**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 Терещенко Олександр Ігорович

(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. Терещенко О.І.,

гр. ДА-62

Допущений до захисту Зал. книжка

№ ДА-6225

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-62 Семестр 6

**ЗАВДАННЯ**

**на курсову роботу студента**

Терещенко Олександр Ігорович

(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 Терещенко О.І.

(підпис)

Керівник викл. Яременко В.С. (підпис)

“20” травня 2019 р.

# **ЗАВДАННЯ**

Задача № 25.

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:

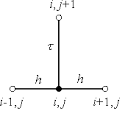


Рис. 1 – Шаблон найбільш поширеного явного методу

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**

Застосуємо формулу диференціалу добутку:

Виберемо наступні значення сталих коефіцієнтів:

Тоді рівняння набуває наступного вигляду:

а точний розв’язок:

, де

Знайдемо початкову умову:

та граничні умови:

Застосуємо до рівняння явну 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 | 5.4299182442407005E-6 | 6.211983774521015E-6 | 6.489820143373901E-6 |
| Максимальна абсолютна | 3.587797545301363E-5 | 4.0314998926427315E-5 | 4.136102049911994E-5 |
| Середня відносна, % | 7.120982857082075E-4 | 8.000571700054988E-4 | 8.30984732861824E-4 |
| Максимальна відносна, % | 0.003949328998140286 | 0.004213176982945489 | 0.00424740672493456 |

Таблиця 1 – Похибки для явної різницевої схеми

|  |  |  |
| --- | --- | --- |
|  | **Послідовний** | **Паралельний** |
|  | 9209676 нс | 109557685 нс |
|  | 6223405 нс | 86509793 нс |
|  | 2261727 нс | 72136267 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.

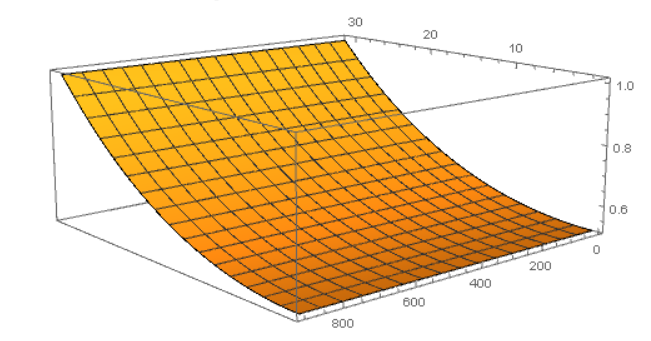


Рис. 1 – Поверхня, знайдена явним методом

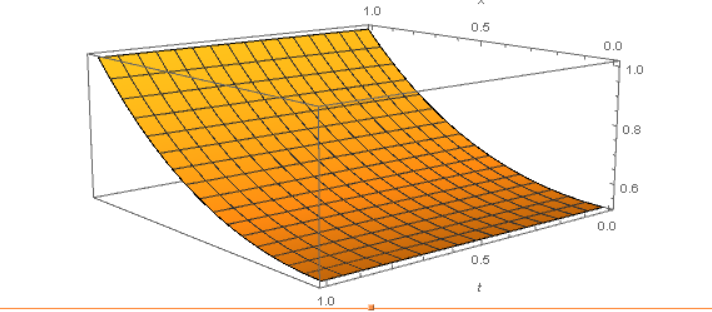


Рис. 2 – Поверхня точного рішення

Кроки і :

