**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»**

**ім. Ігоря Сікорського»**

Кафедра системного проектування

**Курсова робота**

з дисципліни

«Паралельні обчислення»

Тема: «**Чисельне розв’язання диференціальних рівнянь в частинних похідних з використанням паралельних обчислень»**

Виконала студентка 3 курсу

групи ДА-61

Шкульова К. С.

Перевірив Яременко В. С.

**Київ – 2019**

Національний Технічний Університет України “Київський Політехнічний Інститут”

(назва вищого навчального закладу)

Системного проектування

Кафедра\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Паралельні обчислення

Дисципліна\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6.050101 – Комп’ютерні науки

Напрям підготовки\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6.05010102 Інформаційні технології проектування,

6.05010103 Системне проектування \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Спеціальність\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3 ДА -61 6

Курс\_\_\_\_\_\_\_\_\_\_Група\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Семестр\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*804.ДА.6121-01 15*

**ЗАВДАННЯ**

**на курсову роботу з дисципліни**

“Паралельні обчислення”

**1. Тема роботи:** Чисельне розв’язання диференціальних рівнянь в частинних похідних з використанням паралельних обчислень.

**2. Вихідні дані до роботи***.* Розв’язати одновимірну нестаціонарну нелінійну задачу для диференційних рівнянь у частинних похідних. Для знаходження крайових та початкових умов необхідно знайти значення точного розв’язку у відповідних точках. Перевірити по можливості підстановкою точний розв’язок. Значення констант в задачі взяти довільними. Розрахункова область: t [0,1], x [0,1] . Номер задачі співпадає з номером варіанту.

Використовуються наступні методи: 1) явний; 2) неявний; 3) Кранка- Ніколсона. Тип різницевої сітки: 1) фіксована рівномірна.

**3.** **Перелік питань, які мають бути розроблені:**

1. У програмі вивести таблицю точного та наближеного розв’язків для 4-6 точок по вісі Ox для кожного з 5-10 часових шарів.

2. Отриманий розв’язок зобразити графічно у вигляді 3D графіків та їх перерізів по вісі часу.

3. Вивести максимальні абсолютну та відносну різниці між точним та наближеним розв’язками. У програмі кількість вузлів в обох координатних напрямках 100-1000.

4. Обчислювальна частина програми повинна бути простою, наочною і зрозумілою. Паралельну реалізацію виконувати за допомогою технологій OpenMP, MPI, або інших, розглянутих у даному навчальному курсі.

5. Порівняти паралельні реалізації з послідовними.

**Анотація**

В курсовій роботі розглянуто одновимірну нестаціонарну нелінійну задачу для диференційних рівнянь у частинних похідних, знайдено її точний розв’язок за заданою формулою, розроблено послідовну та паралельну реалізації для чисельних розв’язків, побудованих на аналітичному знаходженні явної функції та крайових і початкових умов. Для кожного з отриманих результатів були побудовані відповідні 3D-графіки та обчислені похибки для послідовного та паралельного вирішень при різній щільності різницевої сітки. Було застосовано явний метод та фіксовану рівномірну різницеву сітку. Мова програмування, що використовувалась – Java, для розпаралелювання задачі використовувався клас Thread та його можливості.

**Зміст**

[**Варіант завдання 6**](#_Toc8937833)

[**Виведена формула для явного способу розв’язування рівняння 6**](#_Toc8937834)

[**Результат виконання програми для кроку по x – 0.1 , по t –10-4: 7**](#_Toc8937835)

[**Графічне відображення отриманих результатів 7**](#_Toc8937836)

[**Результат виконання програми для кроку по x – 0.05 , по t –6.25\*10-6: 7**](#_Toc8937837)

[**Графічне відображення отриманих результатів 7**](#_Toc8937838)

[**Результат виконання програми для кроку по x – 1/30, по t – 1.234\*10-6: 8**](#_Toc8937839)

