**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

Лабораторная работа №2

**Решение краевой задачи для уравнения переноса**

Вариант П-4

**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 4

Результаты 7

Выводы 9

Постановка задачи

Для решения краевой задачи вида:

в области на сетке построить разностную схему с весами, используя четырёхточечный шаблон:

Необходимо:

1. Определить порядок аппроксимации разностной схемы при и .
2. Исследовать устойчивость разностной схемы, используя принцип максимума.
3. Реализовать данную разностную схему при и , выбранным из условия устойчивости.
4. Оценить приближённое решение, анализируя погрешность аппроксимации, а также в сравнении с точным решением исходной задачи.

Разностная схема

Теоретические сведения

**Точное решение**

Исходная краевая задача имеет аналитическое решение, которое может быть найдено с помощью метода характеристик:

.

**Схема с весами**

Рассмотрим разбиение области для разбиений по переменной и разбиений по переменной . Величина шага и .

Узлы – (, где , *,* , *,* внутренние точки разбиения (без и).

Схема с весами для уравнения переноса на сетке и на заданном шаблоне записывается в виде (безындексная форма):

где – сеточная функция, , ̶ правая разностная производная по переменной , ̶ правая разностная производная по переменной ,

Запишем индексную форму данной разностной схемы:

**Порядок аппроксимации**

Тогда при .

И при .

Т.е. данная схема имеет первый порядок аппроксимации по каждой переменной.

**Устойчивость**

Используя индексную форму, перепишем аппроксимацию уравнения:

Т.е.

.

Таким образом, получаем условия:

Тогда при схема является условно устойчивой. Она устойчива при

, т.е. .

При схема является абсолютно устойчивой. Она устойчива при .

**Реализация схем**

Для выполнения условия устойчивости возьмём .

Обозначим за .

Для схема реализуется следующим образом:

1. ;
2. ;

Для схема реализуется следующим образом:

1. ;
2. ;

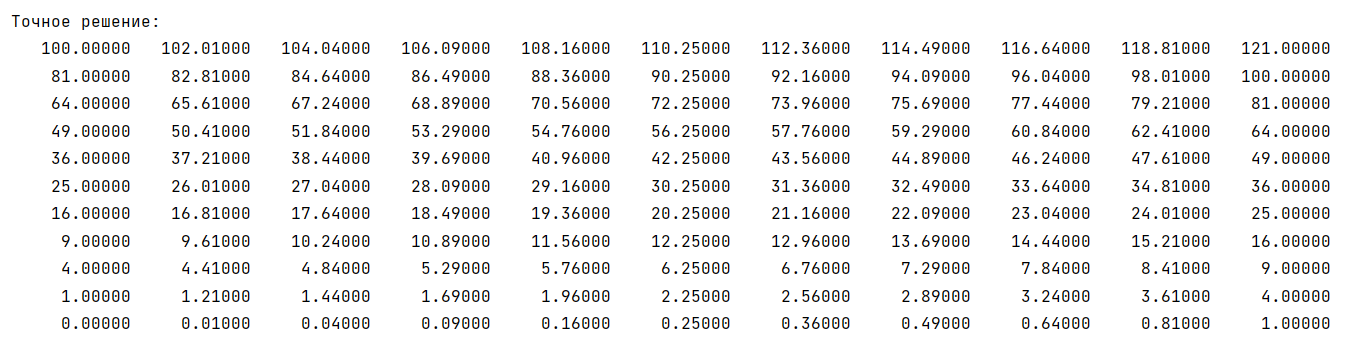
В частности, для :

1. ;
2. ;

