Московский авиационный институт

(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №6

«Численное решение уравнений с частными производными гиперболического типа»

Вариант №4

Студент: Железнов Д.А.

Группа: М8О-409Б-19

Руководитель: Пивоваров Д.Е.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 09.12.2022

Москва 2022

**Лабораторная работа №6**

Метод конечных разностей для решения задачи гиперболического типа

**Задача**

Используя явную схему крест и неявную схему, решить начально-краевую задачу для дифференциального уравнения гиперболического типа. Аппроксимацию второго начального условия произвести с первым и со вторым порядком. Осуществить реализацию трех вариантов аппроксимации граничных условий, содержащих производные: двухточечная аппроксимация с первым порядком, трехточечная аппроксимация со вторым порядком, двухточечная аппроксимация со вторым порядком. В различные моменты времени вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением . Исследовать зависимость погрешности от сеточных параметров .

**Описание метода**

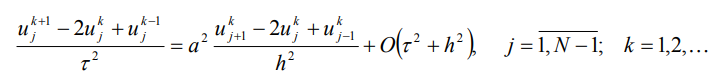
Рассмотрим общий вид уравнения гиперболического типа для третьей начально-краевой задачи:

Изображение выглядит как текст

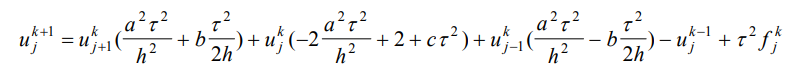
Автоматически созданное описание

Для решения такой задачи применяют метод конечных разностей.

Если аппроксимировать вторую производную по пространству на нижнем временном слое, то получим **явную** конечно-разностную схему:

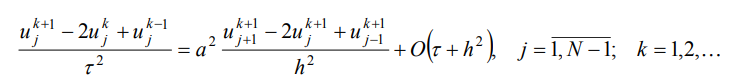


где для каждого j-го уравнения неизвестна только одна величина :



Для устойчивости данной схемы накладывается ограничение на

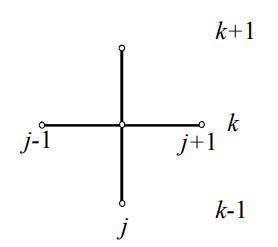
Если аппроксимировать производную по пространству на верхнем временном слое, то получим **неявную** конечно-разностную схему:



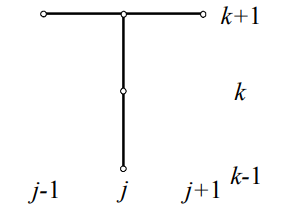
где для нахождения сеточной функции на верхнем временном слое необходимо решать СЛАУ с трёхдиагональной матрицей с помощью метода прогонки.

Шаблоны данных двух схем:

1) Явной схемы



2) Неявной схемы



**Аппроксимация начальных и граничных условий**

**Начальные условия**

В обеих схемах необходимо знать значения на нижних временных слоях. Для k = 1 имеем:

Изображение выглядит как текст

Автоматически созданное описание

Для определения можем воспользоваться аппроксимацией второго начального условия. Тогда получим:

Изображение выглядит как часы

Автоматически созданное описание

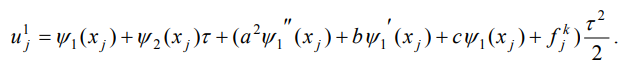


Чтобы повысить порядок аппроксимации второго начального условия раскладывают в ряд Тейлора в окрестности t = 0:

Изображение выглядит как текст, часы

Автоматически созданное описание

Согласно исходному уравнению, получают следующую аппроксимацию:



**Граничные условия**

Рассмотрим 3 способа аппроксимации граничных условий 2Т1П, 3Т2П, 2Т2П.

Двухточечная, с первым порядком:



Трёхточечная, со вторым порядком:

Изображение выглядит как текст

Автоматически созданное описание

При использовании неявной и явно-неявной схемы СЛАУ теряет трёхдиагональность, поэтому сначала необходимо привести СЛАУ к трёхдиагональному виду линейной комбинацией первой строки со второй, предпоследней с последней.