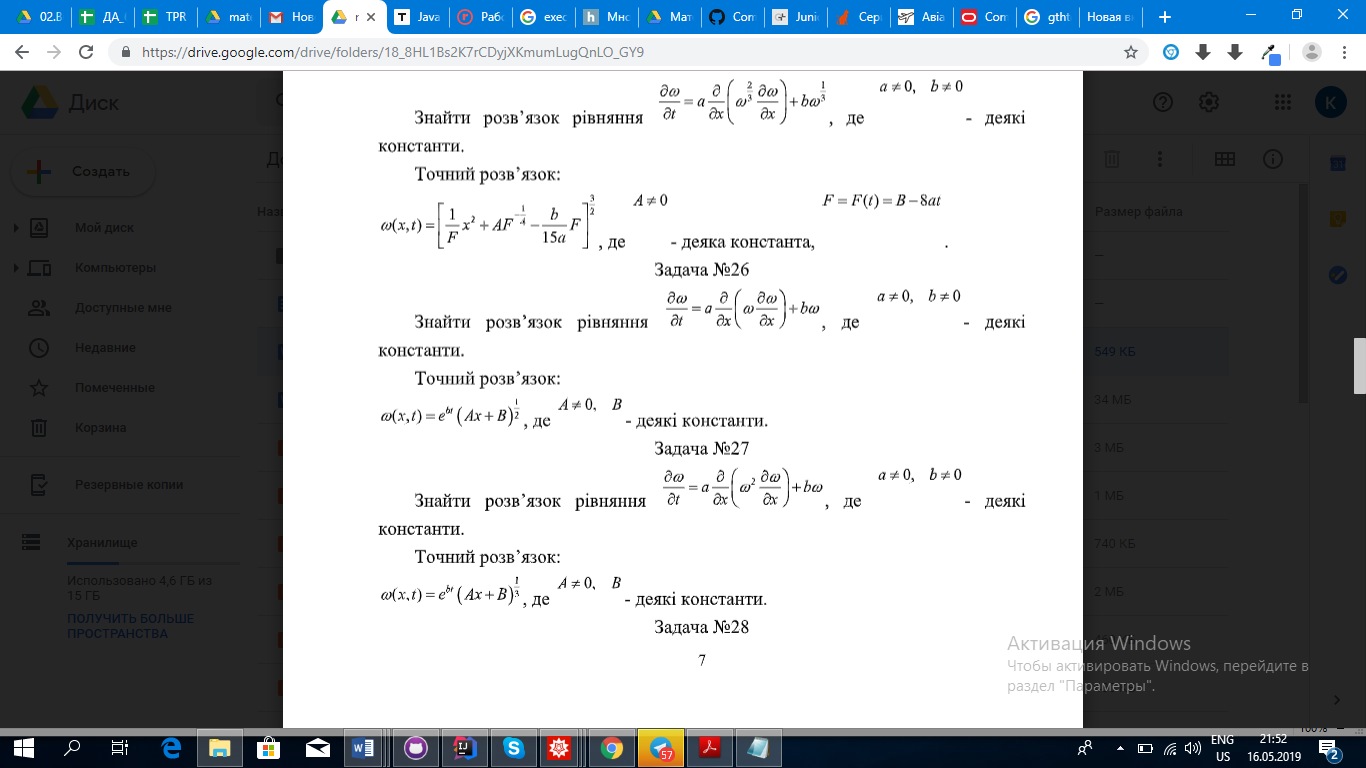
[**Графічне відображення отриманих результатів 8**](#_Toc8937840)

[**Використання системи контролю версій 9**](#_Toc8937841)

