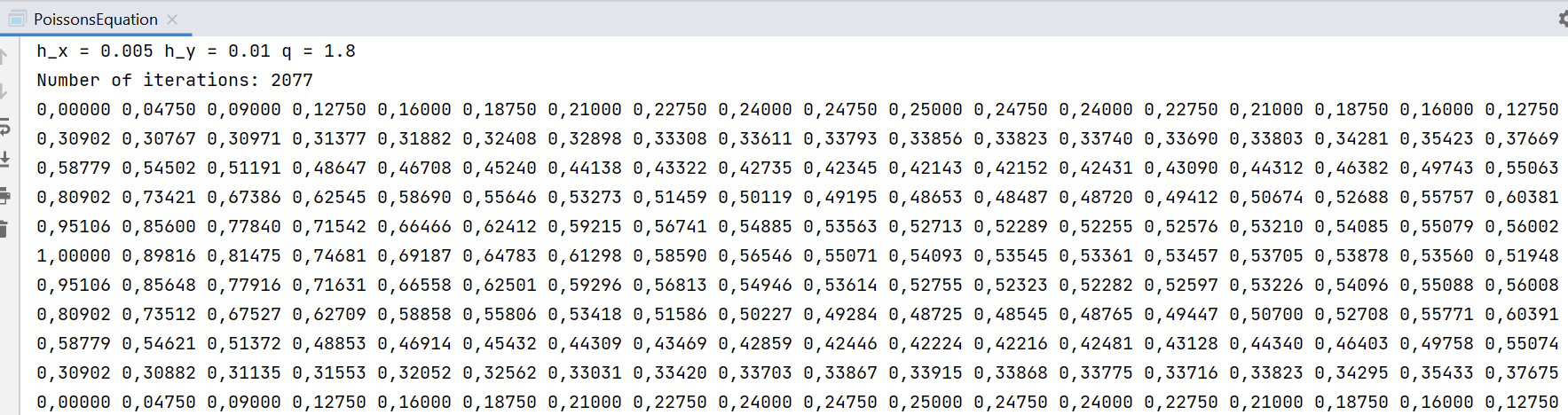
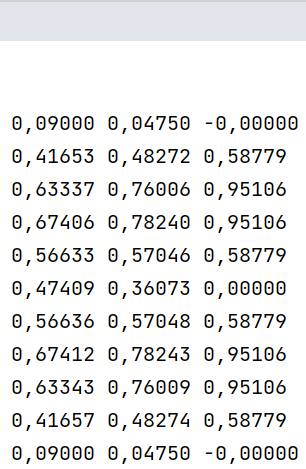
# Листинг программы

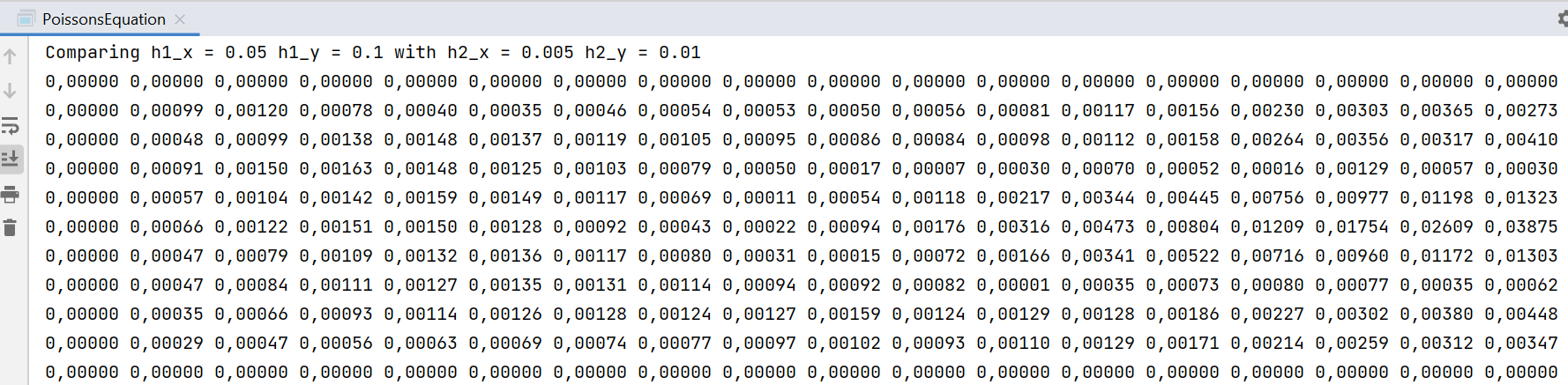
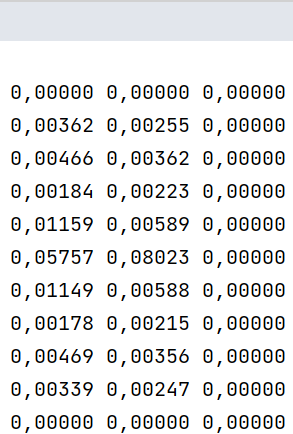
import java.util.Arrays;  
  
