**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-61 Стасевич 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-61

Допущений до захисту Зал. книжка

№ ДА-6130

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 Стасевич О.М.

(підпис)

Керівник викл. Яременко В.С. (підпис)

“20” травня 2019 р.

# **ЗАВДАННЯ**

Задача № 22.

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:

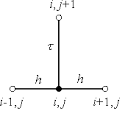


Рис. 1 – Шаблон найбільш поширеного явного методу

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:

де

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 | 6.832726391161041E-6 | 3.436812823637681E-5 | 8.331163598852603E-5 |
| Максимальна абсолютна | 7.422527286529701E-5 | 3.942339129223982E-4 | 9.711908220093601E-4 |
| Середня відносна, % | 0.004734606622131221 | 0.022482610312788386 | 0.053606464703269525 |
| Максимальна відносна, % | 0.06090905268382784 | 0.2808662527984319 | 0.6617866987639445 |

Таблиця 1 – Похибки для явної різницевої схеми

|  |  |  |
| --- | --- | --- |
|  | **Послідовний** | **Паралельний** |
|  | 3771114 нс | 106557150 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.

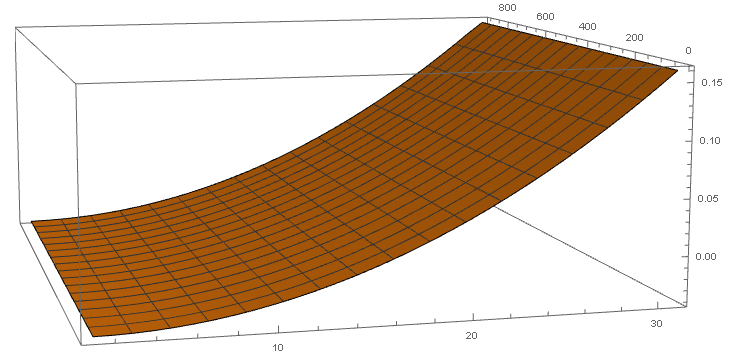


Рис. 1 – Поверхня, знайдена явним методом

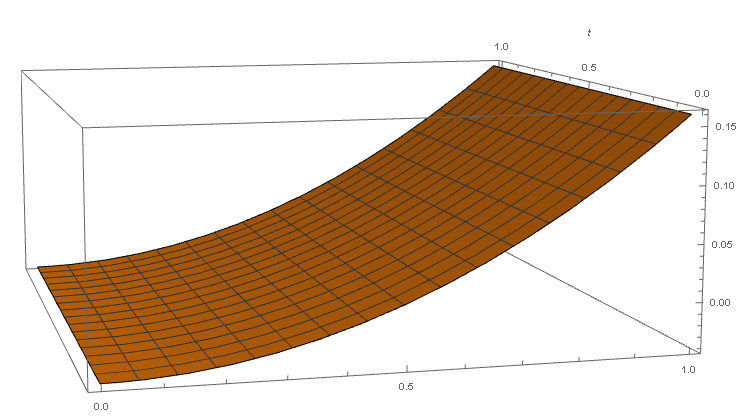


Рис. 2 – Поверхня точного рішення

# **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**

**package** Main;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

System.***out***.println("ПОСЛЕДОВАТЕЛЬНОЕ РЕШЕНИЕ:");

Diffur diffur = **new** Diffur();

**long** startTimeSerial = System.*nanoTime*();

**double**[][] serialSolve = **new** SerialSolve(diffur).solve();

**long** endTimeSerial = System.*nanoTime*();

**long** executeTimeSerial = endTimeSerial - startTimeSerial;

System.***out***.println("Приближённый результат:");

diffur.printMatrix(serialSolve);

*printResult*(executeTimeSerial, diffur, serialSolve, *calculateExactResult*(diffur));

System.***out***.println();

System.***out***.println("\nПАРАЛЛЕЛЬНОЕ РЕШЕНИЕ:");

**long** startTimeParallel = System.*nanoTime*();

**double**[][] parallelSolve = **new** ParallelSolve(diffur).solve();

**long** endTimeParallel = System.*nanoTime*();

**long** executeTimeParallel = endTimeParallel - startTimeParallel;

// for (int i = 0; i < diffur.getTPointsQuantity(); ++i) {

// for (int j = 0; j < diffur.getHPointsQuantity(); ++j) {

// parallelSolve[i][j] = Math.abs(parallelSolve[i][j] - serialSolve[i][j]);