Листинг программы

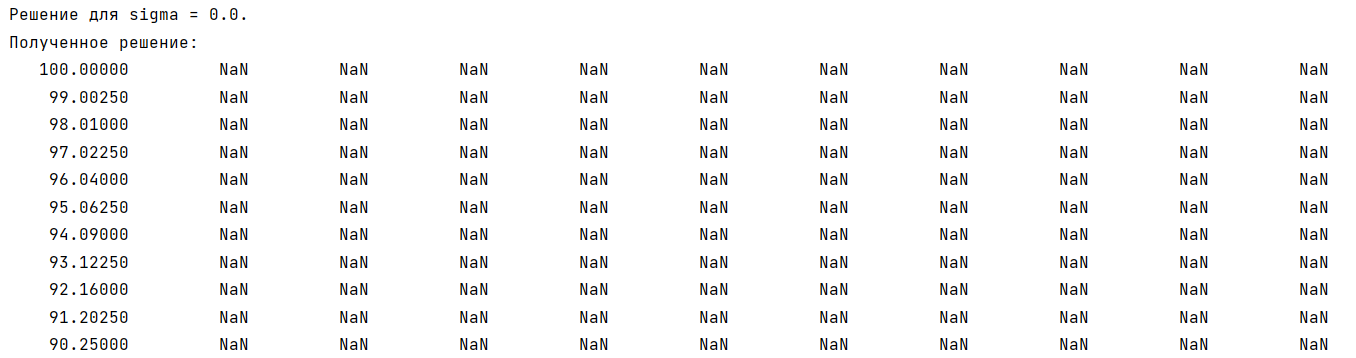
import java.util.\*;  
  
class U {  
 public static double getValue(double x, double t) {  
 return (100. \* Math.*pow*(t, 2.)) + Math.*pow*(x, 2.) + (20. \* x \* t);  
 }  
}  
  
class Mu0 {  
 public static double getValue(double t) {  
 return 100. \* Math.*pow*(t, 2.);  
 }  
}  
  
class U0 {  
 public static double getValue(double x) {  
 return Math.*pow*(x, 2.);  
 }  
}  
  
record ConvectionDiffusionEquationSolver(int n1, int n2, double h, double tau, double[] x, double[] t) {  
 private static final double *A* = -10.;  
  
 public double[][] getY(double sigma) {  
 double[][] res;  
  
 if (sigma == 0.) {  
 res = getYSigma0();  
 } else {  
 res = getYSigmaNot0(sigma);  
 }  
  
 return res;  
 }  
  
 private double[][] getYSigma0() {  
 double[][] res = new double[n2 + 1][n1 + 1];  
 for (double[] row: res) {  
 Arrays.*fill*(row, Double.*NaN*);  
 }  
 setBoundaryValues(res);  
  
 double gamma = - ((*A* \* tau) / h);  
 for (int j = 0; j <= n2 - 1; j++) {  
 for (int i = 1; i <= n1 - j - 1; i++) {  
 res[j + 1][i] = ((1. - gamma) \* res[j][i]) + (gamma \* res[j][i + 1]);  
 }  
 }  
  
 return res;  
 }  
  
 private double[][] getYSigmaNot0(double sigma) {  
 double[][] res = new double[n2 + 1][n1 + 1];  
 for (double[] row: res) {  
 Arrays.*fill*(row, Double.*NaN*);  
 }  
 setBoundaryValues(res);  
  
 double gamma = - ((*A* \* tau) / h);  
 for (int j = 0; j <= n2 - 1; j++) {  
 for (int i = 0; i <= n1 - 1; i++) {  
 res[j + 1][i + 1] = (1. + (gamma \* sigma)) \* res[j + 1][i];  
 res[j + 1][i + 1] -= (1. - (gamma \* (1. - sigma))) \* res[j][i];  
 res[j + 1][i + 1] -= gamma \* (1. - sigma) \* res[j][i + 1];  
 res[j + 1][i + 1] /= gamma \* sigma;  
 }  
 }  
  
 return res;  
 }  
  
 private void setBoundaryValues(double[][] y) {  
 for (int j = 0; j <= n2; j++) {  
 y[j][0] = Mu0.*getValue*(t[j]);  
 }  
  