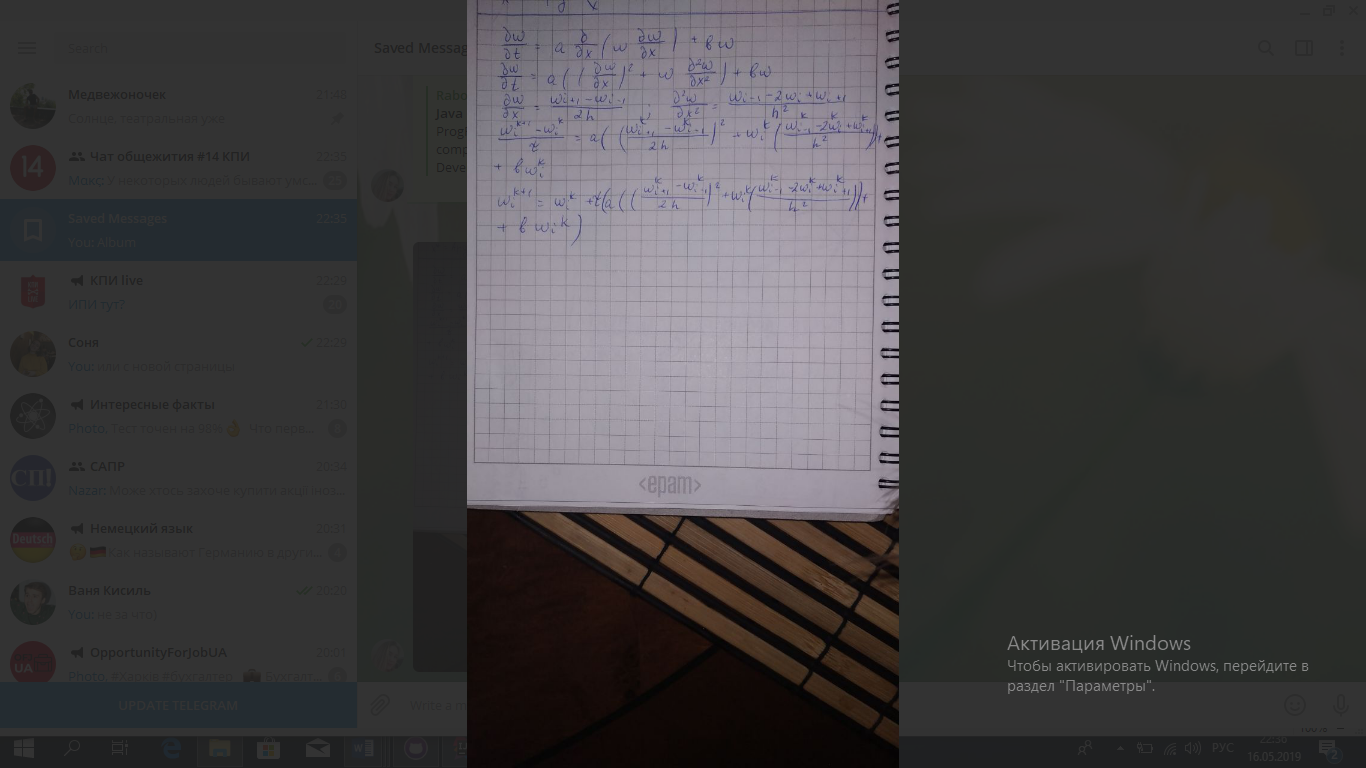
[**Посилання на систему контролю версій (Git-репозиторій): 9**](#_Toc8937842)

[**Лістинг програми: 9**](#_Toc8937843)

**Варіант завдання**:



# **Виведена формула для явного способу розв’язування рівняння**



Отримана формула для загального випадку констант а і b.

Константи ми вибираємо так, щоб знизити похибку, в програмі константи можуть бути змінені. Для оптимальнішого розв’язку було обрано такі константи: a=1.7, b=0.5, A=72, B=14;  
τ – крок по часові шкалі, h – крок по x

