Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Кафедра автоматизації проектування енергетичних процесів і систем

**КУРСОВА РОБОТА**

з дисципліни «Чисельні методи в інформатиці»

на тему: «Розв’язання диференційних рівнянь у частинних похідних»

Студента групи ТМ-61

спеціальність122 «Комп'ютерні науки»

спеціалізація «Інформаційні технології моніторингу довкілля»

Новоселов С.Є.

Керівник доц., к.т.н. Михайлова І.Ю.

Варіант: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Національна оцінка: \_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_ к.т.н. Михайлова І.Ю.

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ - 2018 рік

**Національний технічний університет України**

**“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність 122 «Комп’ютерні науки»

Спеціалізація «Інформаційні технології моніторингу довкілля»

**ЗАВДАННЯ**

**НА КУРСОВУ РОБОТУ СТУДЕНТУ**

Новоселову Сергію Євгенійовичу

1. Тема роботи «Розв’язання диференційних рівнянь у частинних похідних»

керівник курсової роботи – Михайлова Ірина Юріївна, к.т.н.

2. Строк подання студентом роботи – 25 травня 2018 р.

3. Вихідні дані до проекту (роботи): нелінійна нестаціонарна одновимірна задача для ДРЧП.

4. Зміст пояснювальної записки курсової роботи (перелік питань, які потрібно розробити) – Постановка задачі розв’язання ДРЧП, а саме: фізична та математична постановка задачі. Математичні методи розв’язання ДРЧП: загальна характеристика методів розв’язання, типи різницевих сіток, явна схема та схема Кранка-Ніколсона. Опис програмної реалізації: структура програмної системи, класи – MathHandler,SplineChart,SurfaceGraph. Результати досліджень для явної схеми та схеми Кранка-Ніколсона. Висновки, список використаних джерел, додаток

5. Дата видачі завдання – 12 лютого 2018 р.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів виконання курсової роботи | Строк виконання етапів роботи | Примітки |
|  | Термін представлення програмного продукту на «відмінно» | 10.05.2018 |  |
|  | Термін доопрацювання програми на «відмінно» | 25.05.2018 |  |

Студент \_\_\_\_\_\_\_\_\_ Новоселов С.Є.

( підпис ) (прізвище та ініціали)

Керівник курсової роботи \_\_\_\_\_\_\_\_\_ Михайлова І.Ю.

( підпис ) (прізвище та ініціали)

**АНОТАЦІЯ**

**АНОТАЦІЯ**

В даному документі реалізована і описана курсова робота з дисципліни «Чисельні методи в інформатиці-1» на тему «Розв’язання диференційних рівнянь у частинних похідних

Мета курсової роботи - навчити студента розв'язувати наближеними методами за допомогою комп'ютерної техніки математичні задачі, що виникають в інженерній практиці в процесі моделювання та проектування.

Задача розв’язанна методом скінчених різниць, а саме: явна схема та схема Кранка-Ніколсона.

Для кращого розуміння використаних методів, короткої теорії, та аналізу результатів роботи програми використовуюється 14 ілюстрацій.

Програма була написана на мові C++ з використанням фреймворку Qt версії 5.10 та його стандартних класів. В результаті роботи створена програма, яку можна застосувати для будь-якого ДРЧП, потрібно лише замінити математичну основу задачі.

Summary

In this document, the course work on discipline "Numerical methods in computer science - 1" on the topic "Solvic differential equations in partial derivatives".

The main goal of course work - to teach the student to use approximate methods with the help of computer technology mathematical problems that arise in engineering practice in the process of modeling and design.

The problem has solved by the method of completed differences, namely: explicit and implicit scheme.

For better understanding of the methods have used used, short theory, and analysis of the results of the program, 14 illustrations are used.

The program has been written in C++ using Qt ver. 5.10 framework and its default classes. As a result of the work have created a program that can be applied to any DTP, you only need to replace the mathematical basis of the task.

В даному документі реалізована і описана курсова робота з дисципліни «Чисельні методи в інформатиці-1» на тему «Розв’язання диференційних рівнянь у частинних похідних». А саме представлені теоретичні відомості, що стосуються теми роботи і елементів, що були використані в ній, використані методи, програмний код продукту, що є кінцевим об’єктом роботи і опис самої програми.

Мета курсової роботи - навчити студента розв'язувати наближеними методами за допомогою комп'ютерної техніки математичні задачі, що виникають в інженерній практиці в процесі моделювання та проектування.

Індивідуальним завданням запропоновано розв’язати одновимірну нестаціонарну нелінійну задачу для диференційних рівнянь у частинних похідних, вигляду:

Точний розв’язок:



де , - деякі константи.

Задача розв’язанна методом скінчених різниць, а саме: явна та неявна схема.

Для кращого розуміння використаних методів, короткої теорії, та аналізу результатів роботи програми використовуюється 14 ілюстрацій.

Програма була написана на мові Python 3.6 з використанням модулів: math, matplotlib, scipy. В результаті роботи створена програма, яку можна застосувати для будь-якого ДРЧП, потрібно лише замінити математичну основу задачі.

**ВСТУП**

**ЗМІСТ**

Вступ 5

1. Постановка задачі розв’язання ДРЧП 6

1.1. Фізична постановна задачі нагріву стержня 6

1.2. Математична постановна задачі розв’язання ДРЧП 6

2. Математичні методи розв’язання ДРЧП 8

2.1. Загальна характеристика методів розв’язування 8

2.2. Типи різницевих сіток 9

2.3. Явна схема 11

2.4. Неявна схема 12

3. Опис програмної реалізації 15

3.1. Структура програмної системи 15

3.1.2. Модуль common\_functions 15

3.1.3. Модуль config 15

3.1.4. Модуль first\_part 16

3.1.5. Модуль second\_part 16

4. Результати тестового розрахунку 18

4.1. Результати досліджень для явної схеми 18

4.2. Результати досліджень для неявної схеми 19

Висновки 23

Список використаних джерел 24

Додаток. Текст програми 25



Математичне моделювання з використанням комп’ютерної техніки є загальноприйнятим засобом дослідження різноманітних природних явищ та створення нових технічних пристроїв. Комп’ютерні системи проектування скорочують строки проектування та підвищують якість нових технічних систем.

В багатьох випадках для дослідження природних явищ або штучних систем використовується їх описання за допомогою функцій двох або більше змінних.

Ці функції виявляються розв’язками диференціальних рівнянь з частинними похідними.

Теоретичною основою математичного моделювання є фундаментальні закони фізики: збереження енергії, маси та інші. Аналітичні методи розв’язування задач для ДРЧП можуть бути використані лише в деяких випадках і на практиці здебільше застосовують обчислювальні методи з використанням комп’ютерної техніки.

Найчастіше, при моделюванні явищ, пов’язаних з динамікою рідин та газів, статикою та динамікою деформування пружних тіл, електростатикою, електродинамікою, тепловими та дифузійними процесами, оптикою, квантовою механікою, загальною теорією відносності, використовуються диференціальні рівняння з частинними похідними першого або другого порядку, а також системи таких рівнянь.

**1. Постановка задачі розв’язання ДРЧП**

**1.1 Фізична постановка задачі нагріву стержня**

Є одновимірний стержень, розміром . Стержень нерівномірно нагрітий. За умовами задачі задано функцію, яка характеризує розподіл температури в стержні і має вигляд:

(1.1)

де – деяка константа.



Рисунок 1.1 – Графічна постановка задачі

З моменту часу в стержні починається процес нагрівання. Необхідно знайти температуру стержня у вигляді функціональної залежності у визначений час.

**1.2 Математична постановка задачі розв’язування ДРЧП**

Розробити програмне забезпечення для розв’язування одновимірної нестаціонарної нелінійної задачі (1.1) для диференційних рівнянь у частинних похідних.

Розрахункова область: .

Задачу необхідно розв’язати двома способами.

1. З використання фіксованої рівномірної сітки явним методом.
2. З використанням змінної рівномірної сітки методом Кранка-Ніколсона.

Для перевірки результату та отримання початкової та крайових умов задано точний розв’язок:

, (1.2)

де - деякі константи.

Для обох способів розв’язування задачі необхідно вивести таблицю точного та наближеного розв’язків для 4-6 точок по осі Ox для кожного з 5-10 часових шарів. Отриманий розв’язок зобразити графічно у вигляді 3D графіків та їх перерізів по осі часу. Вивести максимальні абсолютну та відносну різниці між точним та наближеним розв’язками. Для фіксованої сітки кількість вузлів в обох координатних напрямках взяти 10-100.

Для змінної рівномірної сітки розв’язати задачу з точністю ε = 0.01. Побудувати графіки залежності від номера кроку наступних параметрів: величин просторового та часового кроків, реальної локальної похибки, різниці між точним та наближеним розв’язками. Зобразити сітку розрахункової області. Порівняти ефективність розрахунку з фіксованою та адаптивною сітками.