 for (int i = 1; i <= n1; i++) {  
 y[0][i] = U0.*getValue*(x[i]);  
 }  
 }  
}  
  
class BoundaryValueProblem {  
 private static final double *H* = 0.1;  
 private static final double *TAU* = 0.005;  
 private final int n1;  
 private final int n2;  
 private final double[] x;  
 private final double[] t;  
 private final double[][] u;  
 private double[][] y1;  
 private double[][] res1;  
 private static final double *SIGMA1* = 0.;  
 private double[][] y2;  
 private double[][] res2;  
 private static final double *SIGMA2* = 1.;  
  
 public BoundaryValueProblem() {  
 n1 = (int) (1. / *H*);  
 n2 = (int) (1. / *TAU*);  
  
 x = getX();  
 t = getT();  
 u = getExact(x, t);  
 }  
  
 public void solve() {  
 ConvectionDiffusionEquationSolver cdeSolver = new ConvectionDiffusionEquationSolver(n1, n2, *H*, *TAU*, x, t);  
 y1 = cdeSolver.getY(*SIGMA1*);  
 res1 = getResidual(y1);  
  
 y2 = cdeSolver.getY(*SIGMA2*);  
 res2 = getResidual(y2);  
 }  
  
 public void out() {  
 Formatter fmt = new Formatter();  
  
 fmt.format("\nТочное решение:\n");  
 outY(fmt, u);  
  
 fmt.format("\nРешение для sigma = %.1f.\n", *SIGMA1*);  
 fmt.format("Полученное решение:\n");  
 outY(fmt, y1);  
 fmt.format("Вектор невязок:\n");  
 outRes(fmt, res1);  
  
 fmt.format("\nРешение для sigma = %.1f.\n", *SIGMA2*);  
 fmt.format("Полученное решение:\n");  
 outY(fmt, y2);  
 fmt.format("Вектор невязок:\n");  
 outRes(fmt, res2);  
  
 System.*out*.println(fmt);  
 }  
  
 private void outY(Formatter fmt, double[][] y) {  
 for (int j = n2; j >= 0; j--) {  
 for (int i = 0; i <= n1; i++) {  
 fmt.format("% 14.7f", y[j][i]);  
 }  
  
 fmt.format("\n");  
 }  
 }  
  
 private void outRes(Formatter fmt, double[][] res) {  
 for (int j = n2; j >= 0; j--) {  
 for (int i = 0; i <= n1; i++) {  
 fmt.format("% 14.7E", res[j][i]);  
 }  
  
 fmt.format("\n");  
 }  
 }  
  
 private double[][] getResidual(double[][] y) {  
 double[][] res = new double[n2 + 1][n1 + 1];  
  
 for (int j = 0; j <= n2; j++) {  
 for (int i = 0; i <= n1; i++) {  
 res[j][i] = Math.*abs*(u[j][i] - y[j][i]);  
 }  
 }  
  
 return res;  
 }  
  
 private double[] getX() {  
 double[] res = new double[n1 + 1];  
 for (int i = 0; i <= n1; i++) {  
 res[i] = i \* *H*;  
 }  
  
 return res;  
 }  
  
 private double[] getT() {  
 double[] res = new double[n2 + 1];  
 for (int j = 0; j <= n2; j++) {  
 res[j] = j \* *TAU*;  
 }  
  
 return res;  
 }  
  
 private double[][] getExact(double[] x, double[] t) {  
 double[][] res = new double[n2 + 1][n1 + 1];  
  
 for (int j = 0; j <= n2; j++) {  
 for (int i = 0; i <= n1; i++) {  
 res[j][i] = U.*getValue*(x[i], t[j]);  
 }  
 }  
  
 return res;  
 }  
}  
  
public class Main {  
  
 public static void main(String[] args) {  
 BoundaryValueProblem bvp = new BoundaryValueProblem();  
 bvp.solve();  
 bvp.out();  
 }  
}