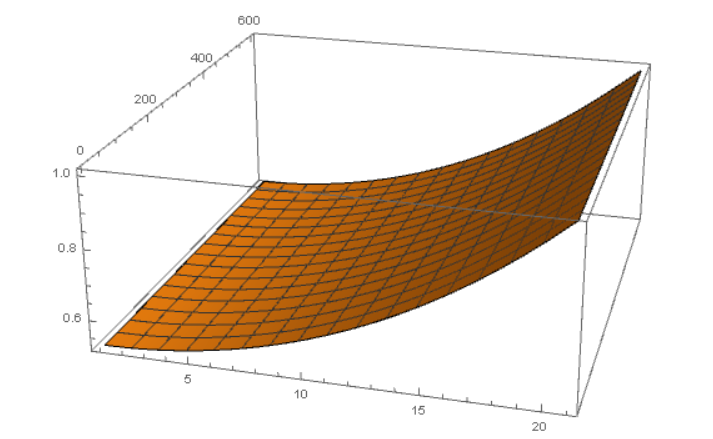


Рис. 3 – Поверхня, знайдена явним методом

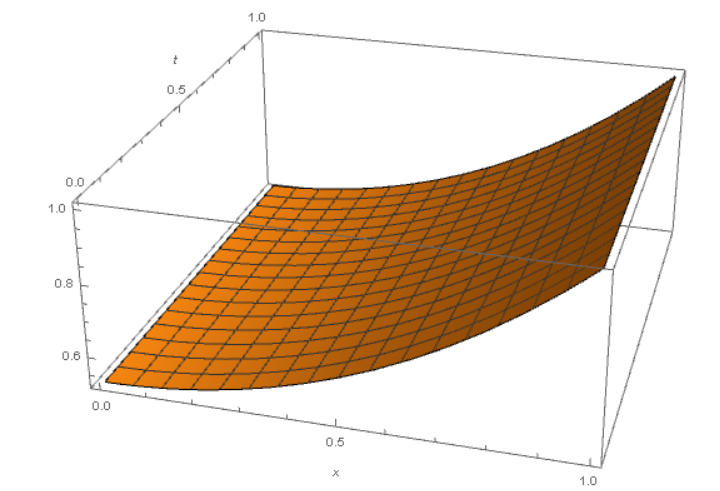


Рис. 4 – Поверхня точного рішення

Кроки і :

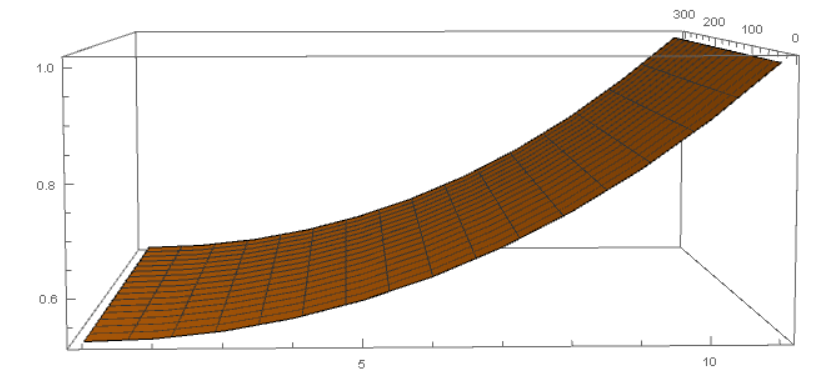


Рис. 5 – Поверхня, знайдена явним методом

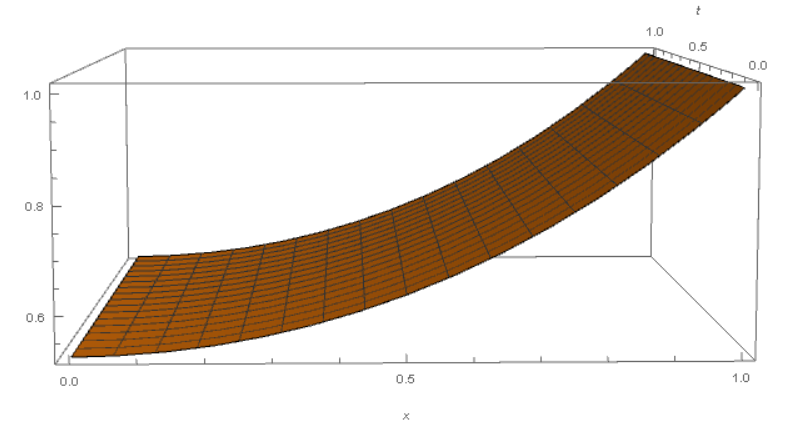
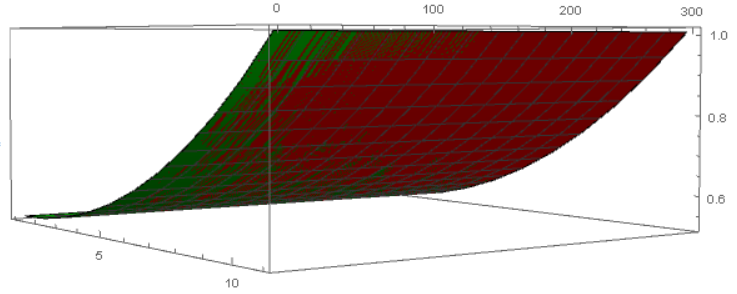


Рис. 6 – Поверхня точного рішення

Накладання точного розв’язку на розв’язок знайдений за допомогою явного методу:



# **ВИСНОВОК**

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

Також було знайдено похибки для явного методу при різних кроках.