Програмне забезпечення розробити з використанням мови C++ з використанням фреймворку Qt 5.10 та його класів.

**2. Математичні методи розв’язування ДРЧП**

Для розв’язання ДРПЧ у якості констант покладемо значення:

a = 0.01, А = В = 2, C = 1.

Тоді рівняння (1.1) прийме вигляд:

(2.1)

З точного розв’язку рівняння знайдемо початкову умову:

(2.2)

та крайові умови:

(2.3)

(2.4)

**2.1. Загальна характеристика методів розв'язування**

Аналітичні методи розв’язування ДРЧП існують лише для деяких їх класів, тому зараз найчастіше використовують числові методи з використанням комп’ютерної техніки. Існує досить багато обчислювальних методів розв’язування ДРЧП, які різняться способом заміни диференційної задачі набором алгебричних. Умовно їх розділяють на дві великі групи: методи скінчених різниць, які використовуються в даній роботі, та методи скінчених елементів. Незважаючи на різноманіття числових методів для розв’язання ДРЧП, вони мають деякі спільні риси. При їх використанні:

* неперервну область зміни аргументів шуканої функції замінюють кінцевою (дискретною) множиною точок (вузлів), яку називають різницевою сіткою;
* на сітці замість функцій неперервного аргументу розглядають функції дискретного аргументу, визначені у вузлах сітки, які називають сітковими функціями;
* диференційне рівняння для кожного моменту модельного часу замінюють системою алгебричних рівнянь (різницевими рівняннями), початкові і крайові умови також замінюють різницевими початковими і крайовими умовами.

У результаті одержують різницеву крайову задачу у вигляді однієї або багатьох систем алгебричних рівнянь. Розв'язком різницевої задачі є сіткова функція, визначена у вузлах сітки, тобто на дискретній множині точок, або кусково-неперервна функція для методу скінчених елементів.

В методах скінчених різниць алгебричний аналог диференційної задачі одержують безпосередньо з диференційного рівняння застосуванням формул чисельного диференціювання або з вимоги виконання законів збереження енергії, маси, кількості руху (методи контрольного об’єму, балансу та ін.).

Основними характеристиками обчислювальних методів розв’язування ДРЧП виступають:

1. збіжність наближеного розв’язку до точного при згущенні сітки;
2. точність апроксимації диференційної задачі різницевою;
3. стійкість розв’язку до флуктуацій вхідних даних;
4. консервативність, тобто виконання законів збереження енергії, маси;
5. монотонність, тобто відповідність наближеного розв’язку фізичному явищу, що моделюється, з точки зору кількості локальних екстремумів та його осциляцій;
6. економічність, яка характеризується лінійним характером залежності кількості операцій, які потрібні для розв’язання алгебричних рівнянь, від кількості вузлів сітки.

Основними етапами методу скінчених різниць є:

* 1. Дискретизація області, тобто побудова різницевої сітки.
  2. Заміна диференційної задачі алгебричною у вигляді систем алгебричних рівнянь.
  3. Розв’язування систем алгебричних рівнянь.

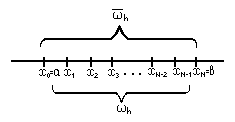
**2.2. Типи різницевих сіток**

Для задачі, яка описана в даній курсовії роботі використовуються наступні різницеві сітки:

* рівномірна незмінна, коли часовий крок та просторовий крок залишаються незмінними протягом всього виконання програми;
* змінна рівномірна, коли змінним є крок за часом *t*, а просторові кроки є однаковими в кожному координатному напрямку для фіксованого *t* і змінюються при переході на наступний часовий шар;

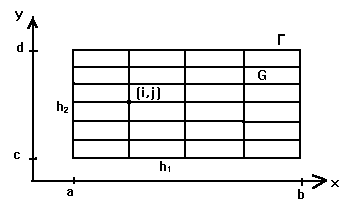
У методі скінченних різниць наближений розв'язок визначається у вузлах сітки. Для одержання рівномірної сітки на відрізку [*a,b]* розіб'ємо його на *n* рівних частин. Відстань між сусідніми вузлами *h=xi*-*xi*-1=(*b-a*)/*n* називають кроком сітки, а точки поділу *xi=a+ih* - вузлами (рис. 2.1а).

Точки *x*1,*x*2,...,*xn*-1, що не належать кінцям відрізка, називають внутрішніми точками сітки і утворюють множину ω*h*={*xi=a+ih*, *i*=1,2,...,*n*-1}. Якщо до цієї множини додати граничні точки *x*0=*a* та *xn*=*b*, то отримаємо множину ={*xi=a+ih*, *i*=0,1,...,*n*}.



а) б)

Рисунок 2.1 — Рівномірні сітки на відрізку (а) та площині (б)



Для одержання рівномірної сітки на площині розглянемо функцію двох аргументів *W(x,y)* з областю визначення у вигляді прямокутника G з межею Г: G\*={*a<x<b, c<y<d*} (рис.2.1б). Розіб’ємо відрізки [*a,b*] вісі *Ox* та [*c,d*] вісі *Oy* відповідно на *n*1 і *n*2 частин з величинами кроків *h*1=(*b-a*)/*n*1 та *h*2=(*d-c*)/*n*2. Через точки поділу проведемо прямі, паралельні відповідним осям. Внаслідок перетину цих прямих одержимо вузли (*i,j*), визначені координатами (*xi,yj*), які утворюють сітку ω\**h*={(*xi,yj*)∈G\*}. Сусідніми вузлами сітки називають вузли, які лежать на одній прямій, відстань між якими дорівнює кроку сітки (*h*1 чи *h*2). Вузли сітки, які лежать на межі Г області G, називають граничними вузлами, а решту вузлів - внутрішніми. Оскільки при постановці задачі часто граничні умови відомі, то в граничних вузлах сіткову функцію також можна вважати відомою.

Змінна рівномірна сітка – це модифікація рівномірної незмінної сітки. Головна відмінність цього типу сітки заключається в тому, що з плином часу просторовий крок змінюється автоматично. Тобто, цей тип сітки дозволяє зменшити машинний час розрахунку і в той самий час досягається наперед задана точність розрахунку. Рисунок 2.2 представлений для кращого розуміння.

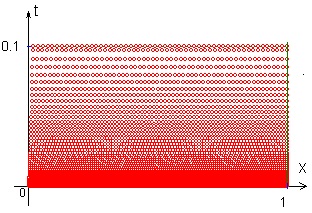


Рисунок 2.2 — Розрахункова область для змінної рівномірної сітки

**2.3 Явна схема**

Розіб’ємо область визначення шуканої функції рівномірною сіткою. Будемо шукати розв’язок для дискретних значень. Позначимо через наближений розв’язок в точці , через  крок у напрямку *Ох.*

Для апроксимації похідних по змінній *х* у випадку рівномірної сітки з кроком *h* будемо використовувати формули другого порядку точності:

, (2.3.1)

 (2.3.2)

У випадку рівномірної сітки різницеві рівняння мають вигляд:

(2.3.4)

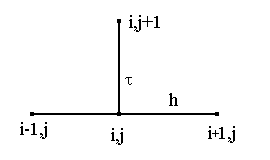
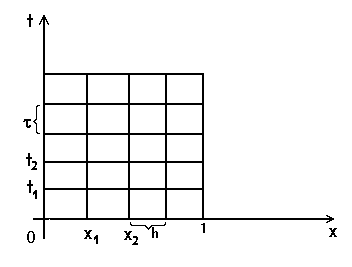
Точні значення невідомої функції *W(x,t)* у вузлах позначимо через *Wij=W(xi,tj)*. Оскільки будемо замінювати диференційну задачу різницевою, то в подальшому знайдемо розв’язок у вигляді сіткової функції *Wij*, яка задовольняє різницеву схему, але не співпадає з точним розв’язком *Wij* вихідної задачі (2.1), де j – часовий шар, i – вузол сітки

Виразимо звідси невідому величину і одержимо формулу, яку необхідно використовувати для кожного з (*n*-1) внутрішніх вузлів поточного часового шару у випадку рівномірної сітки:

(2.3.4)

*i*=1,2,…,*n*-1; *j*=0,1,…, *N*-1.

Таким чином, в цьому рівнянні лише одна невідома величина , яку легко обчислити за явною схемою (рис. 2.3).



а) б)

Рисунок 2.3 — Розрахункова область для одновимірного рівняння

Для явної схеми використовується рівномірна фіксована сітка, коли крок по та по залишається незмінним протягом всього виконання програми.

