

*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

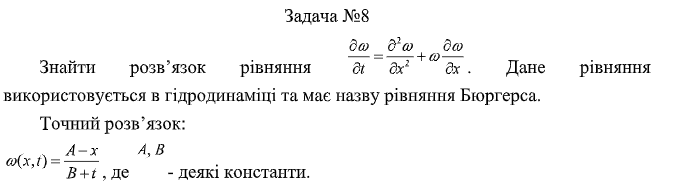
студент III курсу

групи ДА-62

*Данилюк Василь*

***Київ – 2019***

**Завдання**



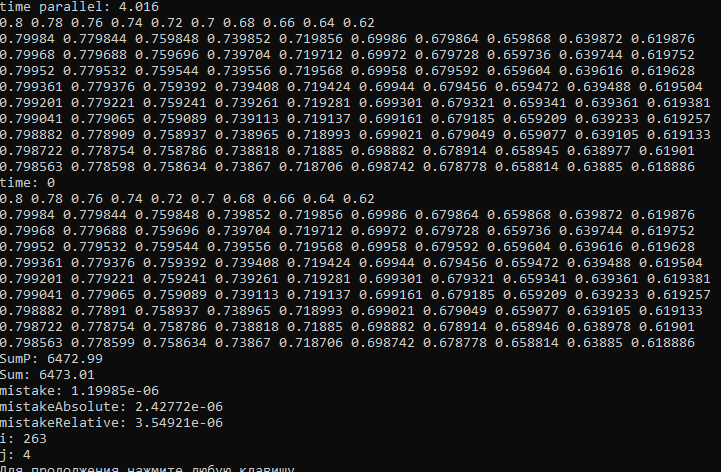
*+w*

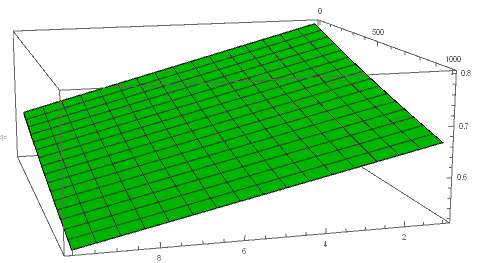
=

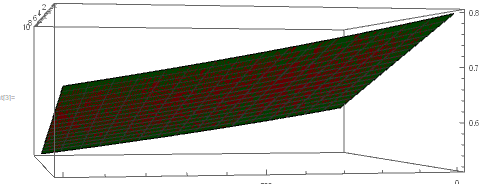
=

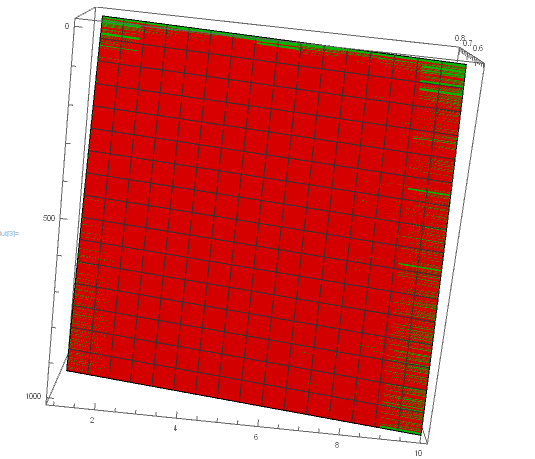
## Явний метод пошуку точок сітки

Крок h = 0.1; t = 0.001 маємо наступний результат:

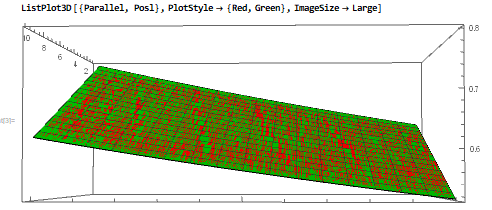








(Вигляд знизу)



## 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, а сам розв’язок був реалізований на мові C++. Згідно з отриманими результатами: із збільшенням кількості точок у сітці розв’язок стає більш точним.