Було порівняно час на вирішення задачі послідовно і паралельно. Час на паралельну програму виявився більшим, ніж при послідовному пошуку рішень. 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) {

DiffEquation diff = **new** DiffEquation();

ExactSolution exactSolve = **new** ExactSolution(diff);

**double**[][] exactResult = exactSolve.solve();

System.***out***.println("Последовательное решение явным методом:");

SerialSolution serialSolve = **new** SerialSolution(diff);

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

**double**[][] serialResult = serialSolve.solve();

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

**long** executeTimeSerial = endTimeSerial - startTimeSerial;

System.***out***.println("Результат работы явного метода:");

diff.printMatrix(serialResult);

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

//System.out.println();

System.***out***.println("\nПогрешности:");

*printResultError*(diff, serialResult, exactResult);

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

System.***out***.println("\nПараллельное решение явным методом:");

ParallelSolution parallelSolve = **new** ParallelSolution(diff);

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

**double**[][] parallelResult = parallelSolve.solve();

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

**long** executeTimeParallel = endTimeParallel - startTimeParallel;

//diff.printMatrix(parallelResult);

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

System.***out***.println("\nПогрешности:");

*printResultError*(diff, parallelResult, exactResult);

}

**private** **static** **void** printResultError(DiffEquation diff, **double**[][] DifferenceSchemeSolve, **double**[][] trueMatrix) {

MathError err = **new** MathError(diff, DifferenceSchemeSolve, trueMatrix);

System.***out***.println("Средняя абсолютная погрешность: " + err.mediumAbsoluteError());

System.***out***.println("Максимальная абсолютная погрешность: " + err.maxAbsoluteError());

System.***out***.println("Средняя относительная погрешность: " + err.mediumRelativeError());

System.***out***.println("Максимальная относительная погрешность: " + err.maxRelativeError());

}

}

**DiffEquation.java**

**public** **class** DiffEquation {

**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 = 0.00006;

**private** **final** **double** b = 0.00004;

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

**private** **final** **double** B = 2.8;

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

**public** **int** getTPointsNumber() {

**return** (**int**) Math.*ceil*((t1 - t0) / tau) + 1;

}

**public** **int** getHPointsNumber() {

**return** (**int**) Math.*ceil*((x1 - x0) / h) + 1;

}

**public** **double** calcExactSolution(**double** x, **double** t) {

**return** Math.*pow*((Math.*pow*(x, 2)/(B-8\*a\*t) + A\*Math.*pow*(B-8\*a\*t, -1.0/4) - (b\*(B-8\*a\*t))/(15\*a)), 3.0/2);

}

**public** **double** calcInitialCondition(**double** x) {

**return** Math.*pow*((Math.*pow*(x, 2)/B + A\*Math.*pow*(B, -1.0/4) - (b\*B)/(15\*a)), 3.0/2);

}

**public** **double** calcLeftBorder(**double** t) {

**return** Math.*pow*((A\*Math.*pow*(B-8\*a\*t, -1.0/4) - (b\*(B-8\*a\*t))/(15\*a)), 3.0/2);

}

**public** **double** calcRightBorder(**double** t) {

**return** Math.*pow*((1.0/(B-8\*a\*t) + A\*Math.*pow*(B-8\*a\*t, -1.0/4) - (b\*(B-8\*a\*t))/(15\*a)), 3.0/2);

}

**public** **double** calcApproximateSolution(**double** wLeft, **double** wCentral , **double** wRight) {

**return** (wCentral + a\*sgm\*((1/6)\*Math.*pow*(wCentral, -1.0/3)\*Math.*pow*(wRight - wLeft, 2) + Math.*pow*(wCentral, 2.0/3)\*(wRight - 2\*wCentral + wLeft)) + b\*tau\*Math.*pow*(wCentral, 1.0/3));

}

**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*("%.6f\t", matrix[i][j]));

}

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

}

}

**public** **double** getX0() {

**return** x0;

}

**public** **double** getX1() {

**return** x1;

}

**public** **double** getT0() {

**return** t0;

}

**public** **double** getT1() {

**return** t1;

}

**public** **double** getH() {

**return** h;

}

**public** **double** getTau() {

**return** tau;

}

}

**SerialSolution.java**

**public** **class** SerialSolution {

**private** **int** TPointsNumber;

**private** **int** HPointsNumber;

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

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

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

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

**private** DiffEquation diff;

**public** SerialSolution(DiffEquation diff) {

**this**.diff = diff;

**this**.TPointsNumber = diff.getTPointsNumber();

**this**.HPointsNumber = diff.getHPointsNumber();

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

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

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

**this**.tau = diff.getTau();

}

**public** **double**[][] solve() {

**double** x = x0;

**double** w[][] = **new** **double**[TPointsNumber][HPointsNumber];

**for** (**int** j = 0; j < HPointsNumber; j++, x += h) {

w[0][j] = diff.calcInitialCondition(x);

}

**double** t = t0 + tau;

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

w[i][0] = diff.calcLeftBorder(t);

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

w[i][j] = diff.calcApproximateSolution(w[i-1][j-1],w[i-1][j],w[i-1][j+1]);

}

w[i][HPointsNumber - 1] = diff.calcRightBorder(t);