**2.4 Схема Кранка-Ніколсона**

Для одного і того ж рівняння (2.1) можна побудувати різні різницеві схеми. Знову розглянемо задачу (2.1). Підготовка до розв’язування ріняння виконується так само, як і у випадку явної схеми. Замінимо похідні різницевими аппроксимаціями.

Одержимо різницеві рівняння:

(2.4.1)

*де* *i*=1,2,...,*n*-1.

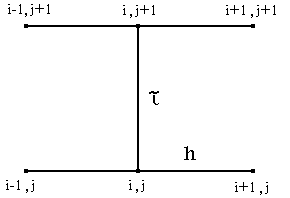


Рис.12 - Шаблон схеми Кранка-Ніколсона

Кожне рівняння містить три невідомих величини , ,  . Величина  відома з попереднього кроку по часу. Отримане рівняння є нелінійним і разом ці рівняння для всіх  утворюють систему  нелінійних алгебричних рівнянь (НАТР) з  невідомими. Але значення  для крайніх точок  відомі з крайових умов. Будемо розв’язувати систему НАТР методом Ньютона.

. (2.4.2)

На кожній його ітерації формується і розв’язується система лінійних алгебричних рівнянь (СЛАР)

 (2.4.3)

відносно приростів невідомих, де  — матриця Якобі , *s*=0,1,.... Параметр *s* пов’язаний з номером ітерації.

У першому і останньому рівняннях цієї системи значення функції в граничних точках *x*0, *xn* відомі з крайових умов:

 (2.4.4)

СЛАР, яка формується на i-тій iтерації Ньютона:

 (2.4.5)

(2.4.6)

(2.4.7)

(2.4.7)

(2.4.8)

Таким чином, у методі Кранка-Ніколсона на кожному часовому шарі потрібно розв’язувати СЛАР (2.4.5). Система є тридіагональною і розв’язується за допомогою метода прогонки.

Також в цій частині програми використовується тип сітки – змінна рівномірна, коли змінним є крок за часом *t*, а просторові кроки є однаковими в кожному координатному напрямку для фіксованого *t* і змінюються при переході на наступний часовий шар.

**3. Опис програмної реалізації**

**3.1. Структура програмної системи**

Програмне забезпечення складається з 2 програм, які схожі, але мають різну реалізацию метода MathHandler::Approximate() для явної схеми та схеми Кранка-Ніколсона. Кожна программа містить \*.h та \*.сpp файли для таких класів: MathHandler,SplineChart,SurfaceGraph. Також main.cpp для обробки процесу програми у операційній системі (відтворення вікон та ін). Програма написана на С++ з використанням Qt 5.10, зокрема: QList, QtDataVisualization, QWidgets, QtCharts, QtGui та ін.

**3.1.2 Клас MathHandler**

Цей клас созданий для обробки математичних дій, а саме – отримування усіх значеннь, необхідних для побудови графіку, виводу значеннь.

void Approximate() – виконує розрахунок за допомогою схеми.

void Exact() – виконує розрахнок за допомогою точної формули. Оба з цих методів заповнюють dataArray, який є public.

QtDataVisualization::QSurfaceDataArray \* dataArray – масив трьохвимерних векторів, необхідний для побудуви 3D графіку.

В private полі усі змінні та константи : const double const\_A =2; const double const\_B = 2; const double a =0.01; double e =0.01; double h = 0.1; та ін.

QtDataVisualization::QSurfaceDataRow \***lagranzDisplayInterpolation**(QList<double> \*Layer,double tau) – інтерполяція Лагранжа, необхідна для відображення 3D графиків, у яких кількість площин на кожному кроці повинна бути однаковою.

double \* **systemSolver**(double \*\*a, double \*y, int n) - вирішуе СЛАР методом прогонки(Томаса) та повертає одновимірний масив відповідей. Також є метод для заповнення СЛАР, та для розрахунку α та β.

**3.1.3. Клас SplineChart**

Цей клас созданий для побудови двохвимірних графиків необхідних для дослідження методу.

QtCharts::QSplineSeries \*series – має у собі n точок QPointF(double,double)

QtCharts:: QChart \*chart – віджет графіку

Також клас має два слоти, які слухають кнопки, та при натисненні кнопок “>>>” та “<<<” виконують певні дії. Наприклад, при долідженні локальної похибки, змінюють номер часового кроку.

public slots:

void **next**();

void **prev**();

**3.1.4. Клас SurfaceGraph**

Цей клас будуе Q3DSurface по точках, отриманих в MathHandler. Має слоти для сигналів слайдерів, які змінюють масштаб. Основна частина класу – стандартна та взята з офіційної документації.

**3.1.5. main.cpp**

У функції main відтворюється вікно з 3D графіком, елементи керування – радіобаттони та слайдери.

void **displayGrid**(QSize screenSize ); - виводить приблизні значення, точні значення, похибку для певної кількості кроків.

void **displayChart**();- створює об’єкт SplineChart та у окремому вікні виводить 2D графік.

**4.Результати тестового розрахунку**

**4.1.Результати досліджень для явної схеми**

Результати виконання програми у якій використовується явна схема представлені на рисунку 4.1 3D графік розрахункової області явної схеми представления на рисунку 4.2



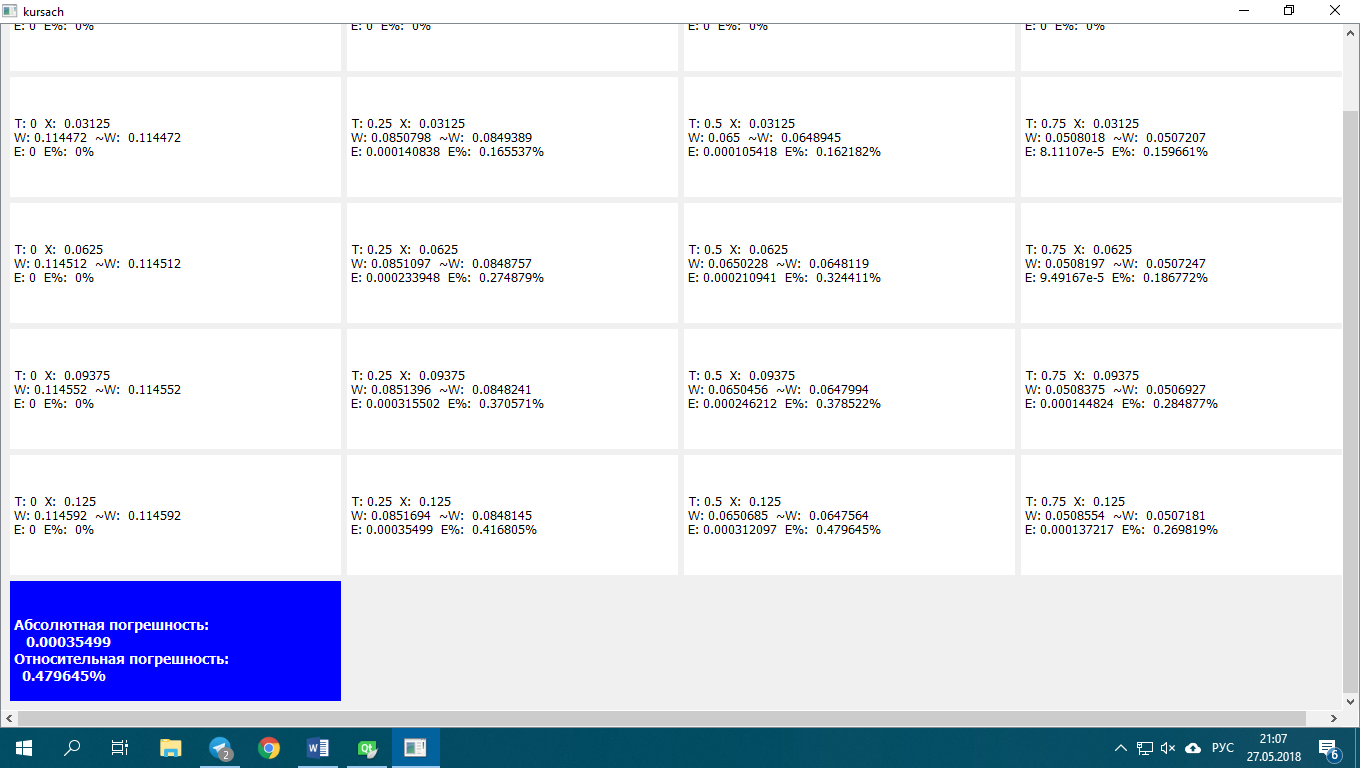


Рисунок 4.1 — Скріншот виконання програми.

