**Міністерство освіти і науки України**

**Національний технічний університет України**

**«Київський політехнічний інститут ім. Ігоря Сікорського»**

Інститут Прикладного системного аналізу

(назва факультету, інституту)

Кафедра Системного проектування

(назва кафедри)

Пояснювальна записка

до курсової роботи на тему:

“Паралельне розв’язання диференційного рівняння у частинних похідних за допомогою методів кінцевих різниць”

**Студент групи** ДА-62 Домілевський Олександр Сергійович

(шифр групи) (прізвище, ім’я, по батькові) (підпис)

**Керівник проекту** к.т.н., викл. Яременко В.С.

(вчені ступінь та звання, прізвище, ініціали) (підпис)

Керівник: Виконавець:

Яременко В.С. ст.Домілевський О.С.,

гр. ДА-62

Допущений до захисту Зал. книжка

№ ДА-6212

Захищено із оцінкою

Київ – 2019

Форма No У-6.01   
Затв. наказом УРСР

ННК «ІПСА» НТУУ “Київський Політехнічний Інститут ім. Ігоря Сікорського”

(назва вищого навчального закладу)

Кафедра Системного проектування

Дисципліна Паралельні обчислення

Спеціальність

Курс 3 Група ДА-62 Семестр 6

**ЗАВДАННЯ**

**на курсову роботу студента**

Домілевського Олександра Сергійовича

(прізвище, ім’я, по батькові)

1. Тема курсової роботи

**“Паралельне розв’язання диференційного рівняння у частинних похідних за допомогою методів кінцевих різниць”**

1. Строк здачі студентом закінченого проекту (роботи) до 20.05.2019 р.

Студент Домілевський О.С

(підпис)

Керівник викл. Яременко В.С. (підпис)

“20” травня 2019 р.

# **ЗАВДАННЯ**

Задача № 10.

Знайти розв’язок рівняння:

, де – деякі константи.

Точний розв’язок:

, де , – деякі константи.

# **ТЕОРЕТИЧНІ ВІДОМОСТІ**

## МЕТОД СКІНЧЕННИХ РІЗНИЦЬ

Розглянемо розв’язання лінійного одновимірного рівняння теплопровідності:

з крайовими

та початковою

умовами.

Одним зі способів чисельного розв’язання такого типу рівнянь є апроксимація всіх похідних рівняння їхніми скінченими різницями. Розділимо ділянку простору, використовуючи сітку , а часовий інтервал – сітку . Нехай крок сітки буде рівномірний і в часі, і в просторі. Тоді різницю між двома послідовними точками в просторі позначимо , а в часі – , а чисельною апроксимацією значення буде .

### ЯВНИЙ МЕТОД

Використовуючи праву різницю по часу та центральну різницю по простору для апроксимації похідних, отримаємо рекурентне рівняння:

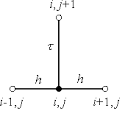


Рис. 1 – Шаблон найбільш поширеного явного методу

Тоді для значення отримуємо:

, де

Таким чином, маючи останнє рекурентне співвідношення та знаючи значення функції в момент часу , можна отримати відповідні значення в момент часу .

Відомо, що явний метод чисельно стійкий і збіжний при , а його похибка пропорційна часовому кроку та квадрату просторового кроку:

# **РОЗВ’ЯЗОК**

Виберемо наступні значення сталих коефіцієнтів:

Тоді рівняння набуває наступного вигляду:

а точний розв’язок:

Знайдемо початкову умову:

та граничні умови:

Застосуємо до рівняння неявну різницеву схему:

Тоді різницеве рівняння матиме наступний вигляд:

звідки

Отримаємо формулу для кожного з (n-1) внутрішніх вузлів поточного часового шару.

# **РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ**

|  |  |  |  |
| --- | --- | --- | --- |
| **Явна різницева схема** | | | |
| Похибки |  |  |  |
| Середня абсолютна | 1.570832218979536E-4 | 1.6642719459362032E-4 | 1.6945222482333866E-4 |
| Максимальна абсолютна | 3.95148197521511E-4 | 3.957427841738692E-4 | 3.958207459746621E-4 |
| Середня відносна, % | 0.012583289297809662 | 0.013331755573917664 | 0.013574069918021067 |
| Максимальна відносна, % | 0.03165462097258323 | 0.031702252254299325 | 0.03170849763582939 |

Таблиця 1 – Похибки для явної різницевої схеми

|  |  |  |
| --- | --- | --- |
|  | **Послідовний** | **Паралельний** |
|  | 4619675 нс | 49613111 нс |

Таблиця 2 – Час виконання у наносекундах

### ПОВЕРХНЯ ЗА ЯВНИМ МЕТОДОМ

На рисунку 1 зображено поверхню, знайдену явним кінцево-різницевим методом із кроками і , а на рисунку 2 – поверхню точного рішення. Поверхні мало відрізняються одна від одної, також можна спостерігати явище накопичення похибки. Графік було побудовано за допомогою пакету Wolfram Mathematica.