#include "stdafx.h"

#include "Calculation.h"

Calculation::Calculation()

{

Calculation::a = 4;

Calculation::b = 5;

Calculation::tau = 0.001;

Calculation::h = 0.1;

}

double Calculation::getExactResolve(double x, double t)

{

double result = (Calculation::a - x) / (Calculation::b + t);

return result;

}

double Calculation::getApproximateSolution(int i, int k, double w\_i, double w\_i\_p, double w\_i\_m)

{

double result = ((w\_i\_m - 2 \* w\_i + w\_i\_p) \* tau / (h \* h)) + (w\_i\*tau\*(w\_i\_p - w\_i\_m) / (2 \* h)) + w\_i;

return result;

};

Calculation::~Calculation()

{

}

////////////////////////////////////

#pragma once

class Calculation

{

public:

int a;

int b;

double tau;

double h;

Calculation();

double getExactResolve(double x, double t);

double getApproximateSolution(int i, int k, double w\_i, double w\_i\_p, double w\_i\_m);

~Calculation();

};

/////////////////////////////////

// CourseWork2.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include "Calculation.h"

#include <iostream>

#include <fstream>

#include <thread>

#include <ctime>

#include <vector>

#include <string>

Calculation calc;

void countVal(double x, double t, double\*\* Arr, int from, int to, int column, int i);

void mistake(double \*\*array1, double \*\*array2, int row, int column);

void printArray(double \*\*array);

double writeArrayToFile(double \*\*array,std::string filename,int row, int column);

int main()

{

double duration;

std::clock\_t start;

int countThread = 4;

const int row = 1000, column=10;

double \*\*mainArray = new double \*[row];

double \*\*mainArray2 = new double \*[row];

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

{

mainArray[i] = new double[column];

mainArray2[i] = new double[column];

}

double x = 0;

double t = 0;

int from = 0;

int to = 0;

int temp = (column / countThread);

start = std::clock();

/////////////////////////////////

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

{

x = 0;

from = 0;

std::vector <std::thread> threadVector;

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

{

if (j == (countThread-1))

{

to = column;

}

else

{

to = temp\*(j+1);

}

double temp\_x = (x + calc.h) \* from;

threadVector.emplace\_back(std::thread(countVal, temp\_x,t,mainArray,from,to,column,i));

from = to;

}

for (int z = 0; z < threadVector.size(); z++)

{

threadVector[z].join();

}

t += calc.tau;

}

duration = (std::clock() - start) / (double)CLOCKS\_PER\_SEC;

std::cout << "time parallel: " << duration << std::endl;

printArray(mainArray);

///

double sumParallel = writeArrayToFile(mainArray, "array.txt", row, column);

start = std::clock();

t = 0;

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

{

x = 0;

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

{

mainArray2[i][j] = calc.getExactResolve(x, t);

x += calc.h;

}

t += calc.tau;

}

duration = (std::clock() - start) / (double)CLOCKS\_PER\_SEC;

std::cout << "time: " << duration << std::endl;

printArray(mainArray2);

double sum = writeArrayToFile(mainArray2, "array2.txt", row, column);

std::cout << "SumP: " << sumParallel << std::endl;

std::cout << "Sum: " << sum << std::endl;

std::cout << "mistake: " << std::abs(sum-sumParallel)/(row\*column) << std::endl;

mistake(mainArray2, mainArray, row, column);

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

delete[]mainArray[i];

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

delete[]mainArray2[i];

system("pause");

return 0;

}

void countVal(double x, double t, double\*\* Arr, int from, int to, int column, int i)

{

for (int j = from; j < to; j++)

{

if (i == 0 || j == 0 || j == (column - 1)) {

Arr[i][j] = calc.getExactResolve(x, t);

}

else {

Arr[i][j] = calc.getApproximateSolution(i, j, Arr[i - 1][j], Arr[i - 1][j + 1], Arr[i - 1][j - 1]);

}

x += calc.h;

}

}

void mistake(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 = std::abs(array1[i][j] - array2[i][j]);

if (temp\_val > absolute) {

absolute = temp\_val;

t\_i = i;

t\_j = j;

}

}

}

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

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

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

std::cout << "i: " << t\_i << std::endl;

std::cout << "j: " << t\_j << std::endl;

}

void printArray(double \*\*array)

{

for (int i = 0; i < 10; i++)

{

for (int j = 0; j < 10; j++)

{

std::cout << array[i][j] << " ";

}

std::cout << std::endl;

}

}

double writeArrayToFile(double \*\*array, std::string filename,int row,int column)

{

std::ofstream out;

double sum = 0;

out.open(filename, std::ios\_base::trunc);

out << "{";

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

{

out << "{";

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

{

sum += array[i][j];

if (j == column - 1) {

out << array[i][j];

}

else {

out << array[i][j] << ",";

}

}

out << "},";

}

out << "}";

out.close();

return sum;

}