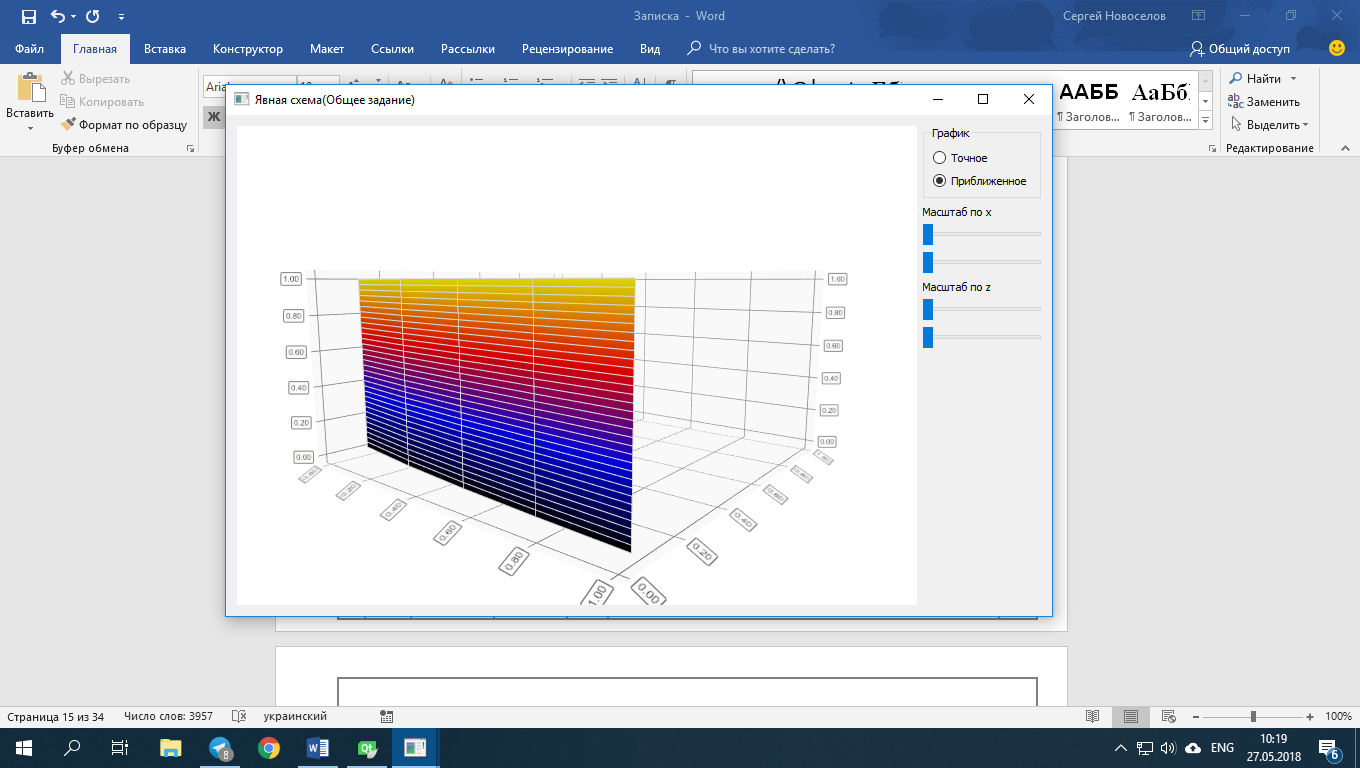


Рисунок 4.2 — 3D графік розрахункової області явної схеми.

**4.2.Результати досліджень для неявної схеми**

Результати виконання програми у якій використовується явна схема представлені на рисунку 4.3. 3D графік розрахункової області явної схеми

представления на рисунку 4.8. На графіках 4.4-4.7 представлені залежності часового кроку, просторового кроку, локальної похибки, різниці між точним і наближеним розв’язком від номеру k-того кроку.



