**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-61Стасевич Олексій Михайлович

(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. Стасевич О.М.,

гр. ДА-62

Допущений до захисту Зал. книжка

№ ДА-6215

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 6.050101

Курс 3 Група ДА-61 Семестр 6

**ЗАВДАННЯ**

**на курсову роботу студента**

Стсевича Олексія Михайловича

(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 Кеба С. П.

(підпис)

Керівник ас. Яременко В.С. (підпис)

“20”травня 2019 р.

# **ЗАВДАННЯ**

Задача № 11.

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:

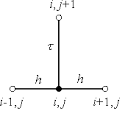


Рис. 1 – Шаблон найбільш поширеного явного методу

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:

# **РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ**

|  |  |  |  |
| --- | --- | --- | --- |
| **Явна різницева схема** | | | |
| Похибки |  |  |  |
| Середня абсолютна | 0.002331077262737 | 0.0024513866772350747 | 0.0024924833916622085 |
| Максимальна абсолютна | 0.005863394359460 | 0.00582857170863349 | 0.005821635980744588 |
| Середня відносна, % | 0.024931652320576574 | 0.02621825818142405 | 0.026657772140037568 |
| Максимальна відносна, % | 0.062712303452855 | 0.06233985559896599 | 0.06226567408474915 |

Таблиця 1 – Похибки для явної різницевої схеми

|  |  |  |
| --- | --- | --- |
|  | **Послідовний** | **Паралельний** |
|  | 9593184нс | 163206441нс |

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 WolframMathematica.



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.

Так сталося з двох причин: на створення потоків необхідний час(менш значна); через особливість операцій(формул) явного метода більшість обчислень не є не залежними і потоки змушені чекати на виконання відповідних операцій в інших потоках.

# **ДОДАТОК 1. ЛІСТИНГ ПРОГРАМИ**

**Main.java**

public class Main {  
  
 public static void main(String[] args) {  
 Equation equation = new Equation();  
 long startTimeSerial = System.*nanoTime*();  
 double[][] serialMethod = new SerialMethod(equation).solve();  
 long endTimeSerial = System.*nanoTime*();  
 long executeTimeSerial = endTimeSerial - startTimeSerial;  
  
 long startTimeParallel = System.*nanoTime*();  
 double[][] parallelMethod = new ParallelMethod(equation).solve();  
 long endTimeParallel = System.*nanoTime*();  
 long executeTimeParallel = endTimeParallel - startTimeParallel;  
  
 *printResult*(executeTimeSerial, executeTimeParallel, equation, serialMethod, parallelMethod, *calculateExactResult*(equation));  
 }  
  
 private static double[][] calculateExactResult(Equation equation) {  
 double trueMatrix[][] = new double[equation.getTPointsQuantity()][equation.getHPointsQuantity()];  
 double t = equation.getT0();  
 for (int i = 0; i < equation.getTPointsQuantity(); i++) {  
 double x = equation.getX0();  
 for (int j = 0; j < equation.getHPointsQuantity(); j++) {  
 trueMatrix[i][j] = equation.calculateTrueSolution(x, t);  
 x += equation.getH();  
 }  
 t += equation.getTau();  
 }  
 return trueMatrix;  
 }

private static void printResult(long executeTimeSerial, long executeTimeParallel,  
 Equation equation, double[][] serialMethod, double[][] parallelMethod, double[][] trueMatrix) {  
  
 System.*out*.println("\nSerial Solution:");  
 equation.printMatrix(serialMethod);  
  
 System.*out*.println("\nParallel Result:");  
 equation.printMatrix(parallelMethod);  
  
 System.*out*.println("\nExact Result:");  
 equation.printMatrix(trueMatrix);  
  
 System.*out*.println();  
 System.*out*.println("Serial Time: " + executeTimeSerial + " ns");  
 System.*out*.println("Parallel Time: " + executeTimeParallel + " ns");  
 System.*out*.println();  
  