Результаты

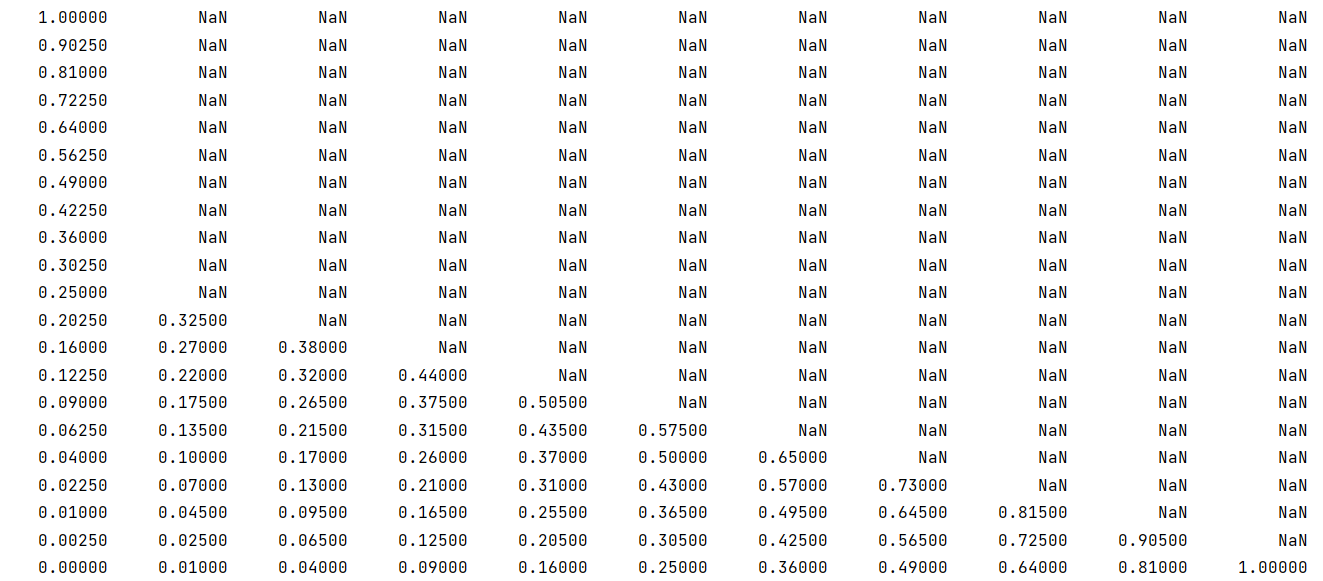


(Вывод с шагами 0.1 и 0.1, т.к. для 0.005 очень много значений)

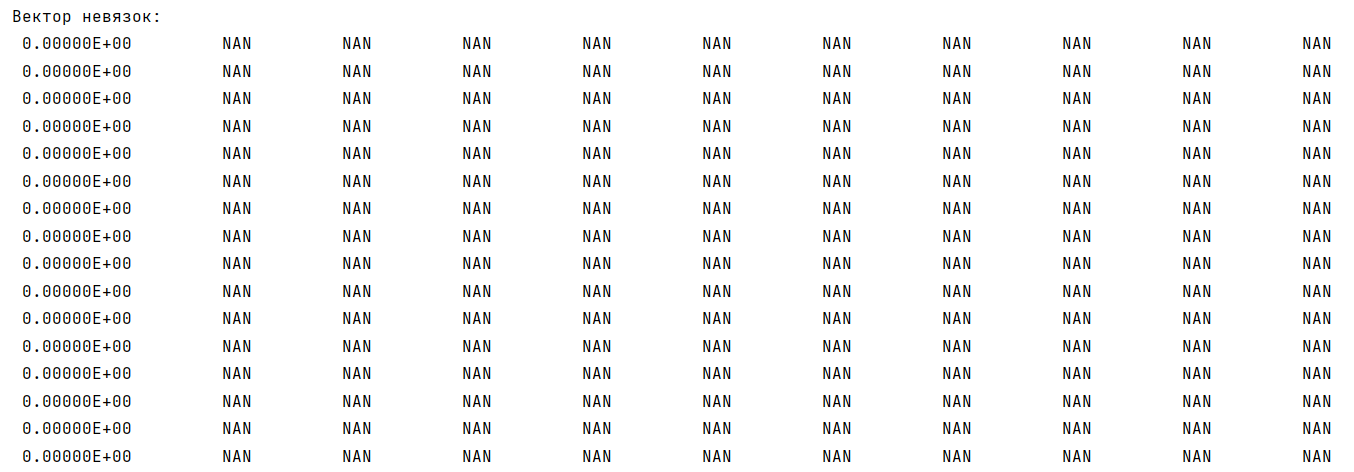
Результат для (возможно получить только часть значений, треугольник):