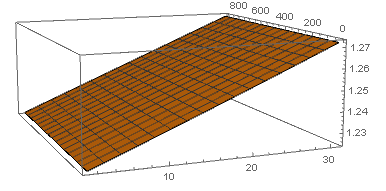


Рис. 1 – Поверхня, знайдена явним методом

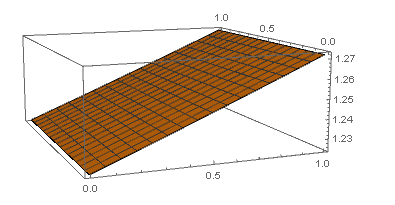


Рис. 2 – Поверхня точного рішення

# **ВИСНОВОК**

В даній роботі було реалізовано паралельне рішення диференційного рівняння в частинних похідних із використанням засобів IntStream.

Також було знайдено похибки для неявного методу при різних кроках.

Було порівняно час на вирішення задачі послідовно і паралельно. Час на паралельну програму виявився більшим, ніж при послідовному пошуку рішень. На мою думку це через те, що в явному методі мало незалежних розрахунків. А також причиною може бути те, що для створення потоків потрібен час.

# **ДОДАТОК 1. ЛІСТИНГ ПРОГРАМИ**

**Main.java**

public class Main {  
 public static void main(String[] args) {  
  
 System.*out*.println("Явний метод");  
 differential diff = new differential();  
  
 long beginTimeSerial = System.*nanoTime*();  
 double[][] approximateResult = new SerialSol(diff).solve();  
 long endTimeSerial = System.*nanoTime*();  
 long serialTime = endTimeSerial - beginTimeSerial;  
 *printMatrix*(approximateResult, diff);  
 System.*out*.println("Время явного решения: " + serialTime);  
 System.*out*.println();  
  
  
 System.*out*.println("Паралельное вычисления");  
 long beginTimeParallel = System.*nanoTime*();  
 double[][] parallelSolution = new ParallelSol(diff).solve();  
 long endTimeParallel = System.*nanoTime*();  
 long parallelTime = endTimeParallel - beginTimeParallel;  
 *printMatrix*(parallelSolution, diff);  
 System.*out*.println("Время паралельного вычисления: " + parallelTime + " ns");  
  
 System.*out*.println("Точное решения:");  
 *printErrors*(approximateResult, *calculateExactResult*(diff), diff);  
 }  
  
 private static double[][] calculateExactResult(differential diff) {  
 double[][] exactMatrix = new double[diff.getTpointsAmount()][diff.getHpointsAmount()];  
 double t = diff.getT0();  
 for (int i = 0; i < diff.getTpointsAmount(); i++) {  
 double x = diff.getX0();  
 for (int j = 0; j < diff.getHpointsAmount(); j++) {  
 exactMatrix[i][j] = diff.calculateTrueSolution(x, t);  
 x += diff.getH();  
 }  
 t += diff.getTau();  
 }  
  
 System.*out*.println();  
  
 for (int i = 0; i < diff.getTpointsAmount(); ++i) {  
 for (int j = 0; j < diff.getHpointsAmount(); ++j) {  
 System.*out*.print(String.*format*("%.5f\t", exactMatrix[i][j]));  
 }  
 System.*out*.println();  
 }  
  
 return exactMatrix;  
 }  
  
 private static void printMatrix(double[][] matrix, differential diff) {  
 for (int i = 0; i < diff.getTpointsAmount(); ++i) {  
 for (int j = 0; j < diff.getHpointsAmount(); ++j) {  
 System.*out*.print(String.*format*("%.5f\t", matrix[i][j]));  
 }  
 System.*out*.println();  
  
 }  
 }  
  
 private static void printErrors(double[][] approximateResult, double[][] exactMatrix, differential diff) {  
 System.*out*.println("Средняя абсолютная ошибка: " + Errors.*mediumAbsoleError*(approximateResult, exactMatrix, diff));  
 System.*out*.println("Максимальная Абсолютная Ошибка: " + Errors.*maxAbsoleError*(approximateResult, exactMatrix, diff));  
 System.*out*.println("Средняя относительная ошибка: " + Errors.*mediumRelativeError*(approximateResult, exactMatrix, diff));  
 System.*out*.println("Максимальная Относительная ошибка: " + Errors.*maxRelativeError*(approximateResult, exactMatrix, diff));  
 }  
  