 Errors error = new Errors();  
 System.*out*.println("Average absolute error: " + error.*calculateError*(equation, serialMethod, trueMatrix, 1));  
 System.*out*.println("Max absolute error: " + error.*calculateError*(equation, serialMethod, trueMatrix, 2));  
 System.*out*.println("Average relative error: " + error.*calculateError*(equation, serialMethod, trueMatrix, 3));  
 System.*out*.println("Max relative error: " + error.*calculateError*(equation, serialMethod, trueMatrix, 4));  
 }  
}

**Equation.java**

public class Equation {  
  
 private final double x0 = 0;  
 private final double x1 = 1;  
 private final double h = 1.0/3;  
 private final double t0 = 0;  
 private final double t1 = 1;  
 private final double tau = 1.0/5;  
 private final double a = 0.2;  
 private final double b = 0.005;  
 private final double c1 = 1.0;  
 private final double alt = 0.005;  
  
 public double getX0() {  
 return x0;  
 }  
  
 public double getX1() {  
 return x1;  
 }  
  
 public double getH() {  
 return h;  
 }  
  
 public double getT0() {  
 return t0;  
 }  
  
 public double getT1() {  
 return t1;  
 }  
  
 public double getTau() {  
 return tau;  
 }  
  
 public int getTPointsQuantity() {  
 return (int) Math.*ceil*((t1 - t0) / tau) + 1;  
 }  
  
 public int getHPointsQuantity() {  
 return (int) Math.*ceil*((x1 - x0) / h) + 1;  
 }  
  
 public double calculateTrueSolution(double x, double t) {  
 return (1/Math.*pow*(c1\*Math.*exp*(alt/a/-2\*(x+alt\*t))-2\*b/(3\*alt),2));  
  
 }  
  
 public double calculateFirstRow(double x) {  
 double t = 0;  
 return (1/Math.*pow*(c1\*Math.*exp*(alt/a/-2\*(x+alt\*t))-2\*b/(3\*alt),2));  
 }  
  
 public double calculateFirstColumn(double t) {  
 double x = 0;  
 return (1/Math.*pow*(c1\*Math.*exp*(alt/a/-2\*(x+alt\*t))-2\*b/(3\*alt),2));  
 }  
  
 public double calculateLastColumn(double t) {  
 double x = 1;  
 return (1/Math.*pow*(c1\*Math.*exp*(alt/a/-2\*(x+alt\*t))-2\*b/(3\*alt),2));  
 }  
  
 public double calculateApproximateSolution(double wLeft, double wCurrent , double wRight) {  
 return (wCurrent + (a\*(wRight -2\* wCurrent + wLeft)/Math.*pow*(h,2) + Math.*sqrt*(wCurrent)\*(wRight - wLeft)\*b/2\*h)\*tau);  
 }  
  
 public void printMatrix(double[][] matrix) {  
 for (int i = 0; i < matrix.length; ++i) {  
 for (int j = 0; j < matrix[i].length; ++j) {  
 System.*out*.print(String.*format*("%.7f\t", matrix[i][j]));  
 }  
 System.*out*.println();  
 }  
 }  
}

**SerialMethod.java**

public class SerialMethod {  
  
  
 private int TPointsQuantity;  
 private int HPointsQuantity;  
 private double x0;  
 private double h;  
 private double t0;  
 private double tau;  
 private Equation equation;  
  
 public SerialMethod(Equation equation) {  
 this.equation = equation;  
 this.TPointsQuantity = equation.getTPointsQuantity();  
 this.HPointsQuantity = equation.getHPointsQuantity();  
 this.x0 = equation.getX0();  
 this.h = equation.getH();  
 this.t0 = equation.getT0();  
 this.tau = equation.getTau();  
 }  
  
 public double[][] solve() {  
 double t = t0 + tau;  
 double x = x0;  
 double w[][] = new double[TPointsQuantity][HPointsQuantity];  
 for (int j = 0; j < HPointsQuantity; j++, x += h) {  
 w[0][j] = equation.calculateFirstRow(x);  
 }  
  
 for (int i = 1; i < TPointsQuantity; ++i, t += tau) {  
 w[i][0] = equation.calculateFirstColumn(t);  
 for (int j = 1; j < HPointsQuantity - 1; j++) {  
 w[i][j] = equation.calculateApproximateSolution(w[i-1][j-1],w[i-1][j],w[i-1][j+1]);  
 }  
 w[i][HPointsQuantity - 1] = equation.calculateLastColumn(t);  
 }  
 return w;  
 }  
}