Двухточечная, со вторым порядком:

Для того, чтобы получить эти формулы для начала раскладывают в ряд Тейлора в окрестности x = 0 и в окрестности x = l:

Изображение выглядит как текст

Автоматически созданное описание

Далее берут информацию из исходного уравнения, выражая оттуда вторую производную и подставляя это выражение и получают выражение для первой производной.

**Вариант**

**Изображение выглядит как текст

Автоматически созданное описание**

**Результаты работы программы**

Изображение выглядит как текст

Автоматически созданное описание

**Выводы**

В данной работе реализована явная и неявная конечно-разностные схемы для решения начально-краевой задачи для дифференциального уравнения гиперболического типа. Осуществлена реализацию трех вариантов аппроксимации граничных условий, содержащих производные: двухточечная аппроксимация с первым порядком, трехточечная аппроксимация со вторым порядком, двухточечная аппроксимация со вторым порядком; и двух вариантов начальных условий: аппроксимация с первым и со вторым порядком второго начального условия.

Для сравнения трёх схем вычисляется погрешность как максимальное абсолютное отклонение между точным решением и полученным решением на каждом шаге j для одного момента времени (s = 25).

Погрешность для двух схем при аппроксимации граничных условий со вторым порядком составила порядка 10^(-4). При аппроксимации первым порядком - 10^(-2).

**Приложение. Листинг программы.**

import java.util.ArrayList;  
import java.util.Collections;  
  
public class Lab6 {  
  
 public static void main(String[] args) {  
 double a = 1;  
 int N = 100;  
 double l = 1;  
 double h = l / N;  
 double sigma = 0.25;  
 double tau = Math.*sqrt*(sigma \* h \* h / a);  
  
 int flag = 0;  
 int tem = 1;  
  
 double[][] u = *explicitMethod*(N, h, sigma, tau, flag, tem);  
  
 int s = 25;  
  
 double max = 0;  
 double delta = 0;  
 double temp;  
 for (int i = 0; i < N; i++) {  
 temp = Math.*abs*(u[s][i] - *resultFunc*(h \* i, tau \* s));  
 delta += temp \* temp;  
 if(temp > max){  
 max = temp;  
 }  
 }  
 System.*out*.println("Явный метод 2Т1П");  
 //System.out.println(Math.sqrt(delta) + " " + max);  
 System.*out*.println(max);  
  
 flag = 1;  
 u = *explicitMethod*(N, h, sigma, tau, flag, tem);  
  
 max = 0;  
 delta = 0;  
  
 for (int i = 0; i < N; i++) {  
 temp = Math.*abs*(u[s][i] - *resultFunc*(h \* i, tau \* s));  
 delta += temp \* temp;  
 if(temp > max){  
 max = temp;  
 }  
 }  
 System.*out*.println("\nЯвный метод 3Т2П");  
 //System.out.println(Math.sqrt(delta) + " " + max);  
 System.*out*.println(max);  
  
 flag = 2;  
 u = *explicitMethod*(N, h, sigma, tau, flag, tem);  
  
 max = 0;  
 delta = 0;  
  
 for (int i = 0; i < N; i++) {  
 temp = Math.*abs*(u[s][i] - *resultFunc*(h \* i, tau \* s));  
 delta += temp \* temp;  
 if(temp > max){  
 max = temp;  
 }  
 }  
 System.*out*.println("\nЯвный метод 2Т2П");  
 //System.out.println(Math.sqrt(delta) + " " + max);  
 System.*out*.println(max);  
  
 flag = 0;  
 u = *implicitMethod*(N, h, sigma, tau, flag, tem);  
  
 max = 0;  
 delta = 0;  
  
 for (int i = 0; i < N; i++) {  
 temp = Math.*abs*(u[s][i] - *resultFunc*(h \* i, tau \* s));  
 delta += temp \* temp;  
 if(temp > max){  
 max = temp;  
 }  
 }  
 System.*out*.println("\nНеявный метод 2Т1П");  
 //System.out.println(Math.sqrt(delta) + " " + max);  
 System.*out*.println(max);  
  
 flag = 1;  
 u = *implicitMethod*(N, h, sigma, tau, flag, tem);  
  
 max = 0;  
 delta = 0;  
  
 for (int i = 0; i < N; i++) {  
 temp = Math.*abs*(u[s][i] - *resultFunc*(h \* i, tau \* s));  
 delta += temp \* temp;  
 if(temp > max){  
 max = temp;  
 }  
 }  
 System.*out*.println("\nНеявный метод 3Т2П");  
 //System.out.println(Math.sqrt(delta) + " " + max);  
 System.*out*.println(max);  
  