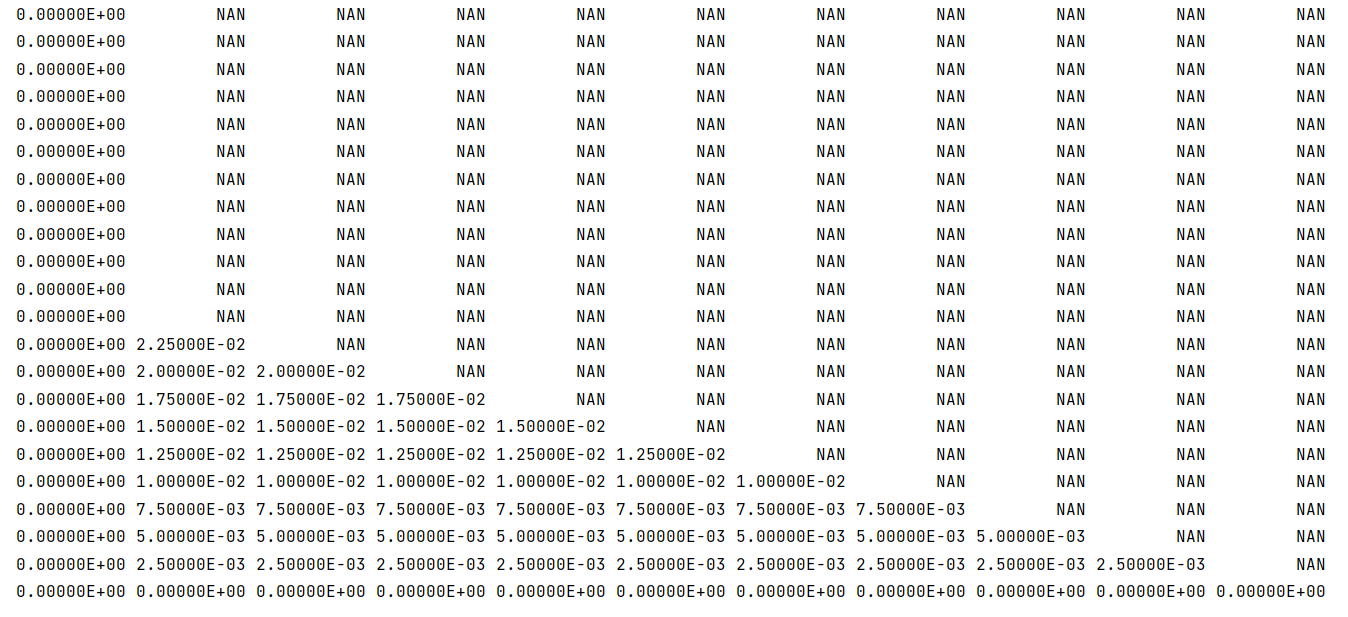
....