public class PoissonsEquation {  
 private final double h\_x;  
 private final double h\_y;  
 private final int n\_x;  
 private final int n\_y;  
 private final double EPS;  
  
 private final double a;  
 private final double c;  
 private final double q;  
  
 private double[][] prev;  
 private double[][] next;  
 private final int print\_param\_x;  
 private final int print\_param\_y;  
  
 public PoissonsEquation(double h\_x, double h\_y, double a, double b, double c, double d, double q) {  
 this.h\_x = h\_x;  
 this.h\_y = h\_y;  
 this.EPS = q \* Math.*max*(Math.*pow*(h\_x, 3), Math.*pow*(h\_y, 3));  
 this.n\_x = (int) ((b - a)/h\_x);  
 this.n\_y = (int) ((d - c)/h\_y);  
 this.print\_param\_x = n\_x / 20;  
 this.print\_param\_y = n\_y / 10;  
  
 this.a = a;  
 this.c = c;  
 this.q = q;  
 this.prev = new double[n\_y + 1][n\_x + 1];  
 this.next = new double[n\_y + 1][n\_x + 1];  
 }  
  
 public double getH\_x() {  
 return h\_x;  
 }  
  
 public double getH\_y() {  
 return h\_y;  
 }  
  
 public double[][] getNext() {  
 return next;  
 }  
  
 public int getPrint\_param\_x() {  
 return print\_param\_x;  
 }  
  
 public int getPrint\_param\_y() {  
 return print\_param\_y;  
 }  
  
 public double psi1(double y){  
 return Math.*sin*(Math.*PI* \* y);  
 }  
  
 public double psi2(double y){  
 return Math.*abs*(Math.*sin*(2 \* Math.*PI* \* y));  
 }  
  
 public double psi3(double x){  
 return - x \* (x + 1);  
 }  
  
 public double psi4(double x){  
 return - x \* (x + 1);  
 }  
  
 public double f(int i, int j){  
 return Math.*cosh*(Math.*pow*(a + i \* h\_x, 2) \* (c + j \* h\_y));  
 }  
  
 public void solve(){  
 System.*out*.println("h\_x = " + h\_x + " h\_y = " + h\_y + " q = " + q);  
  
 double val\_y = c;  
 for(int i = 1; i < n\_y; i++){  
 val\_y += h\_y;  
 next[i][0] = psi1(val\_y);  
 next[i][n\_x] = psi2(val\_y);  
 }  
  
 double val\_x = a;  
 for(int i = 0; i <= n\_x; i++){  
 next[0][i] = psi3(val\_x);  
 next[n\_y][i] = psi4(val\_x);  
 val\_x += h\_x;  
 }  
  
 double hx\_2 = Math.*pow*(h\_x, 2);  
 double hy\_2 = Math.*pow*(h\_y, 2);  
 double coeff = q / (2. / hx\_2 + 2. / hy\_2);  
  