}

**return** w;

}

}

**ParallelSolution.java**

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

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

**public** **class** ParallelSolution {

**private** **int** TPointsNumber;

**private** **int** HPointsNumber;

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

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

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

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

**private** DiffEquation diff;

**public** ParallelSolution(DiffEquation diff) {

**this**.diff = diff;

**this**.TPointsNumber = diff.getTPointsNumber();

**this**.HPointsNumber = diff.getHPointsNumber();

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

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

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

**this**.tau = diff.getTau();

}

**public** **double**[][] solve() {

**double** x = x0;

**double** w[][] = **new** **double**[TPointsNumber][HPointsNumber];

/\*

for (int j = 0; j < HPointsNumber; j++, x += h) {

w[0][j] = diff.calcInitialCondition(x);

}\*/

IntStream.*range*(0, diff.getHPointsNumber()).parallel().forEach(j -> {

w[0][j] = diff.calcInitialCondition(j\*h);

});

**double** t = t0 + tau;

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

w[i][0] = diff.calcLeftBorder(t);

w[i][HPointsNumber - 1] = diff.calcRightBorder(t);

**int** index = i;

IntStream.*range*(1, diff.getHPointsNumber()-1).parallel().forEach(j -> {

w[index][j] = diff.calcApproximateSolution(w[index-1][j-1],w[index-1][j],w[index-1][j+1]);

});

}

**return** w;

}

}

**ExactSolution.java**

**public** **class** ExactSolution {

**private** **int** TPointsNumber;

**private** **int** HPointsNumber;

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

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

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

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

**private** DiffEquation diff;

**public** ExactSolution(DiffEquation diff) {

**this**.diff = diff;

**this**.TPointsNumber = diff.getTPointsNumber();

**this**.HPointsNumber = diff.getHPointsNumber();

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

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

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

**this**.tau = diff.getTau();

}

**public** **double**[][] solve() {

**double** trueMatrix[][] = **new** **double**[TPointsNumber][HPointsNumber];

**double** t = t0;

**for** (**int** i = 0; i < TPointsNumber; i++) {

**double** x = x0;

**for** (**int** j = 0; j < HPointsNumber; j++) {

trueMatrix[i][j] = diff.calcExactSolution(x, t);

x += h;

}

t += tau;

}

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

diff.printMatrix(trueMatrix);

**return** trueMatrix;

}

}

**MathError.java**

**public** **class** MathError {

**private** DiffEquation diff;

**private** **double**[][] DifferenceSchemeSolve;

**private** **double**[][] trueMatrix;

**public** MathError(DiffEquation diff, **double**[][] DifferenceSchemeSolve, **double**[][] trueMatrix) {

**this**.diff = diff;

**this**.DifferenceSchemeSolve = DifferenceSchemeSolve;

**this**.trueMatrix = trueMatrix;

}

**public** **double** mediumAbsoluteError() {

**double**[][] errorMatrix = **new** **double**[diff.getTPointsNumber()][diff.getHPointsNumber()];

**double** error = 0.0;

**for** (**int** i = 0; i < diff.getTPointsNumber(); i++) {

**for** (**int** j = 0; j < diff.getHPointsNumber(); j++) {

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

error += errorMatrix[i][j];

}

}

error = error / (diff.getTPointsNumber() \* diff.getHPointsNumber());

**return** error;

}

**public** **double** maxAbsoluteError() {

**double**[][] errorMatrix = **new** **double**[diff.getTPointsNumber()][diff.getHPointsNumber()];

**double** error = 0.0;

**for** (**int** i = 0; i < diff.getTPointsNumber(); i++) {

**for** (**int** j = 0; j < diff.getHPointsNumber(); j++) {

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

**if**(i == 0 && j == 0) { error = errorMatrix[i][j];}

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

error = errorMatrix[i][j];

}

}

}

**return** error;

}

**public** **double** mediumRelativeError() {

**double**[][] errorMatrix = **new** **double**[diff.getTPointsNumber()][diff.getHPointsNumber()];

**double** error = 0.0;

**for** (**int** i = 0; i < diff.getTPointsNumber(); i++) {

**for** (**int** j = 0; j < diff.getHPointsNumber(); j++) {

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

error += errorMatrix[i][j];

}

}

error = error / (diff.getTPointsNumber() \* diff.getHPointsNumber());

**return** error;

}

**public** **double** maxRelativeError() {

**double**[][] errorMatrix = **new** **double**[diff.getTPointsNumber()][diff.getHPointsNumber()];

**double** error = 0.0;

**for** (**int** i = 0; i < diff.getTPointsNumber(); i++) {

**for** (**int** j = 0; j < diff.getHPointsNumber(); j++) {

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

**if**(i == 0 && j == 0) { error = errorMatrix[i][j];}

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

error = errorMatrix[i][j];

}

}

}

**return** error;

}

}

Посилання на GitHub:

<https://github.com/milnavig/KursWork1>