# **Результат виконання програми для кроку по x – 0.1 , по t –10-4:**

Sequantial time: 389973 ns

Parallel time: 94226773 ns

Absolute error: 0.3453276432274741

Relative error: 0.0392105608476773

# **Графічне відображення отриманих результатів**

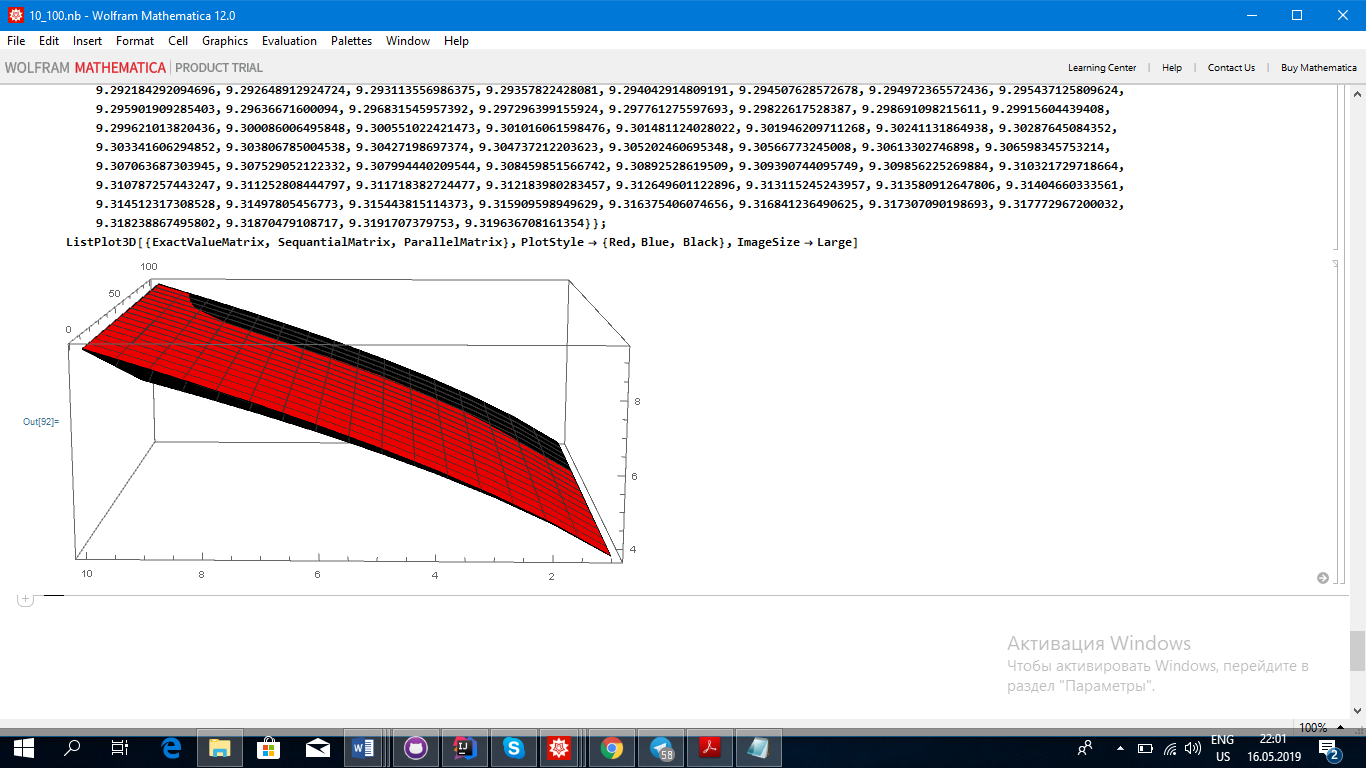


Рис. 1 – Переріз графіків для точного, послідовного і паралельного розв’язків(з вказаними раніше константами)

