БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа №2

**Решение смешанной задачи для уравнения теплопроводности**

Вариант 1

**Выполнила:**

Зуйкевич Лидия

3 курс 7 группа

**Преподаватель:**

Будник А.М.

Минск, 2023

Оглавление

[Постановка задачи 3](#_Toc118753274)

[Алгоритм решения 3](#_Toc118753275)

[Листинг программы 5](#_Toc118753276)

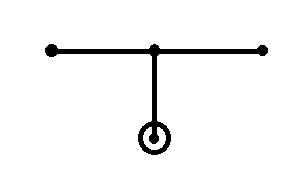
[Вывод программы 8](#_Toc104902787)

[Выводы 9](#_Toc104902787)

# Постановка задачи

# Для решения краевой задачи вида:

на сетке построить разностную схему с погрешностью аппроксимации не ниже , используя четырехточечный шаблон:



Необходимо:

1. Показать, что построенная разностная схема имеет заданный порядок аппроксимации.

2. Исследовать устойчивость полученной разностной схемы по начальным данным, используя метод гармоник.

3. Реализовать данную разностную схему при и , выбранном из условия устойчивости.

4. Оценить приближенное решение, анализируя погрешность аппроксимации при разных шагах.

# Алгоритм решения

1) Используя заданный шаблон, построим разностную схему вида:

Где функции , и константу выберем из условия требуемого порядка аппроксимации.

Рассмотрим погрешность аппроксимации:

1.

Для того, чтобы порядок аппроксимации уравнения был равен *,* достаточно выбрать *.*

2. Начальное и левое граничное условие аппроксимируются точно.

3.

Для достижения заданного порядка аппроксимации выберем: , .

# 2) Исследуем устойчивость полученной разностной схемы по начальным данным методом гармоник:

Так как , то при любых значениях , следовательно, схема безусловно устойчива. Шаг можем брать любым.

3) Алгоритм реализации разностной схемы:

Первый временной слой:

Далее на каждом временном слое методом разностной прогонки решаем систему вида:

