

*МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ*

*НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ*

*„КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. СІКОРСЬКОГО”*

*НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС*

*„ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ”*

***Курсова робота***

*з курсу «Паралельні обчислення*»

*Тема:* «Числові методи в інформатиці»

Виконав

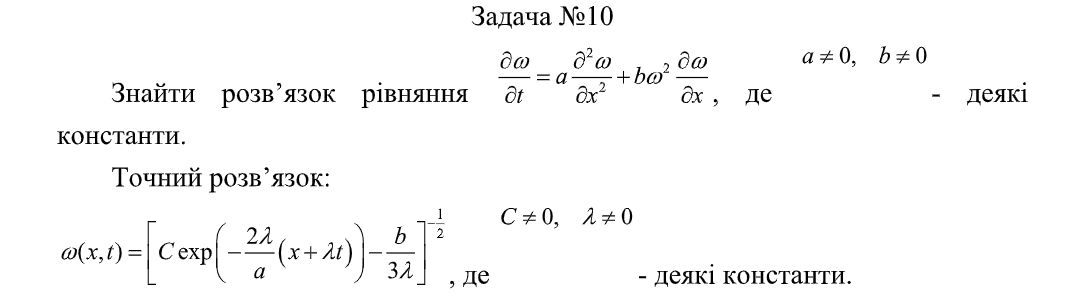
студент III курсу

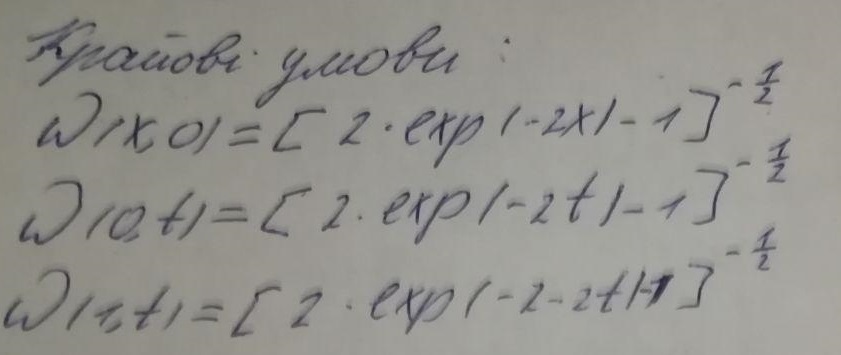
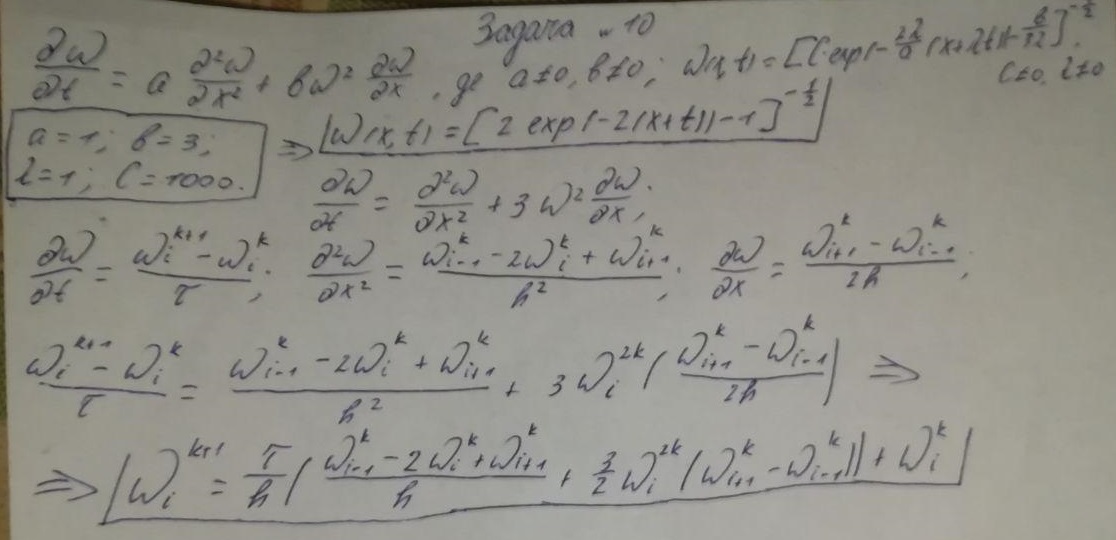
групи ДА-61