  
  
  
  
  
  
  
  
  
  
}

**differential.java**

public class differential {  
  
 private final double x0 = 0;  
 private final double x1 = 1;  
 private final double h = 1.0/30;//шаг по х  
 private final double t0 = 0;  
 private final double t1 = 1;  
 private final double tau = 1.0/900;//шаг по времени  
 private final double a = 0.2;  
 private final double b = 0.005;  
 private final double c1 = 1.0;//конст с точного  
 private final double alt = 0.005;//конст с точно  
  
  
 public double getX0() {  
 return x0;  
 }  
  
 public double getH() {  
 return h;  
 }  
  
 public double getT0() {  
 return t0;  
 }  
  
 public double getTau() {  
 return tau;  
 }  
  
 public int getTpointsAmount() {  
 return (int) Math.*ceil*((t1 - t0) / tau) + 1;  
 }  
  
 public int getHpointsAmount() {  
 return (int) Math.*ceil*((x1 - x0) / h) + 1;  
 }  
  
 public double calculateTrueSolution(double x, double t) {  
 return (1/Math.*sqrt*(c1\*Math.*exp*(-2\*alt/a\*(x+alt\*t))-b/(3\*alt)));  
 }  
  
 public double calculateBottom(double x) {  
 double t = 0;  
 return (1/Math.*sqrt*(c1\*Math.*exp*(-2\*alt/a\*(x+alt\*t))-b/(3\*alt)));  
 }  
  
 public double calculateLeft(double t) {  
 double x = 0;  
 return (1/Math.*sqrt*(c1\*Math.*exp*(-2\*alt/a\*(x+alt\*t))-b/(3\*alt)));  
 }  
  
 public double calculateRight(double t) {  
 double x = 1;  
 return (1/Math.*sqrt*(c1\*Math.*exp*(-2\*alt/a\*(x+alt\*t))-b/(3\*alt)));  
 }  
  
 public double calculateApproximateSolution(double wiLeft, double wiCurrent, double wiRight) {  
 return (wiCurrent + (a\*(wiRight -2\* wiCurrent + wiLeft)/Math.*pow*(h,2) + Math.*pow*(wiCurrent,2)\*(wiRight - wiLeft)\*b/2/h)\*tau);//для приблизительного  
 }  
  
  
 }

**SerialSol.java**

public class SerialSol{  
  
 private int tPointsAmount;  
 private int hPointsAmount;  
 private double x0;  
 private double h;  
 private double t0;  
 private double tau;  
 private differential diff;  
  
 public SerialSol(differential diff) {  
 this.diff = diff;  
 this.tPointsAmount = diff.getTpointsAmount();  
 this.hPointsAmount = diff.getHpointsAmount();  
 this.x0 = diff.getX0();  
 this.h = diff.getH();  
 this.t0 = diff.getT0();  
 this.tau = diff.getTau();  
 }  
  
 public double[][] solve() {  
 double t = t0 + tau;  
 double x = x0;  
 double[][] w = new double[tPointsAmount][hPointsAmount];  
 for (int j = 0; j < hPointsAmount; j++, x += h) {  
 w[0][j] = diff.calculateBottom(x);  
 }  
  
 for (int i = 1; i < tPointsAmount; ++i, t += tau) {  
 w[i][0] = diff.calculateLeft(t);  
 for (int j = 1; j < hPointsAmount - 1; j++) {  
 w[i][j] = diff.calculateApproximateSolution(w[i-1][j-1],w[i-1][j],w[i-1][j+1]);  
 }  
 w[i][hPointsAmount - 1] = diff.calculateRight(t);  
 }  
 return w;  
 }  
}

**ParallelSol.java**

import java.util.concurrent.atomic.AtomicInteger;  
import java.util.stream.IntStream;  
  
public class ParallelSol {  
  
 private int tPointsAmount;  
 private int hPointsAmount;  
 private double x0;  
 private double h;  
 private double t0;  
 private double tau;  
 private differential diff;  
  
 public ParallelSol(differential diff) {  
 this.diff = diff;  
 this.tPointsAmount = diff.getTpointsAmount();  
 this.hPointsAmount = diff.getHpointsAmount();  
 this.x0 = diff.getX0();  
 this.h = diff.getH();  
 this.t0 = diff.getT0();  
 this.tau = diff.getTau();  
 }  
  
 public double[][] solve() {  
 double t = t0 + tau;  
 double x = x0;  
 double[][] w = new double[tPointsAmount][hPointsAmount];  
 for (int j = 0; j < hPointsAmount; j++, x += h) {  
 w[0][j] = diff.calculateBottom(x);  
 }  
  
 for (int i = 1; i < tPointsAmount; ++i, t += tau) {  
 w[i][0] = diff.calculateLeft(t);  
  