 flag = 2;  
 u = *implicitMethod*(N, h, sigma, tau, flag, tem);  
  
 max = 0;  
 delta = 0;  
  
 for (int i = 0; i < N; i++) {  
 temp = Math.*abs*(u[s][i] - *resultFunc*(h \* i, tau \* s));  
 delta += temp \* temp;  
 if(temp > max){  
 max = temp;  
 }  
 }  
 System.*out*.println("\nНеявный метод 2Т2П");  
 //System.out.println(Math.sqrt(delta) + " " + max);  
 System.*out*.println(max);  
 }  
  
 static double resultFunc(double x, double t){  
 return Math.*exp*(2\*x) \* Math.*cos*(t);  
 }  
  
 static double[][] explicitMethod(int N, double h, double sigma, double tau, int flag, int tem) {  
 int K = N \* 2;  
 double[][] u = new double[K + 1][N + 1];  
  
 //u[0][j]  
 for (int i = 0; i < u[1].length; i++) {  
 u[0][i] = Math.*exp*(2 \* i \* h);  
 }  
  
 //u[1][j]  
 if(tem == 0) {  
 for (int i = 0; i < u[1].length; i++) {  
 u[1][i] = Math.*exp*(2 \* i \* h);  
 }  
 }  
 else {  
 for (int i = 0; i < u[1].length; i++) {  
 u[1][i] = Math.*exp*(2 \* i \* h) + (4 \* Math.*exp*(2 \* i \* h) - 5 \* u[0][i]) \* tau \* tau / 2;  
 }  
 }  
  
 for (int k = 1; k < K; k++) {  
 for(int j = 1; j < N; j++){  
 u[k + 1][j] = 2 \* u[k][j] - u[k - 1][j] + sigma \* (u[k][j + 1] - 2\*u[k][j] + u[k][j - 1]) - 5 \* tau \* tau \* u[k][j];  
 }  
 if(flag == 0) {  
 u[k + 1][0] = u[k + 1][1] / (2 \* h + 1);  
 u[k + 1][N] = u[k + 1][N - 1] / (1 - 2 \* h);  
 }  
 else if(flag == 1){  
 u[k + 1][0] = (4 \* u[k + 1][1] - u[k + 1][2]) / (3 + 4 \* h) ;  
 u[k + 1][N] = (4 \* u[k + 1][N - 1] - u[k + 1][N - 2]) / (3 - 4 \* h);  
 }  
 else if(flag == 2){  
 u[k + 1][0] = (-(2 \* u[k][0] - u[k - 1][0]) \* h \* h - 2 \* tau \* tau \* u[k + 1][1]) / (-h \* h - 2 \* tau \* tau - 5 \* h \* h \* tau \* tau - 4 \* h \* tau \* tau);  
 u[k + 1][N] = ((2 \* u[k][N] - u[k - 1][N]) \* h \* h + 2 \* tau \* tau \* u[k + 1][N - 1]) / (h \* h + 2 \* tau \* tau + 5 \* h \* h \* tau \* tau - 4 \* h \* tau \* tau);  
 }  
 }  
 return u;  
 }  
  
 static double[][] implicitMethod(int N, double h, double sigma, double tau, int flag, int tem){  
 int K = N \* 2;  
 double[][] u = new double[K + 1][N + 1];  
  
 double[] a\_arr = new double[N + 1];  
 double[] b\_arr = new double[N + 1];  
 double[] c\_arr = new double[N + 1];  
 double[] d\_arr = new double[N + 1];  
  
 //u[0][j]  
 for (int i = 0; i < u[1].length; i++) {  
 u[0][i] = Math.*exp*(2 \* i \* h);  
 }  
  