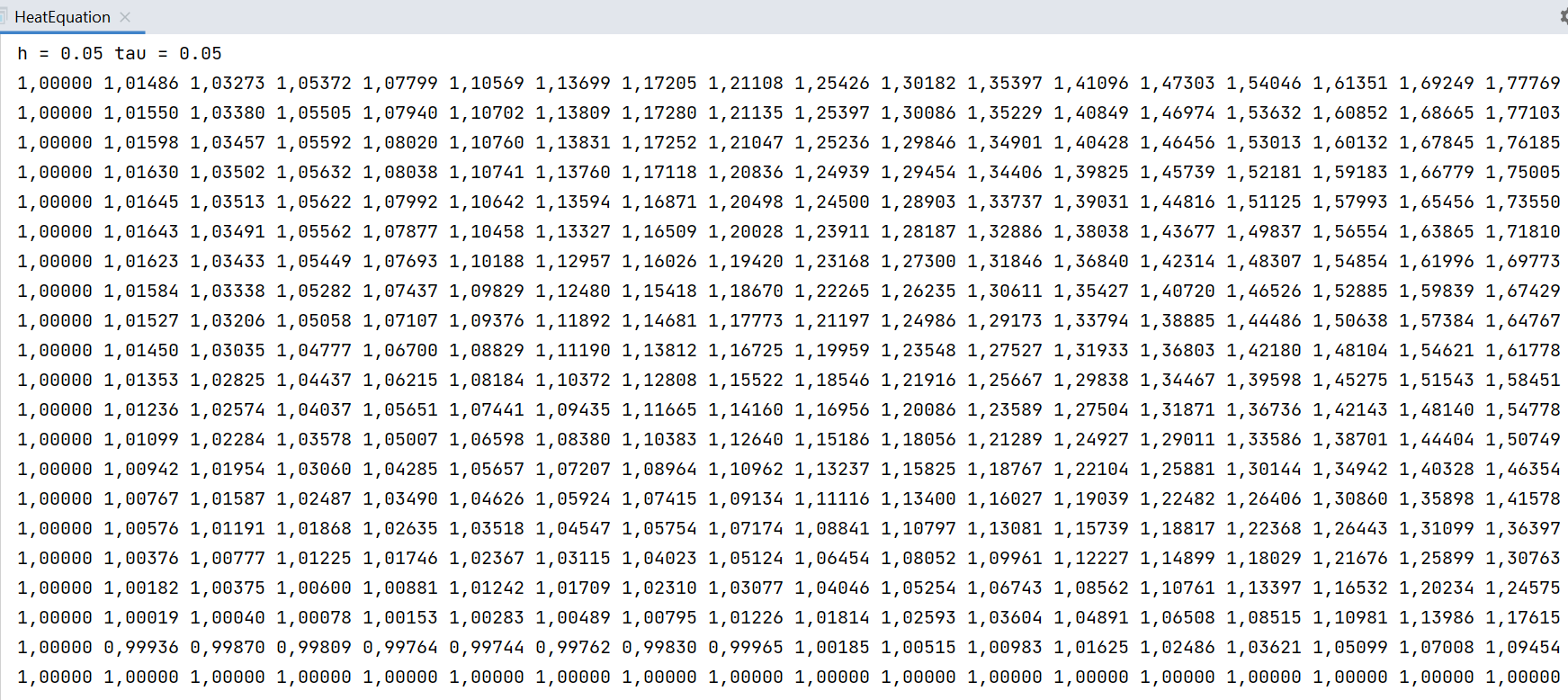
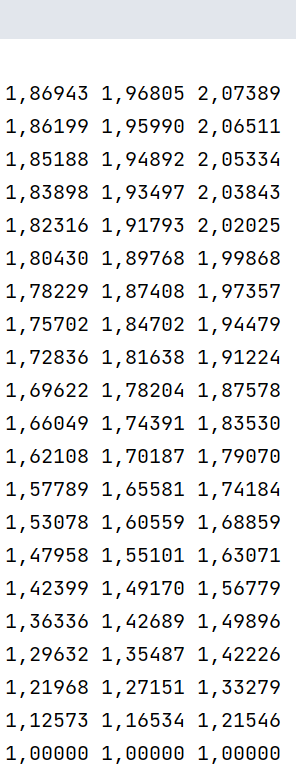
# Листинг программы