# **Результат виконання програми для кроку по x – 0.05 , по t –6.25\*10-6:**

Sequantial time: 1809493

Parallel time: 360968534Parallel: 129 ms

Absolute error: 0.20199499125030407

Relative error: 0.02224876280666162

# **Графічне відображення отриманих результатів**

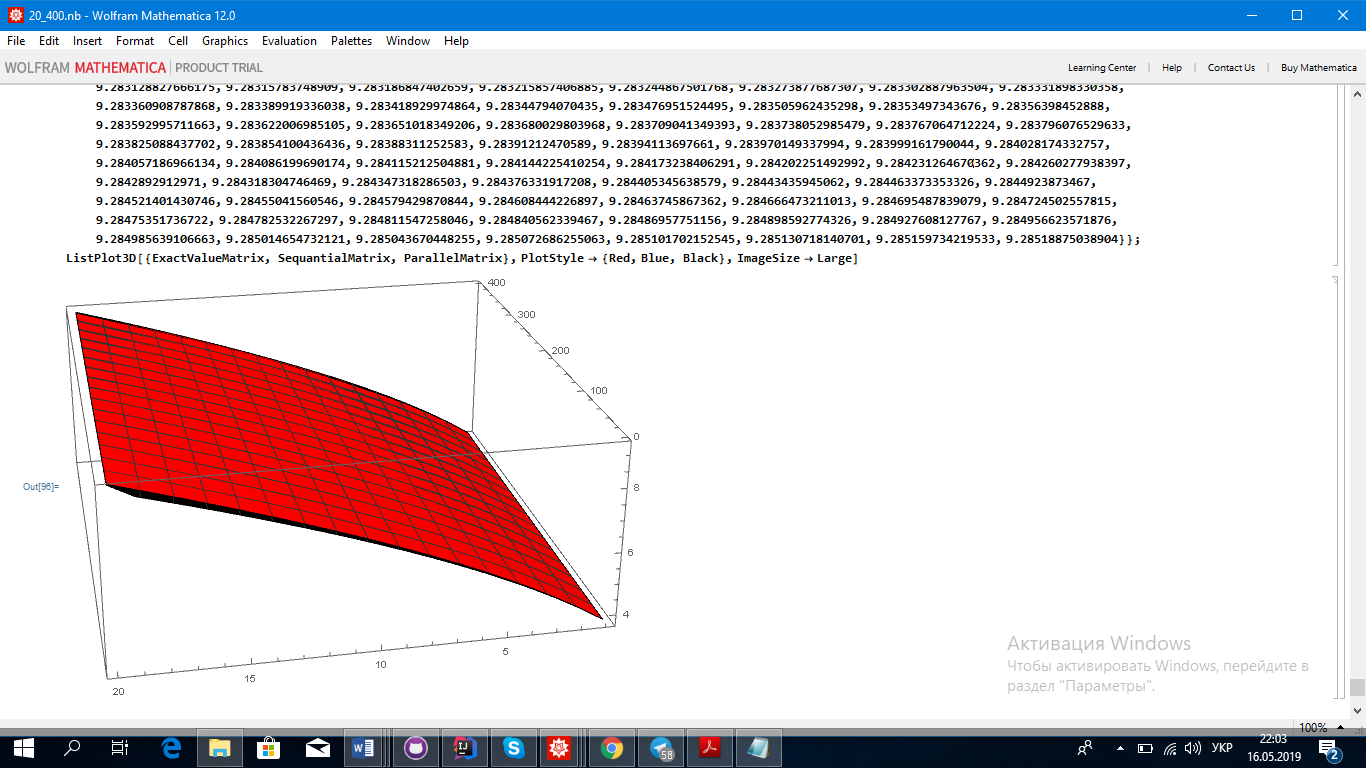


Рис. 2 – Переріз графіків для точного, послідовного і паралельного розв’язків(з вказаними раніше константами)