 //u[1][j]  
 if(tem == 0) {  
 for (int i = 0; i < u[1].length; i++) {  
 u[1][i] = Math.*exp*(2 \* i \* h);  
 }  
 }  
 else {  
 for (int i = 0; i < u[1].length; i++) {  
 u[1][i] = Math.*exp*(2 \* i \* h) + (4 \* Math.*exp*(2 \* i \* h) - 5 \* u[0][i]) \* tau \* tau / 2;  
 }  
 }  
  
 for (int i = 0; i < N + 1; i++) {  
 if(i == 0){  
 if(flag == 0){  
 a\_arr[0] = 0;  
 b\_arr[0] = 1 + 2 \* h;  
 c\_arr[0] = -1;  
 }  
 else if(flag == 1) {  
 a\_arr[0] = 0;  
 b\_arr[0] = 2 + 4 \* h;  
 c\_arr[0] = -4 + (1 + 2 \* sigma + 5 \* tau \* tau) / sigma;  
 }  
 else if(flag == 2){  
 a\_arr[0] = 0;  
 b\_arr[0] = -(h \* h + 2 \* tau \* tau + 5 \* h \* h \* tau \* tau + 4 \* h \* tau \* tau);  
 c\_arr[0] = 2 \* tau \* tau;  
 }  
 }  
 else if(i < N) {  
 a\_arr[i] = sigma;  
 b\_arr[i] = -(1 + 2 \* sigma + 5 \* tau \* tau);  
 c\_arr[i] = sigma;  
 }  
 else{  
 if(flag == 0){  
 a\_arr[i] = -1;  
 b\_arr[i] = 1 - 2 \* h;  
 c\_arr[i] = 0;  
 }  
 else if(flag == 1){  
 a\_arr[i] = -4 + (1 + 2 \* sigma + 5 \* tau \* tau) / sigma;  
 b\_arr[i] = 2 - 4 \* h;  
 c\_arr[i] = 0;  
 }  
 else if(flag == 2){  
 a\_arr[i] = -2 \* tau \* tau;  
 b\_arr[i] = h \* h + 2 \* tau \* tau + 5 \* h \* h \* tau \* tau - 4 \* h \* tau \* tau;  
 c\_arr[i] = 0;  
 }  
 }  
 }  
  
 for(int k = 1; k < K; k++){  
 for(int j = 0; j < N; j++){  
 if(j == 0){  
 if(flag == 0) {  
 d\_arr[0] = 0;  
 }  
 else if(flag == 1){  
 d\_arr[0] = 2 \* u[k][1] / sigma - u[k - 1][1] / sigma;  
 }  
 else if(flag == 2){  
 d\_arr[0] = -h \* h \* (2 \* u[k][0] - u[k - 1][0]);  
 }  
 }  
 else{  
 d\_arr[j] = -2 \* u[k][j] + u[k - 1][j];  
 }  
 }  
 if(flag == 0) {  
 d\_arr[N] = 0;  
 }  
 else if(flag == 1){  
 d\_arr[N] = 2 / sigma \* u[k][N - 1] - 1 / sigma \* u[k - 1][N - 1];  
 }  
 else if(flag == 2){  
 d\_arr[N] = (2 \* u[k][N] - u[k - 1][N]) \* h \* h;  
 }  
  
 ArrayList<Double> res\_progonka = *Progonka*(a\_arr, b\_arr, c\_arr, d\_arr);  
 for (int i = 0; i < N + 1; i++) {  
 u[k + 1][i] = res\_progonka.get(i);  
 }  
 }  
  
 return u;  
 }  
  
 static ArrayList<Double> Progonka(double[] a, double[] b, double[] c, double[] d)  
 {  
 ArrayList<Double> roots = new ArrayList<>();  
 ArrayList<Double> P = new ArrayList<>();  
 ArrayList<Double> Q = new ArrayList<>();  
  
 P.add(-c[0] / b[0]);  
 Q.add(d[0] / b[0]);  
  
 //Прямой ход  
 for(int i = 1; i < a.length; i++)  
 {  
 P.add(-c[i] / (b[i] + a[i] \* P.get(i - 1)));  
 Q.add((d[i] - a[i] \* Q.get(i - 1)) / (b[i] + a[i] \* P.get(i - 1)));  
 }  
  
 Collections.*reverse*(P);  
 Collections.*reverse*(Q);  
  
 //Обратный ход  
 roots.add(Q.get(0));  
 for (int i = 1; i < a.length; i++)  
 {  
 roots.add(P.get(i) \* roots.get(i - 1) + Q.get(i));  
 }  
  
 Collections.*reverse*(roots);  
 return roots;  
 }  
}