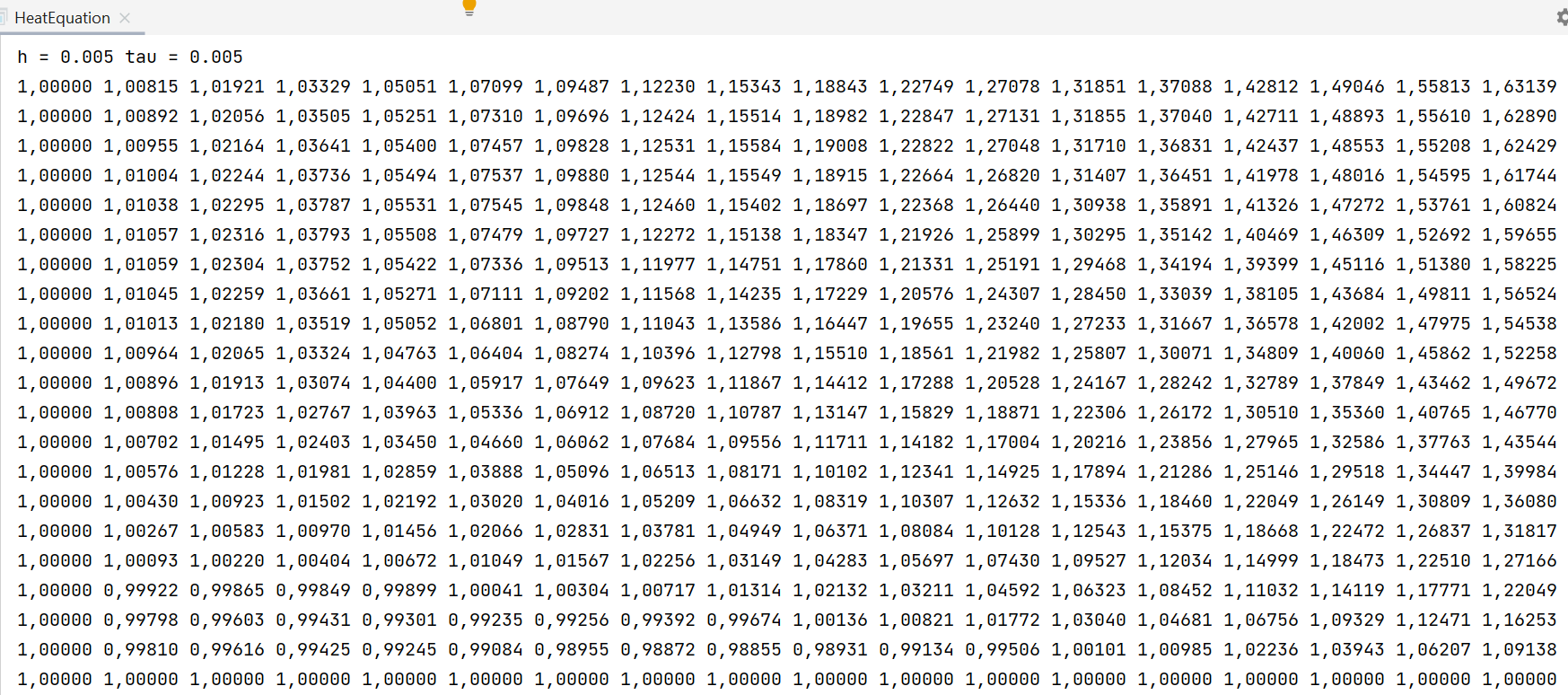
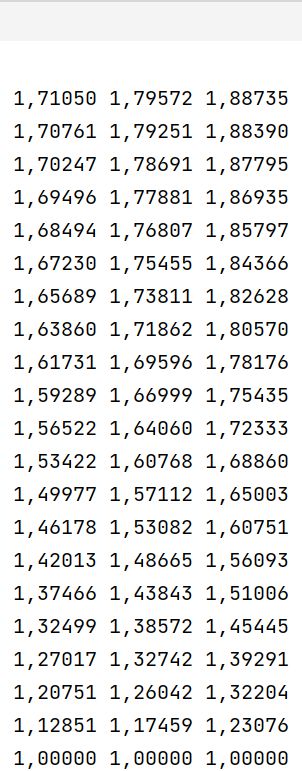
public class HeatEquation {  
 private double h;  
 private double tau;  
 private double[][] y;  
 private int n1;  
 private int n2;  
  
 private double a;  
 private double b;  
 private double c;  
 private int print\_param\_i;  
 private int print\_param\_j;  
 private double kappa;  
 private double[] f;  
 private double[] alpha;  
 private double[] beta;  
  
 public HeatEquation(double h, double tau) {  
 this.h = h;  
 this.tau = tau;  
 this.n1 = (int) (1/h);  
 this.n2 = (int) (1/tau);  
 this.y = new double[n2 + 1][n1 + 1];  
  
 this.print\_param\_i = n1 / 20;  
 this.print\_param\_j = n2 / 20;  
 this.a = tau / Math.*pow*(h, 2);  
 this.b = a;  
 this.c = 1 + (2 \* tau) / Math.*pow*(h, 2);  
 this.kappa = 1;  
 this.alpha = new double[n1];  
 this.beta = new double[n1];  
 }  
  
 public int getPrint\_param\_i() {  
 return print\_param\_i;  
 }  
  
 public int getPrint\_param\_j() {  
 return print\_param\_j;  
 }  
  
 public double[][] getY() {  
 return y;  
 }  
  
 public double getH() {  
 return h;  
 }  
  
 public double getTau() {  
 return tau;  
 }  
  
 public double phi(double i, double j){  
 return -(i \* h + Math.*pow*(j \* tau, 2)) \* Math.*exp*(- i \* h \* j \* tau);  
 }  
  