**ParallelMethod.java**

import java.util.concurrent.atomic.AtomicInteger;  
import java.util.stream.IntStream;  
  
public class ParallelMethod {  
  
 private int TPointsQuantity;  
 private int HPointsQuantity;  
 private double x0;  
 private double h;  
 private double t0;  
 private double tau;  
 private Equation equation;  
  
 public ParallelMethod(Equation equation) {//конструктор  
 this.equation = equation;  
 this.TPointsQuantity = equation.getTPointsQuantity();  
 this.HPointsQuantity = equation.getHPointsQuantity();  
 this.x0 = equation.getX0();  
 this.h = equation.getH();  
 this.t0 = equation.getT0();  
 this.tau = equation.getTau();  
 }  
  
 public double[][] solve() {  
 double x = x0;  
 double t = t0 + tau;  
  
 double w[][] = new double[TPointsQuantity][HPointsQuantity];  
 for (int j = 0; j < HPointsQuantity; j++, x += h) {  
 w[0][j] = equation.calculateFirstRow(x);  
 }  
  
 for (int i = 1; i < TPointsQuantity; ++i, t += tau) {  
 w[i][0] = equation.calculateFirstColumn(t);  
 AtomicInteger atom = new AtomicInteger(i);  
 IntStream.*range*(1, equation.getHPointsQuantity()-1).parallel().forEach(j -> {  
 int m=atom.get();  
 w[m][j] = equation.calculateApproximateSolution(w[m-1][j-1],w[m-1][j],w[m-1][j+1]);  
 });  
 w[i][HPointsQuantity - 1] = equation.calculateLastColumn(t);  
 }  
 return w;  
 }  
}

**Errors.java**

public class Errors {  
  
 public static double calculateError(Equation equation, double[][] serialMethod, double[][] trueMatrix, int flag) {  
 double[][] errorMatrix = new double[equation.getTPointsQuantity()][equation.getHPointsQuantity()];  
 double error = 0.0;  
  
 switch (flag){  
 case 1:  
 for (int i = 0; i < equation.getTPointsQuantity(); i++) {  
 for (int j = 0; j < equation.getHPointsQuantity(); j++) {  
 errorMatrix[i][j] = Math.*abs*(serialMethod[i][j] - trueMatrix[i][j]);  
 error += errorMatrix[i][j];  
 }  
 }  
 error = error / (equation.getTPointsQuantity() \* equation.getHPointsQuantity());  
 break;  
 case 2:  
 for (int i = 0; i < equation.getTPointsQuantity(); i++) {  
 for (int j = 0; j < equation.getHPointsQuantity(); j++) {  
 errorMatrix[i][j] = Math.*abs*(serialMethod[i][j] - trueMatrix[i][j]);  
 if (error < errorMatrix[i][j]) {  
 error = errorMatrix[i][j];  
 }  
 }  
 }  
 break;  
 case 3:  
 for (int i = 0; i < equation.getTPointsQuantity(); i++) {  
 for (int j = 0; j < equation.getHPointsQuantity(); j++) {  
 errorMatrix[i][j] = 100 \* (Math.*abs*(serialMethod[i][j] - trueMatrix[i][j])) / trueMatrix[i][j];  
 error += errorMatrix[i][j];  
 }  
 }  
 error = error / (equation.getTPointsQuantity() \* equation.getHPointsQuantity());  
 break;  
 case 4:  
 for (int i = 0; i < equation.getTPointsQuantity(); i++) {  
 for (int j = 0; j < equation.getHPointsQuantity(); j++) {  
 errorMatrix[i][j] = 100 \* (Math.*abs*(serialMethod[i][j] - trueMatrix[i][j])) / trueMatrix[i][j];  
 if (error < errorMatrix[i][j]) {  
 error = errorMatrix[i][j];  
 }  
 }  
 }  
 break;  
 }  
 return error;  
 }  
  
}

# **Посилання на GitHub**

<https://github.com/KeS334/DiffurKesProject>