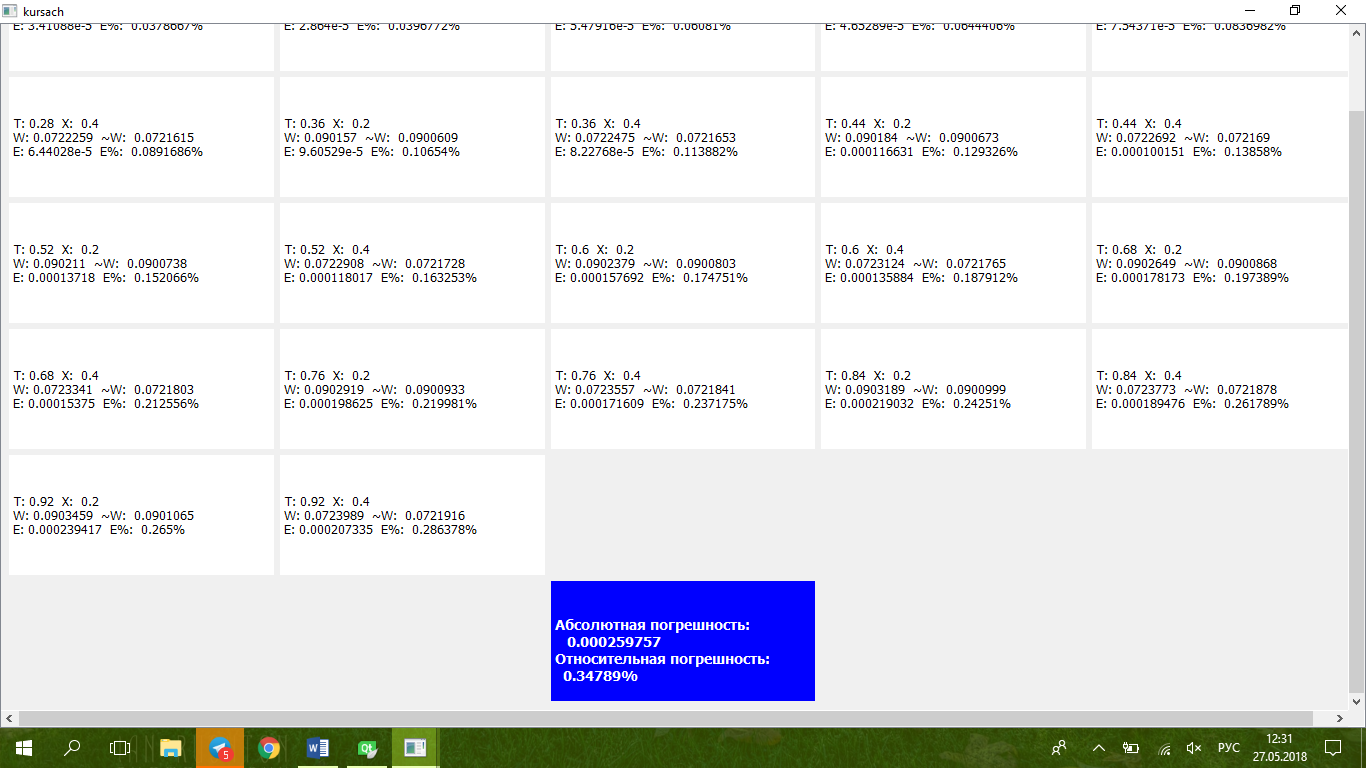


Рисунок 4.3 — Скріншот виконання програми

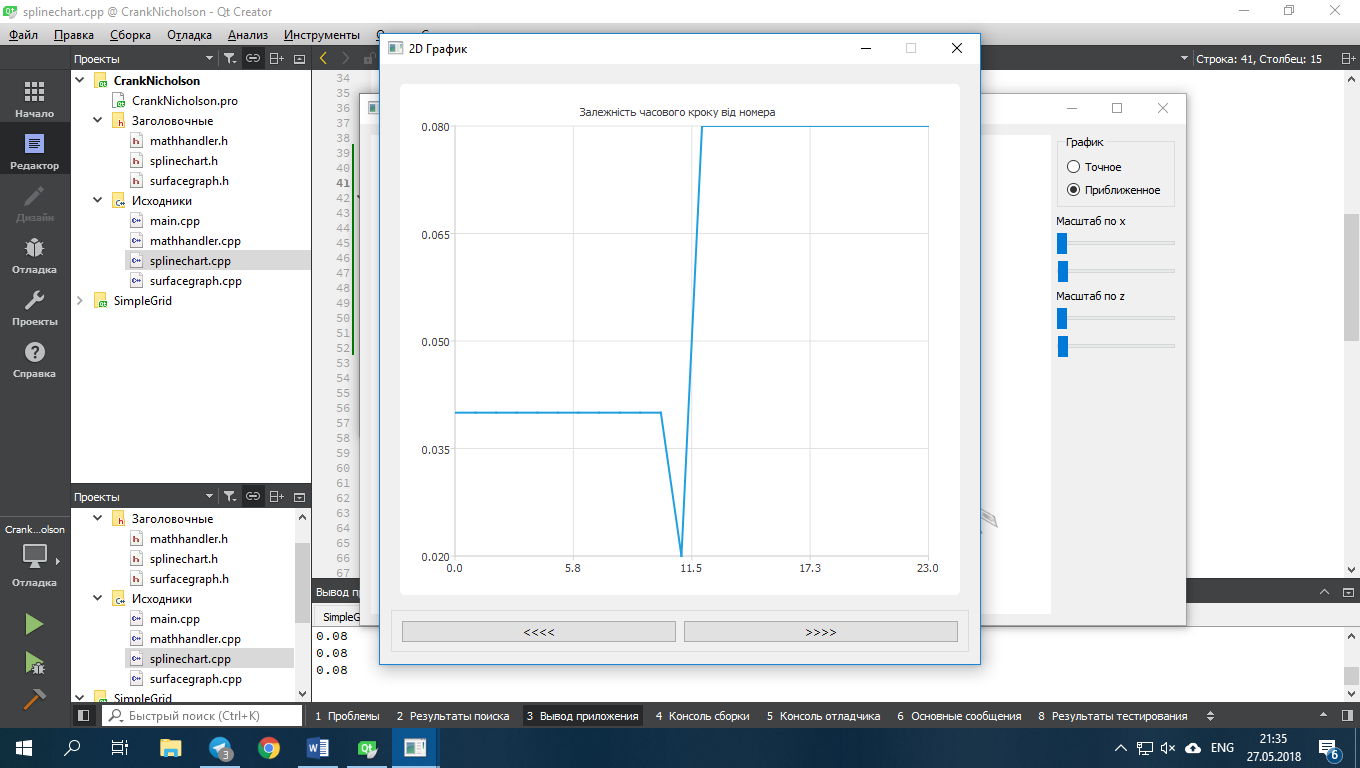


Рисунок 4.4 — Залежність часового кроку від номера кроку

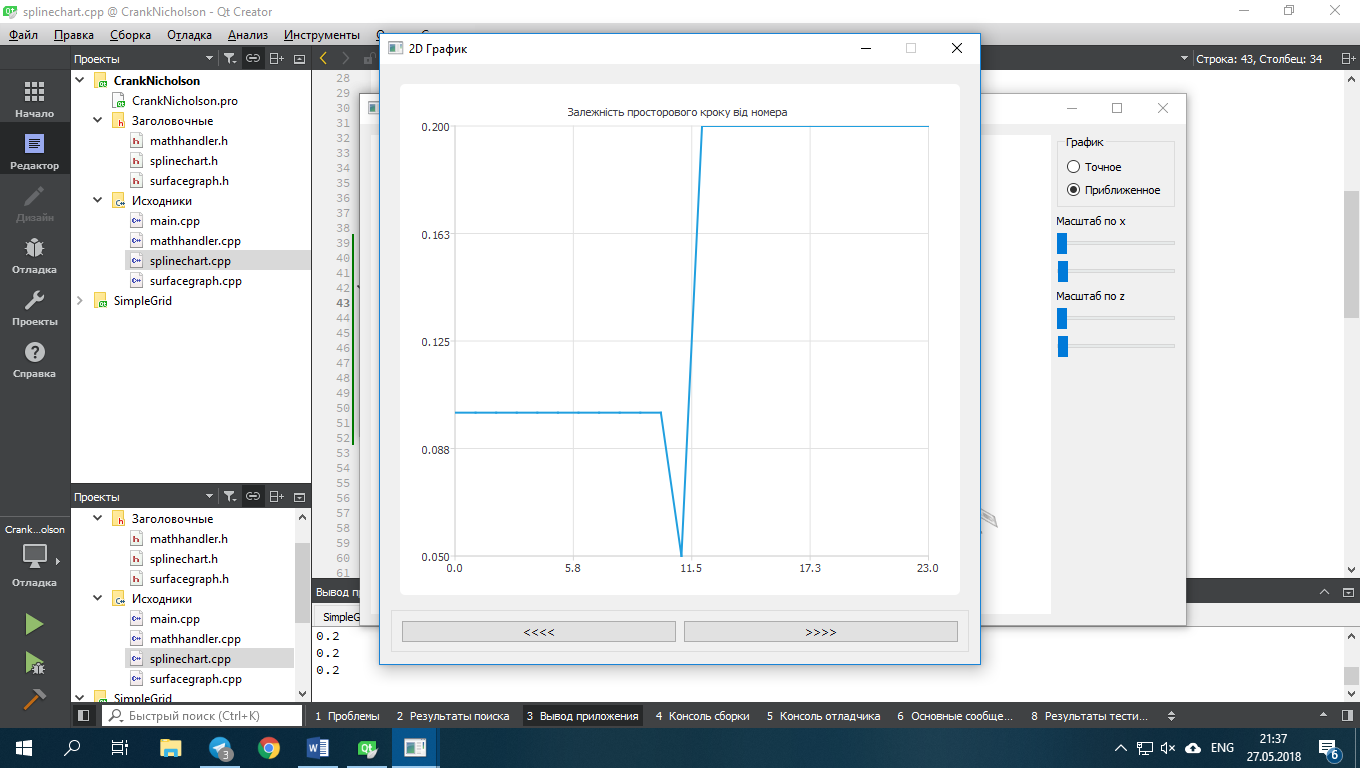


Рисунок 4.5 – Залежність просторового кроку від номера кроку

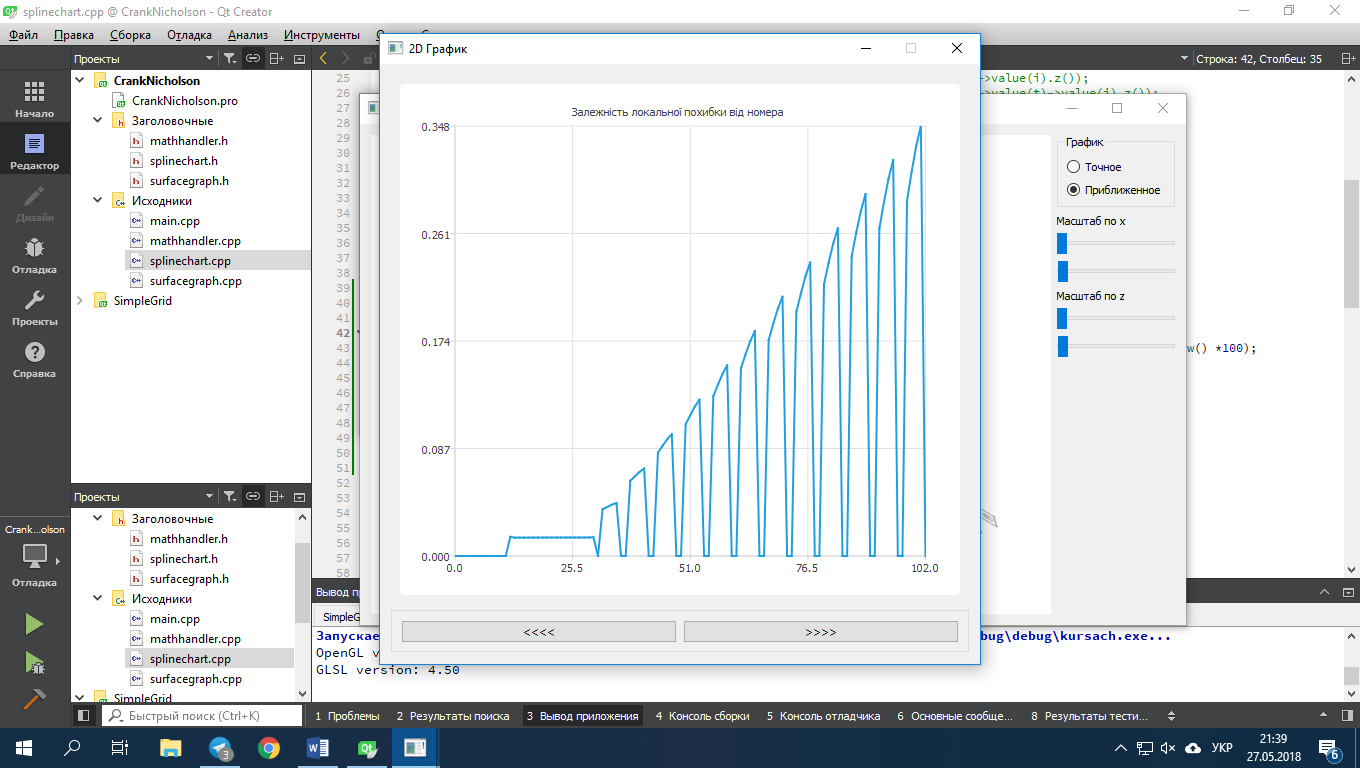


Рисунок 4.6 – Залежність локальної похибки від номера кроку

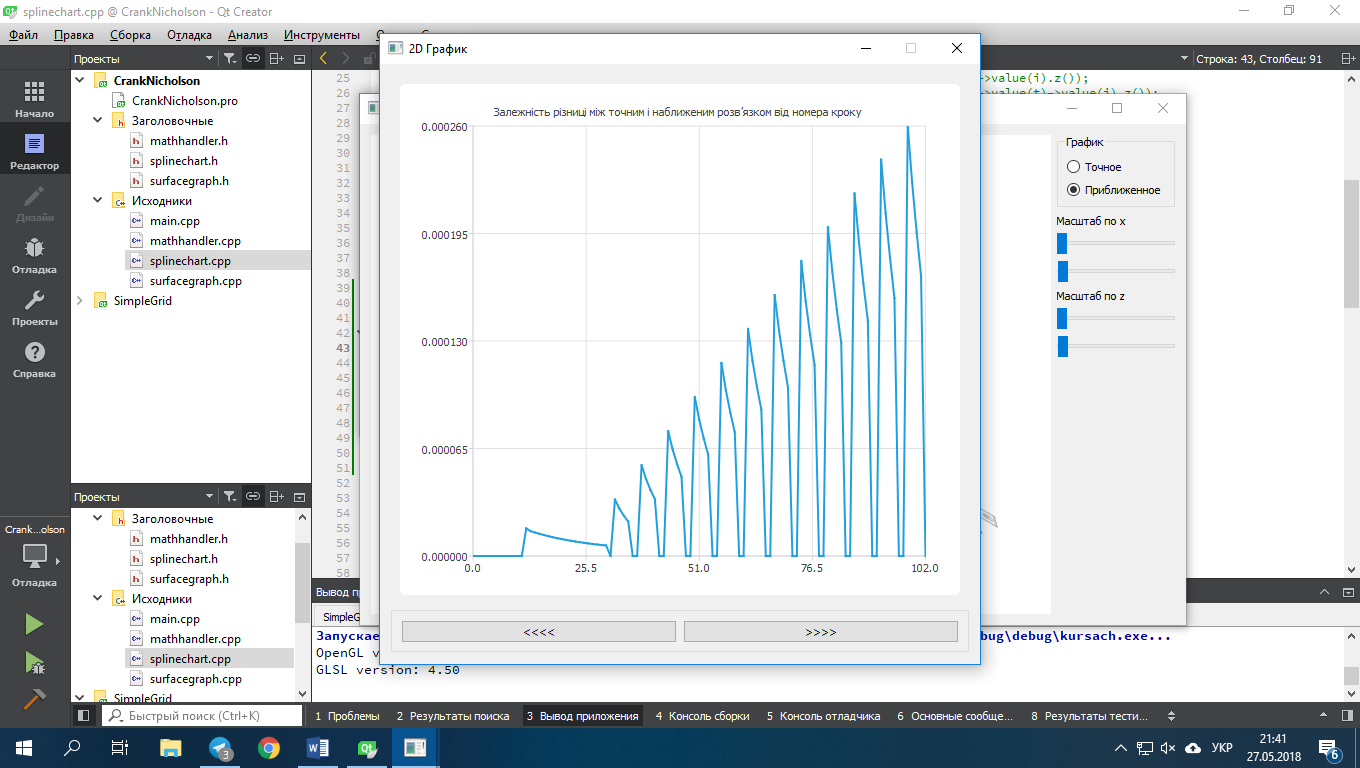


Рисунок 4.7 – Залежність різниці між точним і наближеним розв’язком від номера кроку

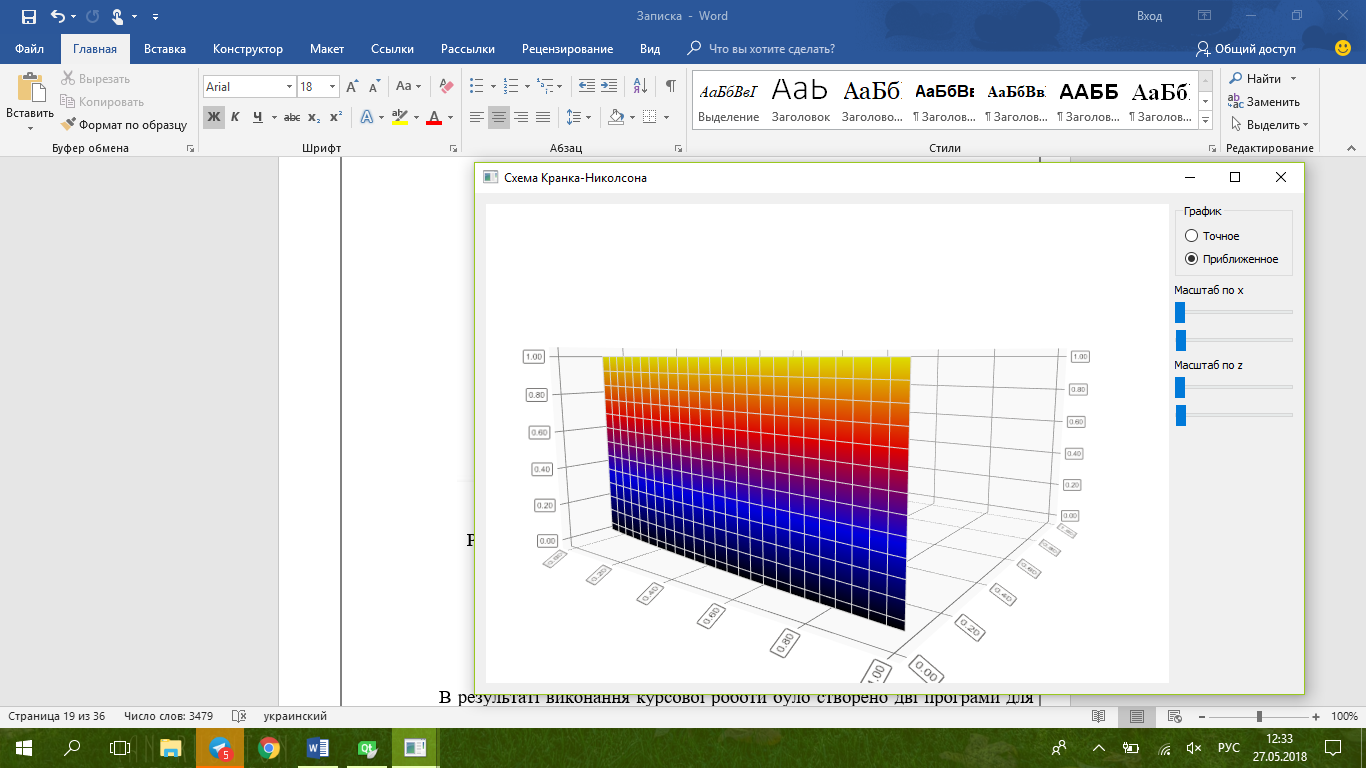


Рисунок 4.8 — 3D графік розрахункової області схеми Кранка-Ніколсона.

**ВИСНОВКИ**

В результаті виконання курсової роботи було створено дві програми для розв’язання ДРЧП, що реалізують явну схему і схему Кранка-Ніколсона. У явній схемі максимальна відносна похибка дорівнює 0.479645. При тій самій допустимій похибці в схемі Кранка-Ніколсона відносна похибка дорівнює 0.34789. Ще одна перевага схеми Кранка-Ніколсона над явною заключається в тому, що вимога стійкості явної схеми накладає обмеження на співвідношення кроків по аргументу і по часу : . Однак,схема Кранка-Ніколсона вимагає більше машинних обчислень, тому зростає машинний час. Також в другій програмі (схема Кранка-Ніколсона) використовується зовсім інший тип сітки – змінна рівномірна сітка. Перевага цього типу сітки над рівномірною незмінною в тому, що крок і підбирається автоматично при переході на наступний часовий шар. Це дозволяє зменшити кількість обчислень, але водночас досягається наперед задана точність. Рівномірна незмінна сітка не може змінювати свої кроки, тому це вимагає більше машинного часу для виконання.

**Список використаних джерел**

Лук’яненко С.О. Числові методи розв’язування диференційних рівнянь.–“Видавництво «Політехніка»”

<http://algowikiproject.org/ru/Метод_Ньютона_для_систем_нелинейных_уравнений>

Числові методи в інформатиці: навч. посіб./ С. О. Лук’яненко; НТУУ “КПІ”.