*Корнієнко Олександр*

***Київ – 2019***

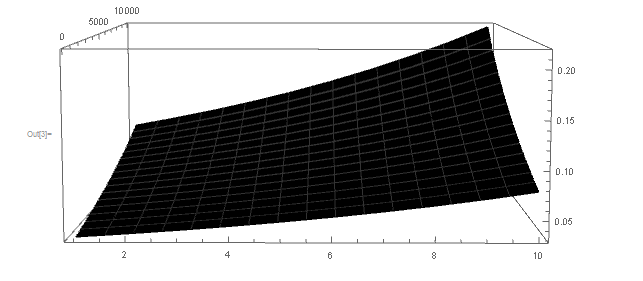
**Завдання**





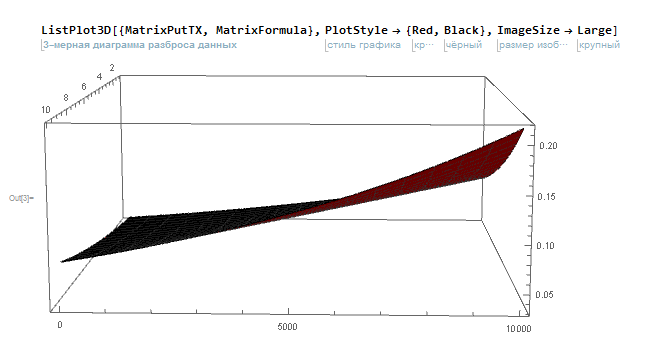
## Явний метод пошуку точок сітки

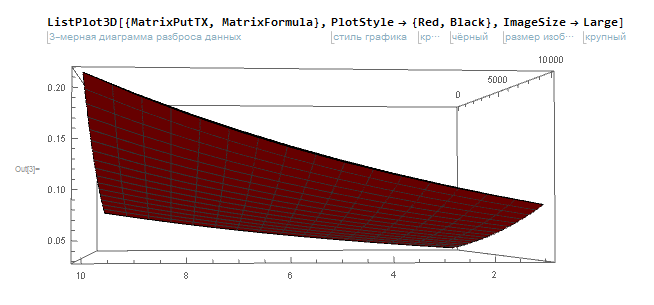
Крок h = 0.1; t = 0.0001 маємо наступний результат:



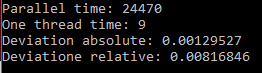
**Точний розв’язок**

**Наближений розв’язок**





**Абсолютна та відносна похибки та час роботи:**



## Висновок

У даній роботи була розв’язана одновимірна нестаціонарна нелінійна задача для диференційних рівнянь у частинних похідних.

Для розв’язку був використаний явний метод чисельного розв’язку диференційного рівняння. Згідно з отриманими результатами: із збільшенням кількості точок у сітці розв’язок стає більш точним.

Calculations.cpp

#include "pch.h"

#include "Calculations.h"

#include <math.h>

int Calculations::ColumnsNumber = 100;

int Calculations::RowsNumber = 20000;

double Calculations::H = (double) 1 / Calculations::ColumnsNumber;

double Calculations::T = (double) 1 / Calculations::RowsNumber;

double Calculations::GetExactResult(double t, double x)

{

return

pow(

1000 \* exp(-2 \* (x + t)) - 1

, -0.5);

}

void Calculations::CalcAndSetApproximateResult(double\*\* matrix, int i, int j)

{

matrix[i][j] =

T / H \* ((matrix[i - 1][j - 1] - 2 \* matrix[i - 1][j] + matrix[i - 1][j + 1]) / H +

3 / 2 \* pow(matrix[i - 1][j], 2) \* (matrix[i - 1][j + 1] - matrix[i - 1][j - 1]))

+ matrix[i - 1][j];

}

ParallelCoursework.cpp

#include "pch.h"

#include <iostream>

#include "Calculations.h"

#include <vector>

#include <iostream>

#include <thread>

#include <chrono>

#include <fstream>

using namespace std;

static int ThreadsCount = 4;

static int ChunkSize = Calculations::ColumnsNumber / ThreadsCount;

static void Task1(int index, double\*\* matrix)

{

int indexFrom;

int indexTo;

int chunkSize = Calculations::RowsNumber / ThreadsCount;

if (index == ThreadsCount - 1)

{

indexFrom = index \* chunkSize;

indexTo = Calculations::RowsNumber - 1;

}

else

{

indexFrom = index \* chunkSize;

indexTo = (index + 1) \* chunkSize - 1;

}

for (int i = indexFrom; i <= indexTo; i++)

{

matrix[i][0] = Calculations::GetExactResult(Calculations::T \* i, 0);

matrix[i][Calculations::ColumnsNumber - 1] =

Calculations::GetExactResult(Calculations::T \* i, Calculations::H \* (Calculations::ColumnsNumber - 1));

}

}

static void Task2(int index, double\*\* matrix)