# **Результат виконання програми для кроку по x – 1/30, по t – 1.234\*10-6:**

Sequantial time: 4884907

Parallel time: 620139520

Absolute error: 0.14126864042344955

Relative error: 0.01543519281253452

# **Графічне відображення отриманих результатів**

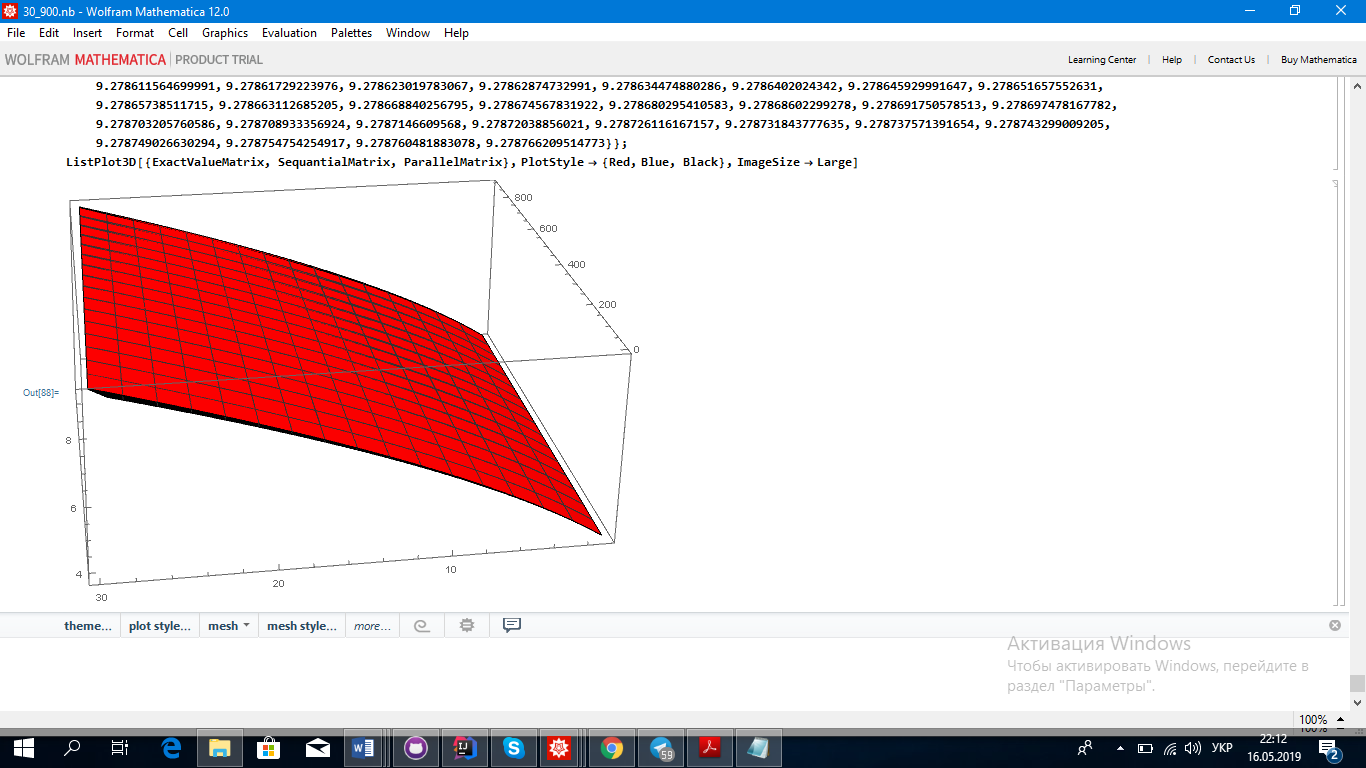
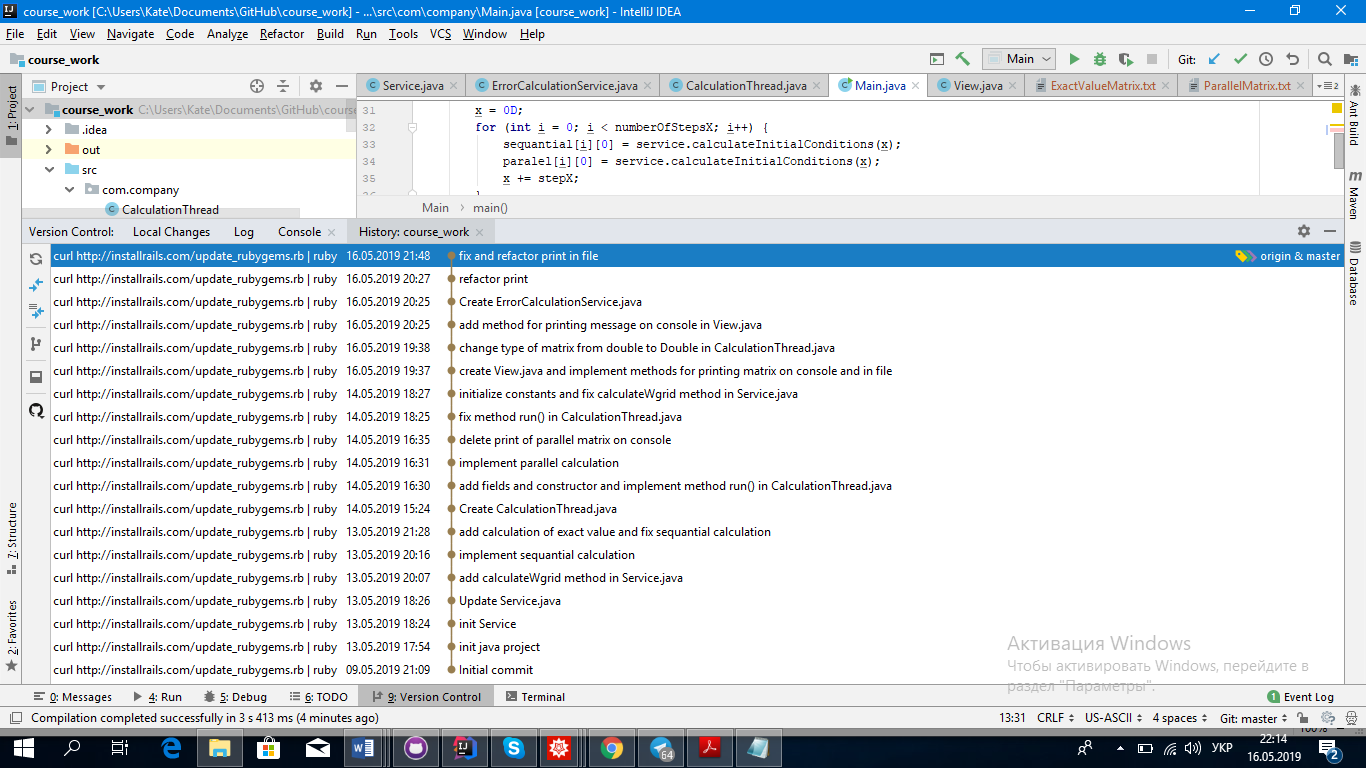


Рис. 3 – Переріз графіків для точного, послідовного і паралельного розв’язків(з вказаними раніше константами)

Як можна зрозуміти з отриманих графіків та комп’ютерних обчислень, при зменшенні щільності збільшується похибка обчислення наближених значень.