 sor(hx\_2, hy\_2, coeff);  
 print\_matrix(next);  
 System.*out*.println();  
 }  
  
 public void copyMatrix(double[][] destination, double[][] source){  
 for (int i = 0; i <= n\_y; i++) {  
 destination[i] = Arrays.*copyOf*(source[i], n\_x + 1);  
 }  
 }  
  
 public void sor(double hx\_2, double hy\_2, double coeff){  
 double max;  
 int num = 0;  
 do{  
 copyMatrix(prev, next);  
 max = - Double.*MAX\_VALUE*;  
  
 for (int j = 1; j < n\_y; j++) {  
 for (int i = 1; i < n\_x; i++) {  
 next[j][i] = coeff \* ((prev[j][i+1] + next[j][i - 1]) / hx\_2 + (prev[j + 1][i] + next[j - 1][i]) / hy\_2 + f(i, j)) + (1 - q) \* prev[j][i];  
 double res = Math.*abs*(next[j][i] - prev[j][i]);  
  
 max = Math.*max*(res, max);  
 }  
 }  
 num++;  
 } while(max > EPS);  
 System.*out*.println("Number of iterations: " + num);  
 }  
  
 public void print\_matrix(double[][] matr){  
 for(int j = 0; j <= n\_y; j += print\_param\_y){  
 for(int i = 0; i <= n\_x; i += print\_param\_x){  
 System.*out*.print(String.*format*("%7.5f", matr[j][i]) + " ");  
 }  
 System.*out*.println();  
 }  
 }  
  
 public void compare(PoissonsEquation obj){  
 double h2x = obj.getH\_x();  
 double h2y = obj.getH\_y();  
 System.*out*.println("Comparing h1\_x = " + h\_x +" h1\_y = " + h\_y + " with h2\_x = " + h2x + " h2\_y = " + h2y);  
 double[][] y2 = obj.getNext();  
 int i2 = obj.getPrint\_param\_x();  
 int j2 = obj.getPrint\_param\_y();  
  
 for (int i = 0; i <= 10; i++) {  
 for(int j = 0; j <= 20; j++){  
 double elem = Math.*abs*(next[i \* print\_param\_y][j \* print\_param\_x] - y2[i \* j2][j \* i2]);  
 System.*out*.print(String.*format*("%7.5f", elem) + " ");  
 }  
 System.*out*.println();  
 }  
 System.*out*.println();  
 }  
  
  
 public static void main(String[] args){  
 double q = 1.8;  
 PoissonsEquation obj = new PoissonsEquation(0.05, 0.1, -1, 0, 0, 1, q);  
 PoissonsEquation obj2 = new PoissonsEquation(0.005, 0.01, -1, 0, 0, 1, q);  
 obj.solve();  
 obj2.solve();  
 obj.compare(obj2);  
 }  
}

Вывод программы

Сравним значения для и :







Выводы

Данная схема имеет порядок аппроксимации , поэтому, учитывая значения шагов , погрешность решения должна быть порядка . При сравнении решения с решением с шагами (это решение взято в качестве точного), видно, что невязка имеет порядок , т.е. в большинстве узлов погрешность даже лучше, чем теоретическая, однако для погрешности используется не точное решение, а приближенное с меньшим шагом. Граничные условия 1 рода, поэтому они аппроксимируются точно. Значение параметра релаксации было выбрано в результате выполнения программы с различными значениями , при выборе значения параметра ориентировались на количество итераций метода релаксации. Так, для шагов при было 236 итераций, при - 127 итераций, при – 89 итераций, а при выбранном значении - всего 32 итерации. Таким образом, благодаря выбору параметра релаксации можно значительно снизить количество итераций метода и получить решение с требуемой погрешностью. Для количество итераций при равно 2077, эти значения тоже значительно меньше, чем при меньших значениях, но все равно достаточно велики, программа работает долго.