{

int indexFrom;

int indexTo;

if (index == ThreadsCount - 1)

{

indexFrom = index \* ChunkSize;

indexTo = Calculations::ColumnsNumber - 1;

}

else

{

indexFrom = index \* ChunkSize;

indexTo = (index + 1) \* ChunkSize - 1;

}

for (int i = indexFrom; i <= indexTo; i++)

{

matrix[0][i] = Calculations::GetExactResult(0, Calculations::H \* i);

}

}

static void Task3(int row, int index, double\*\* matrix)

{

int indexFrom;

int indexTo;

if (index == ThreadsCount - 1)

{

indexFrom = index \* ChunkSize;

indexTo = Calculations::ColumnsNumber - 1;

}

else

{

indexFrom = index \* ChunkSize;

indexTo = (index + 1) \* ChunkSize;

}

if (indexFrom == 0)

indexFrom = 1;

for (int j = indexFrom; j < indexTo; j++)

{

Calculations::CalcAndSetApproximateResult(matrix, row, j);

}

}

static void Deviation(double \*\*array1, double \*\*array2, int row, int column)

{

double absolute = -1;

int t\_i;

int t\_j;

double relative;

for (int i = 0; i < row; i++)

{

for (int j = 0; j < column; j++)

{

double temp\_val = abs(array1[i][j] - array2[i][j]);

if (temp\_val > absolute)

{

absolute = temp\_val;

t\_i = i;

t\_j = j;

}

}

}

std::cout << "Deviation absolute: " << absolute << std::endl;

relative = absolute / array2[t\_i][t\_j];

std::cout << "Deviatione relative: " << relative << std::endl;

}

int main()

{

vector<thread> threads;

double\*\* dataMatrixAproximate = new double\*[Calculations::RowsNumber];

double\*\* dataMatrixExact = new double\*[Calculations::RowsNumber];

for (int i = 0; i < Calculations::RowsNumber; i++)

{

dataMatrixAproximate[i] = new double[Calculations::ColumnsNumber];

dataMatrixExact[i] = new double[Calculations::ColumnsNumber];

}

chrono::time\_point<chrono::system\_clock> TimerStart = chrono::system\_clock::now();

if (ThreadsCount > Calculations::ColumnsNumber)

{

ThreadsCount = Calculations::ColumnsNumber;

ChunkSize = 1;

}

int count = 0;

for (int i = 0; i < ThreadsCount; i++)

{

threads.emplace\_back(thread(Task2, count++, dataMatrixAproximate));

}

for (int i = 0; i < ThreadsCount; i++)

{

threads[i].join();

}

threads.clear();

count = 0;

for (int i = 0; i < ThreadsCount; i++)

{

threads.emplace\_back(thread(Task1, count++, dataMatrixAproximate));

}

for (int i = 0; i < ThreadsCount; i++)

{

threads[i].join();

}

threads.clear();

for (int i = 1; i < Calculations::RowsNumber; i++)

{

count = 0;

for (int j = 0; j < ThreadsCount; j++)

{

threads.emplace\_back(thread(Task3, i, count++, dataMatrixAproximate));

}

for (int j = 0; j < ThreadsCount; j++)

{

threads[j].join();

}

threads.clear();

}

cout << "Parallel time: "

<< std::chrono::duration\_cast<std::chrono::milliseconds>(chrono::system\_clock::now() - TimerStart).count() << endl;

ofstream fs;

fs.precision(15);

fs.open("output\_aproximate.txt");

for (int i = 0; i < Calculations::RowsNumber; i++)

{

fs << "{";

for (int j = 0; j < Calculations::ColumnsNumber; j++)

{

fs << dataMatrixAproximate[i][j] << ",";

}

fs << "}," << endl;

}

fs.close();

TimerStart = chrono::system\_clock::now();

for (int i = 0; i < Calculations::RowsNumber; i++)

{

for (int j = 0; j < Calculations::ColumnsNumber; j++)

{

dataMatrixExact[i][j] = Calculations::GetExactResult(Calculations::T \* i, Calculations::H \* j);

}

}

cout << "One thread time: "

<< std::chrono::duration\_cast<std::chrono::milliseconds>(chrono::system\_clock::now() - TimerStart).count() << endl;

fs.open("output\_exact.txt");

for (int i = 0; i < Calculations::RowsNumber; i++)

{

fs << "{";

for (int j = 0; j < Calculations::ColumnsNumber; j++)

{

fs << dataMatrixExact[i][j] << ",";

}

fs << "}," << endl;

}

fs.close();

Deviation(dataMatrixAproximate, dataMatrixExact, Calculations::RowsNumber, Calculations::ColumnsNumber);

getchar();

}