 public double mu(double j){  
 return (j \* tau - 1) \* Math.*exp*(-j \* tau);  
 }  
   
 public double f(int i, int j){  
 return tau \* phi(i, j) + y[j][i];  
 }  
  
 public void calculate\_alpha(){  
 alpha[0] = 0;  
 for (int i = 1; i < n1; i++) {  
 alpha[i] = b / (c - a \* alpha[i - 1]);  
 }  
 }  
  
 public void calculate\_beta(int j){  
 beta[0] = 1;  
 for (int i = 1; i < n1; i++) {  
 beta[i] = (f(i - 1, j - 1) + beta[i - 1] \* a) / (c - a \* alpha[i - 1]);  
 }  
 }  
  
 public void tridiagonal\_matrix\_algorithm(int j){  
 calculate\_beta(j);  
  
 double v = ((h \* mu(j)) / 2 + (1 + h / 2) \* y[j - 1][n1]) \* h;  
  
 y[j][n1] = (v + kappa \* beta[n1 - 1]) / (1 - alpha[n1 - 1] \* kappa);  
  
 for (int i = n1 - 1; i >= 0; i--){  
 y[j][i] = alpha[i] \* y[j][i + 1] + beta[i];  
 }  
 }  
  
 public void solve(){  
 System.*out*.println("h = " + h + " tau = " + tau);  
 for (int i = 0; i <= n1; i++) {  
 y[0][i] = 1;  
 }  
 calculate\_alpha();  
  
 for (int j = 1; j <= n2; j++) {  
 tridiagonal\_matrix\_algorithm(j);  
 }  
  
 for (int i = n2; i >= 0; i-=print\_param\_j) {  
 for(int j = 0; j <= n1; j+=print\_param\_i){  
 System.*out*.print(String.*format*("7%.5f", y[i][j]) + " ");  
 }  
 System.*out*.println();  
 }  
 System.*out*.println();  
 }  
  
  
 public void compare(HeatEquation obj){  
 double h2 = obj.getH();  
 double tau2 = obj.getTau();  
 System.*out*.println("Comparing h1 = " + h +" tau1 = " + tau + " with h2 = " + h2 + " tau2 = " + tau2);  
 double[][] y2 = obj.getY();  
 int i2 = obj.getPrint\_param\_i();  
 int j2 = obj.getPrint\_param\_j();  
  
 for (int i = 20; i >= 0; i--) {  
 for(int j = 0; j <= 20; j++){  
 double elem = Math.*abs*(y[i \* print\_param\_j][j \* print\_param\_i] - y2[i \* j2][j \* i2]);