// System.out.print(String.format("%.7f\t", parallelSolve[i][j]));

// }

// System.out.println();

// }

System.***out***.println("\nВремя выполнения параллельного решения: " + executeTimeParallel + " нс");

}

**private** **static** **double**[][] calculateExactResult(Diffur diffur) {

**double** trueMatrix[][] = **new** **double**[diffur.getTPointsQuantity()][diffur.getHPointsQuantity()];

**double** t = diffur.getT0();

**for** (**int** i = 0; i < diffur.getTPointsQuantity(); i++) {

**double** x = diffur.getX0();

**for** (**int** j = 0; j < diffur.getHPointsQuantity(); j++) {

trueMatrix[i][j] = diffur.calculateTrueSolution(x, t);

x += diffur.getH();

}

t += diffur.getTau();

}

System.***out***.println("\nТочный резульат:");

diffur.printMatrix(trueMatrix);

**return** trueMatrix;

}

**private** **static** **void** printResult(**long** executeTimeSerial, Diffur diffur, **double**[][] serialSolve, **double**[][] trueMatrix) {

System.***out***.println("\nВремя выполнения последовательного решения: " + executeTimeSerial + " нс");

System.***out***.println("Средняя абсолютная погрешность: " + *calculateError*(diffur, serialSolve, trueMatrix, 1));

System.***out***.println("Максимальная абсолютная погрешность: " + *calculateError*(diffur, serialSolve, trueMatrix, 2));

System.***out***.println("Средняя относительная погрешность: " + *calculateError*(diffur, serialSolve, trueMatrix, 3));

System.***out***.println("Максимальная относительная погрешность: " + *calculateError*(diffur, serialSolve, trueMatrix, 4));

}

**private** **static** **double** calculateError(Diffur diffur, **double**[][] serialSolve, **double**[][] trueMatrix, **int** flag) {

**double**[][] errorMatrix = **new** **double**[diffur.getTPointsQuantity()][diffur.getHPointsQuantity()];

**double** error = 0.0;

**if** (flag == 1 || flag == 2) {

**for** (**int** i = 0; i < diffur.getTPointsQuantity(); i++) {

**for** (**int** j = 0; j < diffur.getHPointsQuantity(); j++) {

errorMatrix[i][j] = Math.*abs*(serialSolve[i][j] - trueMatrix[i][j]);

}

}

**if** (flag == 1) {

**for** (**int** i = 0; i < diffur.getTPointsQuantity(); i++) {

**for** (**int** j = 0; j < diffur.getHPointsQuantity(); j++) {

error += errorMatrix[i][j];

}

}

error = error / (diffur.getTPointsQuantity() \* diffur.getHPointsQuantity());

} **else** **if** (flag == 2) {

error = errorMatrix[0][0];

**for** (**int** i = 0; i < diffur.getTPointsQuantity(); i++) {

**for** (**int** j = 0; j < diffur.getHPointsQuantity(); j++) {

**if** (error < errorMatrix[i][j]) {

error = errorMatrix[i][j];

}

}

}

}

} **else** **if** (flag == 3 || flag == 4) {

**for** (**int** i = 0; i < diffur.getTPointsQuantity(); i++) {

**for** (**int** j = 0; j < diffur.getHPointsQuantity(); j++) {

errorMatrix[i][j] = 100 \* (Math.*abs*(serialSolve[i][j] - trueMatrix[i][j])) / trueMatrix[i][j];

}

}

**if** (flag == 3) {

**for** (**int** i = 0; i < diffur.getTPointsQuantity(); i++) {

**for** (**int** j = 0; j < diffur.getHPointsQuantity(); j++) {

error += errorMatrix[i][j];

}

}

error = error / (diffur.getTPointsQuantity() \* diffur.getHPointsQuantity());

} **else** **if** (flag == 4) {

error = errorMatrix[0][0];

**for** (**int** i = 0; i < diffur.getTPointsQuantity(); i++) {

**for** (**int** j = 0; j < diffur.getHPointsQuantity(); j++) {

**if** (error < errorMatrix[i][j]) {

error = errorMatrix[i][j];