**Використання системи контролю версій:**



# **Посилання на систему контролю версій (Git-репозиторій):**

<https://github.com/katiesh/course_work>

# **Лістинг програми:**

**public class** CalculationThread **extends** Thread{  
 **private int startIndex**;  
 **private int endIndex**;  
 **private int column**;  
 **private** Double **matrix**[][];  
 **private double stepX**;  
 **private double stepT**;  
 **private** Service **service**;  
 **public** CalculationThread(**int** startIndex, **int** endIndex, **int** column, Double matrix[][],  
 **double** stepX, **double** stepT){  
 **this**.**startIndex** = startIndex;  
 **this**.**endIndex** = endIndex;  
 **this**.**column** = column;  
 **this**.**matrix** = matrix;  
 **this**.**stepX** = stepX;  
 **this**.**stepT** = stepT;  
 **this**.**service** = **new** Service();  
 }  
  
 @Override  
 **public void** run() **throws** RuntimeException {  
 **for** (**int** i = **startIndex**; i <**endIndex**; i++) {  
 **matrix**[i][**column**] = **service**.calculateWgrid(**matrix**[i][**column**-1], **matrix**[i-1][**column**-1],  
 **matrix**[i+1][**column**-1], **stepX**, **stepT**);  
 }  
 }  
}

**public class** ErrorCalculationService {  
 **private int iMax**;  
 **private int jMax**;  
 **private double maxDifference**;  
 **public double** calculateAbsoluteError(Double matrix1[][], Double[][] matrix2){  
 **for** (**int** i = 0; i < matrix1.**length** ; i++) {  
 **for** (**int** j = 0; j < matrix1[i].**length** ; j++) {  
 **if** (Math.*abs*(matrix2[i][j] - matrix1[i][j]) > **maxDifference**){  
 **maxDifference** = Math.*abs*(matrix2[i][j] - matrix1[i][j]);  
 **iMax** = i;  
 **jMax** = j;  
 }  
 }  
 }  
 **return maxDifference**;  
 }  
 **public double** calculateRelativeError(Double[][] matrix){  
 **return maxDifference**/matrix[**iMax**][**jMax**];  
 }  
  
}

**public class** Service {  
 **public static final double *a*** = 1.7;  
 **public static final double *b*** = 0.5;  
 **public static final double *A*** = 72;  
 **public static final double *B*** = 14;  
  
 **public double** calculateInitialConditions(**double** x){  
 **return** calculateExactValue(x,0);  
 }  
 **public double** calculateBoundaryConditionsForOne(**double** t){  
 **return** calculateExactValue(1, t);  
 }  
 **public double** calculateBoundaryConditionsForZero(**double** t){  
 **return** calculateExactValue(0, t);  
 }  
 **public double** calculateExactValue(**double** x, **double** t){  
 **return** Math.*exp*(***b***\*t)\*Math.*pow*((***A***\*x + ***B***), 0.5D);  
 }  
 **public double** calculateWgrid(**double** wi, **double** wiPrev, **double** wiNext, **double** h, **double** tau){  
 **return** wi + tau\*(***a***\*(Math.*pow*((wiNext - wiPrev)/(2D\*h), 2) + wi\*((wiPrev - 2D\*wi + wiNext)/Math.*pow*(h,2))) + ***b***\*wi);  
 }  
}

**public class** View {  
 **public** <T> **void** printMatrix(T matrix[][], String name){  
 System.***out***.println(name + **":"**);  
 **for** (**int** i = 0; i <matrix.**length** ; i++) {  
 **for** (**int** j = 0; j <matrix[i].**length** ; j++) {  
 System.***out***.print(matrix[i][j] + **" "**);  
 }  
 System.***out***.println();  
 }  
 }  
  
 **public** <T> **void** printMatrixInFile(T matrix[][], String fileName, String name){  
 **try** ( FileWriter fileWriter = **new** FileWriter(fileName)){  
 fileWriter.write(name + **" = {"**);  
 **for** (**int** i = 0; i < matrix.**length**; i++) {  
 fileWriter.write(**"{ "**);  
 **for** (**int** j = 0; j < matrix[i].**length**; j++) {  
 **if** (j < matrix[i].**length** - 1) {  
 fileWriter.write(matrix[i][j] + **", "**);  
 } **else** {  
 fileWriter.write(matrix[i][j] + **""**);  
 }  
 }  
 **if** (i < matrix.**length** - 1) {  
 fileWriter.write(**" }, "**);  
 } **else** {  
 fileWriter.write(**" } "**);  
 }  
 }  
 fileWriter.write(**" };"**);  
 fileWriter.flush();  
 } **catch** (IOException e){  
 e.printStackTrace();  
 }  
 }  
  