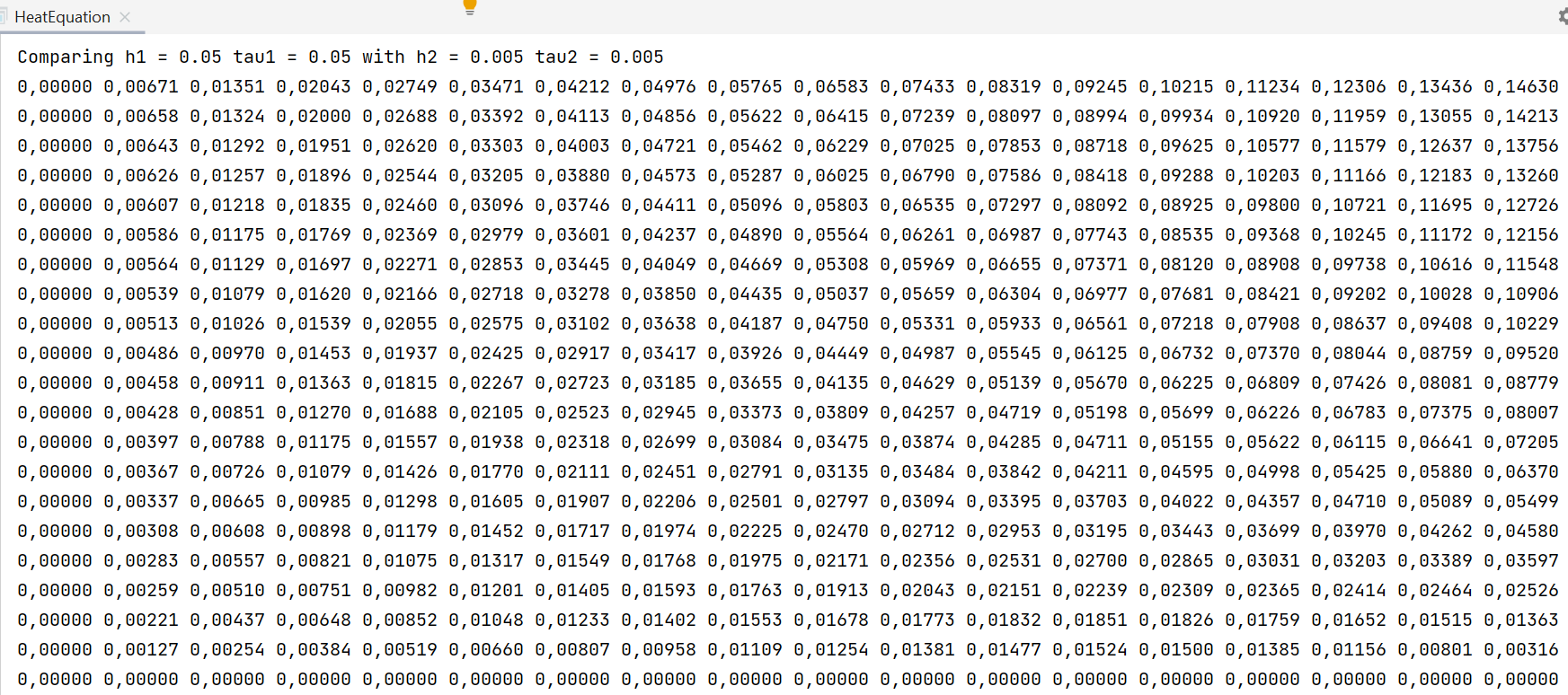
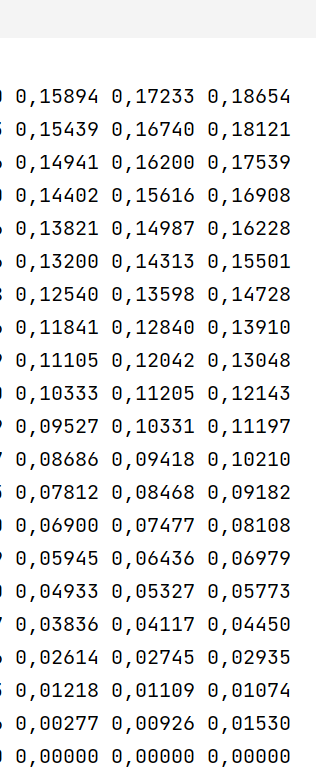
System.*out*.print(String.*format*("%7.5f", elem) + " ");  
 }  
 System.*out*.println();  
 }  
 System.*out*.println();  
 }  
  
  
 public static void main(String[] args){  
 HeatEquation obj = new HeatEquation(0.05, 0.05);  
 HeatEquation obj3 = new HeatEquation(0.005, 0.005);  
 obj.solve();  
 obj2.solve();  
 obj3.solve();  
  
 obj.compare(obj3);  
 }  
}

Вывод программы

Сравним значения для и :







Выводы

Построенная схема имеет порядок аппроксимации *,* т.е. невязка решения должна иметь порядок . Невязка при сравнении решений, полученных с шагами и (взято в качестве точного решения), имеет порядок , т.е. согласуется с теоретической погрешностью. Начальное и левое граничное условие аппроксимируются точно.