}

}

}

}

}

**return** error;

}

}

**Diffur.java**

**package** Main;

**public** **class** Diffur {

**private** **final** **double** x0 = 0;

**private** **final** **double** x1 = 1;

**private** **final** **double** h = 1.0/30;

**private** **final** **double** t0 = 0;

**private** **final** **double** t1 = 1;

**private** **final** **double** tau = 1.0/900;

**private** **final** **double** a = 0.00005;

**private** **final** **double** b = 0.00005;

**private** **final** **double** A = 1.0;

**private** **final** **double** B = 5.0;

**private** **double** sigma = tau / (h \* h);

**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** (Math.*pow*(x, 2)/(B-6\*a\*t) + A\*Math.*pow*(B-6\*a\*t, -1.0/3) - (b\*(B-6\*a\*t))/(8\*a));

}

**public** **double** calculateBottomBorder(**double** x) {

**return** (Math.*pow*(x, 2)/B + A\*Math.*pow*(B, -1.0/3) - (b\*B)/(8\*a));

}

**public** **double** calculateLeftBorder(**double** t) {

**return** (A\*Math.*pow*(B-6\*a\*t, -1.0/3) - (b\*(B-6\*a\*t))/(8\*a));

}

**public** **double** calculateRightBorder(**double** t) {

**return** (1.0/(B-6\*a\*t) + A\*Math.*pow*(B-6\*a\*t, -1.0/3) - (b\*(B-6\*a\*t))/(8\*a));

}

**public** **double** calculateApproximateSolution(**double** wLeft, **double** wCurrent , **double** wRight) {

**return** (wCurrent + a\*sigma\*(0.25\*Math.*pow*(wRight - wLeft, 2) + wCurrent\*(wRight - wCurrent + wLeft))+b\*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();

}

}

}

**SerialSolve.java**

**package** Main;

**public** **class** SerialSolve {

**private** **int** TPointsQuantity;

**private** **int** HPointsQuantity;

**private** **double** x0;

**private** **double** h;

**private** **double** t0;

**private** **double** tau;

**private** Diffur diffur;

**public** SerialSolve(Diffur diffur) {

**this**.diffur = diffur;

**this**.TPointsQuantity = diffur.getTPointsQuantity();

**this**.HPointsQuantity = diffur.getHPointsQuantity();

**this**.x0 = diffur.getX0();

**this**.h = diffur.getH();

**this**.t0 = diffur.getT0();

**this**.tau = diffur.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] = diffur.calculateBottomBorder(x);

}

**for** (**int** i = 1; i < TPointsQuantity; ++i, t += tau) {

w[i][0] = diffur.calculateLeftBorder(t);

**for** (**int** j = 1; j < HPointsQuantity - 1; j++) {

w[i][j] = diffur.calculateApproximateSolution(w[i-1][j-1],w[i-1][j],w[i-1][j+1]);

}

w[i][HPointsQuantity - 1] = diffur.calculateRightBorder(t);

}

**return** w;

}

}

**ParallelSolve.java**

**package** Main;

**import** java.util.concurrent.atomic.AtomicInteger;

**import** java.util.stream.IntStream;

**public** **class** ParallelSolve {

**private** **int** TPointsQuantity;

**private** **int** HPointsQuantity;

**private** **double** x0;

**private** **double** h;

**private** **double** t0;

**private** **double** tau;

**private** Diffur diffur;

**public** ParallelSolve(Diffur diffur) {

**this**.diffur = diffur;

**this**.TPointsQuantity = diffur.getTPointsQuantity();

**this**.HPointsQuantity = diffur.getHPointsQuantity();

**this**.x0 = diffur.getX0();

**this**.h = diffur.getH();

**this**.t0 = diffur.getT0();

**this**.tau = diffur.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] = diffur.calculateBottomBorder(x);

}

**for** (**int** i = 1; i < TPointsQuantity; ++i, t += tau) {

w[i][0] = diffur.calculateLeftBorder(t);

AtomicInteger ai = **new** AtomicInteger(i);

IntStream.*range*(1, diffur.getHPointsQuantity()-1).parallel().forEach(j -> {

**int** m=ai.get();

w[m][j] = diffur.calculateApproximateSolution(w[m-1][j-1],w[m-1][j],w[m-1][j+1]);

});

w[i][HPointsQuantity - 1] = diffur.calculateRightBorder(t);

}

**return** w;

}

}