 **public void** print (String message){  
 System.***out***.println(message);  
 }  
}

**public class** Main {  
  
 **public static void** main(String[] args) **throws** InterruptedException, IOException {  
 Service service = **new** Service();  
 View view = **new** View();  
 **int** numberOfStepsX = 30; *// 20 30* **int** numberOfStepsT = 900; *// 400 900* **double** stepX = 1D / numberOfStepsX;  
 **double** sigma = 1D / numberOfStepsT;  
 **double** stepT = Math.*pow*(stepX, 2) \* sigma;  
 Double[][] exactValue = **new** Double[numberOfStepsX][numberOfStepsT];  
 Double[][] sequantial = **new** Double[numberOfStepsX][numberOfStepsT];  
 Double[][] paralel = **new** Double[numberOfStepsX][numberOfStepsT];  
 **double** x = 0D;  
 **double** t = 0D;  
 **for** (**int** i = 0; i < numberOfStepsX; i++) {  
 **for** (**int** j = 0; j < numberOfStepsT; j++) {  
 exactValue[i][j] = service.calculateExactValue(x, t);  
 t += stepT;  
 }  
 x += stepX;  
 }  
 x = 0D;  
 **for** (**int** i = 0; i < numberOfStepsX; i++) {  
 sequantial[i][0] = service.calculateInitialConditions(x);  
 paralel[i][0] = service.calculateInitialConditions(x);  
 x += stepX;  
 }  
 t = 0D;  
 **for** (**int** i = 0; i < numberOfStepsT; i++) {  
 sequantial[0][i] = service.calculateBoundaryConditionsForZero(t);  
 paralel[0][i] = service.calculateBoundaryConditionsForZero(t);  
 sequantial[numberOfStepsX - 1][i] = service.calculateBoundaryConditionsForOne(t);  
 paralel[numberOfStepsX - 1][i] = service.calculateBoundaryConditionsForOne(t);  
 t += stepT;  
 }  
 **long** sequantialTime = System.*nanoTime*();  
 **for** (**int** j = 1; j < numberOfStepsT; j++) {  
 **for** (**int** i = 1; i < numberOfStepsX - 1; i++) {  
 sequantial[i][j] = service.calculateWgrid(sequantial[i][j - 1], sequantial[i - 1][j - 1],  
 sequantial[i + 1][j - 1], stepX, stepT);  
 }  
 }  
 sequantialTime = System.*nanoTime*() - sequantialTime;  
 **int** numberOfThreads = 3;  
 **long** parallelTime = System.*nanoTime*();  
 **for** (**int** j = 1; j < numberOfStepsT; j++) {  
 List<CalculationThread> calculationThreads = **new** ArrayList<>();  
 **for** (**int** i = 0; i < numberOfThreads; i++) {  
 calculationThreads.add(**new** CalculationThread(i == 0 ? 1 : (numberOfStepsX - 1) / numberOfThreads \* i,  
 i == (numberOfThreads - 1) ? (numberOfStepsX - 1) : (numberOfStepsX - 1) / numberOfThreads \* (i + 1),  
 j, paralel, stepX, stepT));  
  
 calculationThreads.get(i).start();  
 }  
 **for** (**int** i = 0; i < numberOfThreads; i++) {  
 calculationThreads.get(i).join();  
 }  
 }  
 parallelTime = System.*nanoTime*() - parallelTime;  
 view.print(**"Sequantial time: "** + sequantialTime);  
 view.print(**"Parallel time: "** + parallelTime);  
 view.printMatrix(sequantial, **"Sequantial"**);  
 view.printMatrix(paralel, **"Parallel"**);  
 view.printMatrixInFile(exactValue, **"ExactValueMatrix.txt"**, **"ExactValueMatrix"**);  
 view.printMatrixInFile(sequantial, **"SequantialMatrix.txt"**, **"SequantialMatrix"**);  
 view.printMatrixInFile(paralel, **"ParallelMatrix.txt"**, **"ParallelMatrix"**);  
 ErrorCalculationService errorCalculationService = **new** ErrorCalculationService();  
 view.print(**"Absolute error: "** + errorCalculationService.calculateAbsoluteError(exactValue, paralel));  
 view.print(**"Relative error: "** + errorCalculationService.calculateRelativeError(exactValue));  
 }  
}