**ДОДАТОК**

***Текст програми***

**MathHandler.cpp**

void MathHandler::**Approximate**(){

double r=(1/h\_max)+1;

int rowSteps = r;

QList <double>\*lastLayer= new QList <double>();

QList <double>\*currentLayer= new QList <double>();

X\_Y\_Z\_W\_ExW = new QList <QVector4D>;

dataArray = new QtDataVisualization::QSurfaceDataArray;

ExactdataArray = new QtDataVisualization::QSurfaceDataArray;

double \*\*A = new double\* [rowSteps];

for (int count = 0; count < rowSteps; count++) A[count] = new double[rowSteps];

double \*B = new double [rowSteps];

double \*X= new double[rowSteps];

double x=0;

bool resetLayer;

// начальные условия

while(x<=1){

lastLayer->append(pow(const\_A,0.75)\*pow((x+const\_B)\*(x+const\_B+1),-3/(double)2));

currentLayer->append(pow(const\_A,0.75)\*pow((x+const\_B)\*(x+const\_B+1),-3/(double)2));

x+=h;

}

//Отобразим слой - при необходимости интерполируем

\*dataArray << lagranzDisplayInterpolation(lastLayer,0);

for(double tau =t; tau<=1;tau+=t){

//-------------------Метод Кранка-Николсона-------------------------

for (int count = 0; count <100; count++){

double max\_delta\_x =0;

//Заполнили матрицы

fillMatrix(A,B,rowSteps,lastLayer,currentLayer,tau);

//решили систему

X=systemSolver(A,B,rowSteps);

//Нужны ли еще итерации?

for (int c = 0; c < rowSteps; c++){

currentLayer->replace(c,currentLayer->value(c)+X[c]);

if(max\_delta\_x<X[c]) max\_delta\_x = X[c];

}

if (max\_delta\_x< e) count=101;

}

//перейдем на этот же слой с h/2 t/2;

resetLayer =false;

QList <double>\*HalfTauHalfHCurrentLayer= OnThisLayer\_HalfTauHalfH(lastLayer,tau);

//проверим необходимость сгущения сетки

for(int i =0,q=0; i<rowSteps; i++,q+=2){

if(fabs(1/(double)3\*fabs(currentLayer->value(i)-HalfTauHalfHCurrentLayer->value(q))) >e)

if(h/2 > h\_min && t/2 > t\_min){

//сгущенная опорная функция

lastLayer =InterpolateLayer(lastLayer,h/2);

\*currentLayer=\*lastLayer;

h/=2;

t/=2;

tau-=t;

r=(1/h)+1;

rowSteps = r;

//TODO:освободить память

A = new double\* [rowSteps];

for (int count = 0; count < rowSteps; count++) A[count] = new double[rowSteps];

B = new double [rowSteps];

X= new double[rowSteps];

resetLayer = true;

}

}

//шаг принят

if(!resetLayer){

//Отобразим слой - при необходимости интерполируем

\*dataArray << lagranzDisplayInterpolation(currentLayer,tau);

//новые шаги

double \*alpha = new double;

double \*beta = new double;

calculateAlpa\_Beta(alpha,beta,lastLayer,lastLayer,tau);

if(h\*(\*alpha) <h\_max && t\*(\*beta) < t\_max){

//новая опорная функция

currentLayer =InterpolateLayer(currentLayer,h\*(\*alpha));

h\*=(\*alpha);

t\*=(\*beta);

r=(1/h)+1;

rowSteps = r;

//TODO:освободить память

A = new double\* [rowSteps];

for (int count = 0; count < rowSteps; count++) A[count] = new double[rowSteps];

B = new double [rowSteps];

X= new double[rowSteps];

}

\*lastLayer=\*currentLayer;

}

}

}

void MathHandler::**Exact**(){

double r=(1/h)+1;

int rowSteps = r;

double x =0;

dataArray = new QtDataVisualization::QSurfaceDataArray;

// слои

for(double i =0; i<=1+t; i+=t){

QtDataVisualization:: QSurfaceDataRow \*newRow = new QtDataVisualization::QSurfaceDataRow(rowSteps );

x=0;

int index =0;

while(x<=1){

/\*-----------------Точная формула----------------\*/

(\*newRow)[index++].setPosition(QVector3D(x,i,pow(a\*i+const\_A,0.75)\*pow((x+const\_B)\*(x+const\_B+1),-3/(double)2)));

x+=h;

}

\*dataArray << newRow;

}

}

double \* MathHandler::**systemSolver**(double \*\*a, double \*y, int n)

{

int i,k,n1;

double z;

double eps[n];

double \*x = new double [n];

double et[n];

n1=n-1;

eps[0]=-a[0][1]/a[0][0];

et[0]=y[0]/a[0][0];

for(i=1;i<n1;i++)

{

z=a[i][i]+a[i][i-1]\*eps[i-1];

eps[i]=-a[i][i+1]/z;

et[i]=(y[i]-a[i][i-1]\*et[i-1])/z;

}

x[n1]=(y[n1]-a[n1][n1-1]\*et[n1-1])/(a[n1][n1]+a[n1][n1-1]\*eps[n1-1]);

for(i=n1-1;i>=0;i--)

x[i]=eps[i]\*x[i+1]+et[i];

return x;

}

void MathHandler::**fillMatrix**(double \*\*A, double \*B, int sz, QList <double>\*lastLayer, QList<double> \*currentLayer, double tau){

//Обнуляем матрицу

for (int i=0;i<sz;i++) for (int k=0;k<sz;k++){A[i][k] =0; }

//границы

A[0][0] =1;

A[sz-1][sz-1] =1;

B[0] = pow(a\*tau+const\_A,0.75)\*pow((0+const\_B)\*(0+const\_B+1),-3/(double)2)-currentLayer->value(0) ;

B[sz-1] = pow(a\*tau+const\_A,0.75)\*pow((1+const\_B)\*(1+const\_B+1),-3/(double)2)-currentLayer->value(sz-1);

//Заполняем матрицу

for ( int i=1;i<sz-1;i++){

for (int k=0;k<sz-1;k++){

if(i==k){

//a2

A[i][k] = -1 +a\*(7/(double)9 \* pow(currentLayer->value(i),-10/(double)3)\*

pow((currentLayer->value(i+1)-currentLayer->value(i-1))/h,2) +8/(double)3\*pow(currentLayer->value(i),-7/(double)3)/(h\*h));

//a3

if((k+1) <sz) A[i][k+1] =a\*(2/(double)3 \*pow(currentLayer->value(i),-7/(double)3)/(h\*h)

\*(currentLayer->value(i+1)-currentLayer->value(i-1)) +pow(currentLayer->value(i),-4/(double)3)/(h\*h)); ;

//a1

if((k-1) >-1) A[i][k-1] = a\*(-2/(double)3\*((currentLayer->value(i+1)- currentLayer->value(i-1)))\*

(pow(currentLayer->value(i),-7/(double)3)/(h\*h))+pow(currentLayer->value(i),-4/(double)3)/(h\*h));

}

}

}

for (int i=1;i<sz-1;i++){

B[i] = lastLayer->value(i) -currentLayer->value(i) +(a\*t)/2\*(

(-2/3 \* pow(lastLayer->value(i),-7/3) \*

pow((lastLayer->value(i+1)-lastLayer->value(i-1))/h,2) +

pow(lastLayer->value(i),-4/3) \*

((lastLayer->value(i+1) - 2\*lastLayer->value(i)+lastLayer->value(i-1))/(h\*h)) )+

( -2/3 \* pow(currentLayer->value(i),-7/3) \*

pow((currentLayer->value(i+1)-currentLayer->value(i-1))/h,2) +

pow(currentLayer->value(i),-4/3) \*

((currentLayer->value(i+1) - 2\*currentLayer->value(i)+currentLayer->value(i-1))/(h\*h)) )

);

;

}

}

QtDataVisualization:: QSurfaceDataRow \* MathHandler::**lagranzDisplayInterpolation**(QList<double> \*Layer,double tau){

//перед этим запомним слой в 4д вектор

double x=0;int i =0;

while(x<=1){

X\_Y\_Z\_W\_ExW->append(QVector4D(x,tau,Layer->value(i),

pow(a\*tau+const\_A,0.75)\*pow((x+const\_B)\*(x+const\_B+1),-3/(double)2)));

x+=h;

i++;

}

double r=(1/h\_min)+1;

int rowSteps = r;

int index=0;

int indexExact=0;

double x\_interpolated =0;

x=0;

double z=0;

double Y[Layer->size()];

double X[Layer->size()];

for(int i=0; i<Layer->size();i++){

Y[i] = Layer->value(i);

X[i] =x;

x+=h;

}

QtDataVisualization:: QSurfaceDataRow \*newRow = new QtDataVisualization::QSurfaceDataRow(rowSteps);

QtDataVisualization:: QSurfaceDataRow \*ExactnewRow = new QtDataVisualization::QSurfaceDataRow(rowSteps);

for(double c=0; c<rowSteps;c++){

//Если точки такой нет - интерполируем

double p1,p2;

z=0;

for (int j=0; j<Layer->size(); j++){

p1=1; p2=1;

for (int i=0; i<Layer->size(); i++){

if (i==j){

p1=p1\*1;p2=p2\*1;