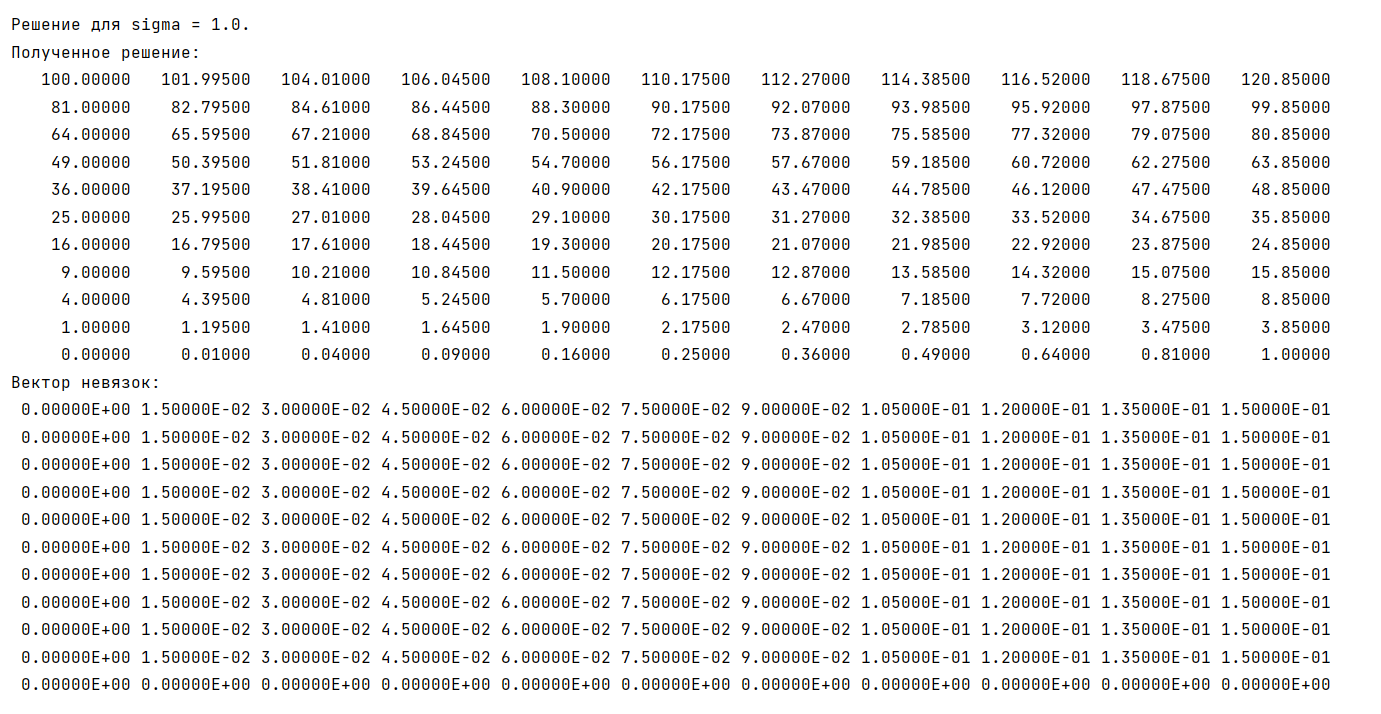
Невязка:



....



Результат для :



(Решение строится для и 0.005, но для удобства просмотра значения выводятся с шагами 0.1 и 0.1)

Выводы

Обе разностные схемы, которые мы использовали, имеют первый порядок по каждой переменной. Соответственно, погрешность аппроксимации составляет в нашем случае и 0.005.

Значения на границе вычисляются точно из начального и граничного условия, поэтому там невязка = 0.

Из вида вектора невязок (разница между “точным” и полученным решением) для решения с можем сказать, что реальная погрешность на внутренних узлах составляет примерно от до , т.е. на порядок-два меньше, чем ожидаемая погрешность – . Результаты вышли лучше ожидаемого, это может быть связано с величиной коэффициента при главном члене погрешности. Также видно, что с увеличением временного слоя погрешность растёт (погрешность наращивается из-за вида шаблона для данного значения ).

Что касается решения с можем сказать, что реальная погрешность на внутренних узлах составляет примерно от до , т.е. примерно соответствует ожидаемой погрешности – . В данном случае погрешность растёт с увеличением шага по переменной (погрешность наращивается из-за вида шаблона для данного значения ).

В итоге получаем, что для нашей задачи явная схема на практике проявила себя на порядок лучше, чем неявная (это не будет выполняться в общем случае, т.к. погрешности у схем одинаковые). Но у нее есть недостаток – требуется специально выбирать шаг для выполнения условия устойчивости схемы.