 AtomicInteger valueI = new AtomicInteger(i);  
  
 IntStream.*range*(1, diff.getHpointsAmount() - 1).parallel().forEach(j -> {  
 int k = valueI.get();  
//нашое  
 w[k][j] = diff.calculateApproximateSolution(w[k - 1][j - 1], w[k - 1][j], w[k - 1][j + 1]);  
  
 });  
  
 w[i][hPointsAmount - 1] = diff.calculateRight(t);  
 }  
 return w;  
 }  
  
}

**Errors.java**

public class Errors {  
 public static double mediumAbsoleError(double[][] approximateResult, double[][] exactMatrix, differential diff) {  
 double[][] matrix = new double[diff.getTpointsAmount()][diff.getHpointsAmount()];  
 double error = 0;  
 double errorResult = 0;  
  
 for (int i = 0; i < diff.getTpointsAmount(); i++) {  
 for (int j = 0; j < diff.getHpointsAmount(); j++) {  
 matrix[i][j] = Math.*abs*(approximateResult[i][j] - exactMatrix[i][j]);  
 }  
 }  
  
 for (int i = 0; i < diff.getTpointsAmount(); i++) {  
 for (int j = 0; j < diff.getHpointsAmount(); j++) {  
 error += matrix[i][j];  
 }  
 }  
 errorResult = error / (diff.getTpointsAmount() \* diff.getHpointsAmount());  
  
 return errorResult;  
 }  
 public static double maxAbsoleError(double[][] approximateResult, double[][] exactMatrix, differential diff) {  
 double[][] matrix = new double[diff.getTpointsAmount()][diff.getHpointsAmount()];  
 double maxError = 0;  
  
 for (int i = 0; i < diff.getTpointsAmount(); i++) {  
 for (int j = 0; j < diff.getHpointsAmount(); j++) {  
 matrix[i][j] = Math.*abs*(approximateResult[i][j] - exactMatrix[i][j]);  
 }  
 }  
  
 maxError = matrix[0][0];  
 for (int i = 0; i < diff.getTpointsAmount(); i++) {  
 for (int j = 0; j < diff.getHpointsAmount(); j++) {  
 if (maxError < matrix[i][j]) {  
 maxError = matrix[i][j];  
 }  
 }  
 }  
  
  
 return maxError;  
 }  
 public static double mediumRelativeError(double[][] approximateResult, double[][] exactMatrix, differential diff) {  
 double[][] matrix = new double[diff.getTpointsAmount()][diff.getHpointsAmount()];  
 double error = 0;  
 double errorResult = 0;  
  
 for (int i = 0; i < diff.getTpointsAmount(); i++) {  
 for (int j = 0; j < diff.getHpointsAmount(); j++) {  
 if (exactMatrix[i][j] == 0) {  
 matrix[i][j] = 0;  
 } else {  
 matrix[i][j] = (Math.*abs*(approximateResult[i][j] - exactMatrix[i][j]) / exactMatrix[i][j]) \* 100;  
 }  
 error += matrix[i][j];  
 }  
 }  
  
  
 errorResult = error / (diff.getTpointsAmount() \* diff.getHpointsAmount());  
  
 return errorResult;  
 }  
 public static double maxRelativeError(double[][] approximateResult, double[][] exactMatrix, differential diff) {  
 double[][] matrix = new double[diff.getTpointsAmount()][diff.getHpointsAmount()];  
 double maxError = 0;  
  
 for (int i = 0; i < diff.getTpointsAmount(); i++) {  
 for (int j = 0; j < diff.getHpointsAmount(); j++) {  
 if (exactMatrix[i][j] == 0) {  
 matrix[i][j] = 0;  
 } else {  
 matrix[i][j] = (Math.*abs*(approximateResult[i][j] - exactMatrix[i][j]) / exactMatrix[i][j]) \* 100;  
 }  
 }  
 }  
  
 maxError = matrix[0][0];  
 for (int i = 0; i < diff.getTpointsAmount(); i++) {  
 for (int j = 0; j < diff.getHpointsAmount(); j++) {  
 if (maxError < matrix[i][j]) {  
 maxError = matrix[i][j];  
 }  
 }  
 }  
  
  
 return maxError;  
 }  
  
}

Посилання на GitHub:

[**https://github.com/DomilevskiyOleksand/kurs**](https://github.com/DomilevskiyOleksand/kurs)