}

else {

p1=p1\*( x\_interpolated-X[i]);

p2=p2\*(X[j]-X[i]);

}

}

z=z+Y[j]\*p1/p2;

}

(\*newRow)[index++].setPosition(QVector3D(x\_interpolated,tau,z));

(\*ExactnewRow)[indexExact++].setPosition(QVector3D(x\_interpolated,tau,pow(a\*tau+const\_A,0.75)\*

pow((x\_interpolated+const\_B)\*(x\_interpolated+const\_B+1),-3/(double)2)));

x\_interpolated+=h\_min;

}

\*ExactdataArray<<ExactnewRow;

return newRow;

}

QList<double> \* MathHandler::**OnThisLayer\_HalfTauHalfH**(QList <double>\*lastLayer, double tau){

double r=(1/(h/2))+1;

int rowSteps = r;

QList <double>\*lastLayerInterpolated;

QList <double>\* result = new QList <double>();

lastLayerInterpolated = InterpolateLayer(lastLayer,h/2);

\*result=\*lastLayerInterpolated;

double save\_h =h;

double save\_t =t;

h/=2;

t/=2;

double \*\*A = new double\* [rowSteps];

for (int count = 0; count < rowSteps; count++) A[count] = new double[rowSteps];

double \*B = new double [rowSteps];

double \*W= new double[rowSteps];

//-------------------Метод Кранка-Николсона-------------------------

for (int count = 0; count <100; count++){

double max\_delta\_x =0;

//Заполнили матрицы

fillMatrix(A,B,rowSteps,lastLayerInterpolated,result,tau - t);

//решили систему

W=systemSolver(A,B,rowSteps);

//Нужны ли еще итерации?

for (int c = 0; c < rowSteps; c++){

result->replace(c,result->value(c)+W[c]);

if(max\_delta\_x<W[c]) max\_delta\_x = W[c];

}

if (max\_delta\_x< e) count=101;

}

\*lastLayerInterpolated = \*result;

//-------------------Метод Кранка-Николсона-------------------------

for (int count = 0; count <100; count++){

double max\_delta\_x =0;

//Заполнили матрицы

fillMatrix(A,B,rowSteps,lastLayerInterpolated,result,tau);

//решили систему

W=systemSolver(A,B,rowSteps);

//Нужны ли еще итерации?

for (int c = 0; c < rowSteps; c++){

result->replace(c,result->value(c)+W[c]);

if(max\_delta\_x<W[c]) max\_delta\_x = W[c];

}

if (max\_delta\_x< e) count=101;

}

delete lastLayerInterpolated ;

delete B;

delete W;

t=save\_t;

h=save\_h;

return result;

}

void MathHandler::**calculateAlpa\_Beta**(double\*alpha, double\*beta, QList <double>\*lastLayer,QList<double> \*currentLayer, double tau){

double r=(1/(h/2))+1;

int rowSteps = r;

QList <double>\*lastLayerInterpolated= InterpolateLayer(lastLayer,h/2);

QList <double>\* result = new QList <double>();

\*result=\*lastLayerInterpolated;

double save\_h =h;

h/=2;

double \*\*A = new double\* [rowSteps];

for (int count = 0; count < rowSteps; count++) A[count] = new double[rowSteps];

double \*B = new double [rowSteps];

double \*W= new double[rowSteps];

//-------------------Метод Кранка-Николсона-------------------------

for (int count = 0; count <100; count++){

double max\_delta\_x =0;

//Заполнили матрицы

fillMatrix(A,B,rowSteps,lastLayerInterpolated,result,tau - t);

//решили систему

W=systemSolver(A,B,rowSteps);

//Нужны ли еще итерации?

for (int c = 0; c < rowSteps; c++){

result->replace(c,result->value(c)+W[c]);

if(max\_delta\_x<W[c]) max\_delta\_x = W[c];

}

if (max\_delta\_x< e) count=101;

}

delete lastLayerInterpolated ;

delete B;

delete W;

h=save\_h;

QList <double>\*HalfTauHalfHLayer= OnThisLayer\_HalfTauHalfH(lastLayer,tau);

double curr\_alpha ;

double curr\_beta;

double last\_alpha= alpha\_max;

double last\_beta=beta\_max;

for(int i=0,q=0; i<currentLayer->size() && q<HalfTauHalfHLayer->size();i++,q+=2){

curr\_alpha =pow((3\*e)/(8\*(result->value(q) - currentLayer->value(i))),1/(double)3);

if(curr\_alpha > alpha\_min && curr\_alpha < last\_alpha) {last\_alpha = curr\_alpha;}

curr\_beta =sqrt( (3\*( e - ( 4/(3\*t\*t\*t) )\*( HalfTauHalfHLayer->value(q) - result->value(q) ) ) )

/(4\*curr\_alpha \*(result->value(q) - currentLayer->value(i)) ) );

if(curr\_beta > beta\_min && curr\_beta < last\_beta) last\_beta = curr\_beta;

}

\*(alpha) = last\_alpha;

\*(beta)=last\_beta;

}

QList<double> \*MathHandler::**InterpolateLayer**(QList <double>\*Layer, double new\_h){

double r=(1/(new\_h))+1;

int rowSteps = r;

double x\_interpolated =0;

double x=0;

double z=0;

QList <double>\*lastLayerInterpolated= new QList <double>();

double Y[Layer->size()];

double X[Layer->size()];

for(int i=0; i<Layer->size();i++){

Y[i] = Layer->value(i);

X[i] =x;

x+=h;

}

//опорная функция с шагом h/2

for(double c=0; c<rowSteps;c++){

//Если точки такой нет - интерполируем

double p1,p2;

z=0;

for (int j=0; j<Layer->size(); j++){

p1=1; p2=1;

for (int i=0; i<Layer->size(); i++){

if (i==j){

p1=p1\*1;p2=p2\*1;

}

else {

p1=p1\*(x\_interpolated-X[i]);

p2=p2\*(X[j]-X[i]);

}

}

z=z+Y[j]\*p1/p2;

}

lastLayerInterpolated->append(z);

x\_interpolated+=new\_h;

}

return lastLayerInterpolated;

}

**MathHandler::Approximate()для явної схеми**

void MathHandler::**Approximate**(){

double x=0;

double rowSteps = (1/h) +1;

QList <double>\*lastLayer= new QList <double>();

QList <double>\*currentLayer= new QList <double>();

dataArray = new QtDataVisualization::QSurfaceDataArray;

//Нулевой слой

QtDataVisualization:: QSurfaceDataRow \*newRow = new QtDataVisualization::QSurfaceDataRow(rowSteps);

int index = 0;

while(x<=1){

//Начальные условия

lastLayer->append(pow(const\_A,0.75)\*pow((x+const\_B)\*(x+const\_B+1),-3/(double)2));

(\*newRow)[index++].setPosition(QVector3D(x,0,pow(const\_A,0.75)\*pow((x+const\_B)\*(x+const\_B+1),-3/(double)2)));

currentLayer->append(0);

x+=h;

}

\*dataArray << newRow;

//Остальные слои

for(double i =t; i<=1; i+=t){

QtDataVisualization:: QSurfaceDataRow \*newRow = new QtDataVisualization::QSurfaceDataRow(rowSteps);

//граничные условия

currentLayer->replace(0,pow(a\*i+const\_A,0.75)\*pow((0+const\_B)\*(0+const\_B+1),-3/(double)2));

currentLayer->replace(currentLayer->size()-1,pow(a\*i+const\_A,0.75)\*pow((1+const\_B)\*(1+const\_B+1),-3/(double)2));

(\*newRow)[0].setPosition(QVector3D(0,i,currentLayer->value(0)));

(\*newRow)[currentLayer->size()-1].setPosition(QVector3D(1,i,currentLayer->value(currentLayer->size()-1)));

x=h;

int j = 1;index = 1;

while(x<1){

/\*-----------------Примерная формула----------------\*/

currentLayer->replace(j,lastLayer->value(j) + a\*t\*( -4/(double)3 \* pow(lastLayer->value(j), -7/(double)3) \*

pow((lastLayer->value(j+1) - lastLayer->value(j-1))/(2\*h),2) +

pow(lastLayer->value(j), -4/(double)3) \*

(lastLayer->value(j-1) -2\*lastLayer->value(j)

+ lastLayer->value(j+1))/(h\*h)

)

);

(\*newRow)[index++].setPosition(QVector3D(x,i, currentLayer->value(j)));

j++;

x+=h;

}

\*dataArray << newRow;

\*lastLayer=\*currentLayer;

}