НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра прикладної математики

«На правах рукопису»

УДК 519.622.2:004.42:004.021

«До захисту допущено»
Завідувач кафедри
_____ Олег ЧЕРТОВ
«___» _____ 2023 р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-науковою програмою «Наука про дані та математичне моделювання»
зі спеціальності 113 «Прикладна математика»
на тему: «Математичне та програмне забезпечення системи наближеного розв'язання матричних диференціальних рівнянь в аналітичному вигляді»

Виконав: студент ІІ курсу, групи КМ-11мн
Герасименко Владислав Русланович _____

Керівник:
Проф., д-р фіз.-мат. наук, проф.
Лось Валерій Миколайович _____

Консультант з нормоконтролю:
старший викладач,
Мальчиков Володимир Вікторович _____

Рецензент:
Проф., канд. фіз.-мат. наук, доц.,
Ільєнко Марина Костянтинівна

_____

Засвідчую, що в цій магістерській дисертації немає запозичень із праць інших авторів без відповідних посилань.

Студент _____

Київ — 2023

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра прикладної математики

Рівень вищої освіти — другий (магістерський)

Спеціальність – 113 «Прикладна математика»

Освітньо-наукова програма «Наука про дані та математичне моделювання»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олег ЧЕРТОВ
«___» _____ 2023 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Герасименку Владиславу Руслановичу

1. Тема дисертації: «Математичне та програмне забезпечення системи наближеного розв’язання матричних диференціальних рівнянь в аналітичному вигляді», науковий керівник дисертації Лось Валерій Миколайович, проф., д-р фіз.-мат. наук, затверджені наказом по університету від «30» травня 2023 р. № 1359-С.
2. Термін подання студентом дисертації: «15» травня 2023 р.
3. Об’єкт дослідження: Наближені методи знаходження розв’язку систем звичайних диференціальних рівнянь першого порядку
4. Предмет дослідження: Алгоритми методів наближеного знаходження розв’язку систем диференціальних рівнянь із заданими початковими умовами у аналітичному вигляді.
5. Перелік завдань, які потрібно розробити: систематизувати існуючі методи розв’язання задачі Коші для систем звичайних диференціальних рівнянь першого порядку, провести аналіз роботи алгоритму метода Пікара для систем диференціальних рівнянь, модифікувати метод для його застосування на ЕОМ,

довести збіжність такої модифікації, оцінити похибку, спроектувати систему розв'язання матричних диференціальних рівнянь за допомогою даної модифікації на ЕОМ, здійснити програмну реалізацію розробленої системи, провести тестування системи, провести експериментальні дослідження.

6. Орієнтовний перелік ілюстративного матеріалу: блок-схеми розроблених алгоритмів, загальна схема роботи системи, схема знаходження наближення розв'язку за методом Пікара, таблиці тестових випадків, знімки екранних форм

7. Орієнтовний перелік публікацій: стаття «Модифікований метод ітерацій для систем диференціальних рівнянь».

8. Дата видачі завдання: «1» лютого 2023 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Ґрунтовне ознайомлення з предметною областю	15.12.2021	
2	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури	01.03.2022	
3	Робота над першим розділом магістерської дисертації	15.05.2022	
4	Проведення наукового дослідження; робота над другим розділом магістерської дисертації	15.10.2022	
5	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	15.11.2022	
6	Робота над третім розділом магістерської дисертації; розроблення програмного забезпечення	01.03.2023	
7	Завершення роботи над основною частиною магістерської дисертації;	15.04.2023	

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
8	Оформлення текстової і графічної частин магістерської дисертації	25.04.2023	

Студент _____ Владислав ГЕРАСИМЕНКО

Науковий керівник дисертації _____ Валерій ЛОСЬ

РЕФЕРАТ

Дисертацію виконано на 102 аркушах, вона містить 2 додатки та перелік посилань на використані джерела з 9 найменувань. У роботі наведено 33 рисунки та 10 таблиць.

Актуальність теми. Добре відомо, що диференціальні рівняння, а зокрема системи таких рівнянь, представляють собою надважливий клас задач, які виникають у різних розділах науки та у прикладних задачах багатьох сфер промисловості, у тому числі і військової. Існує велика кількість чисельних та точних методів для розв'язання задач, пов'язаних з диференціальними рівняннями, в тому числі і для знаходження розв'язку задачі Коші для системи з m диференціальних рівнянь. Одним з таких методів є ітеративна процедура Пікара, яка є наближеним методом, тобто таким методом, що надає розв'язок у аналітичному вигляді. На жаль, застосування цього, та інших наближених методів, як правило, обмежується ілюстративними прикладами в рамках курсів чисельних методів, адже його реалізація на практиці, і тим паче на ЕОМ, є значно ускладненою необхідністю багаторазового інтегрування потенційно неполіноміальних функцій, з цієї причини наближені методи, і зокрема метод Пікара, майже ніколи не імплементуються програмно. Саме тому розробка нових та удосконалення існуючих наближених методів для їх застосування на ЕОМ є дуже актуальною проблемою на сьогодні.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційна робота виконувалась згідно з планом науково-дослідних робіт кафедри прикладної математики Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

Мета і задачі дослідження. Метою дисертаційної роботи є розробка математичного апарату та програмного забезпечення для наближеного розв'язання матричних диференціальних рівнянь у аналітичному вигляді.

Для досягнення вказаної мети було розв’язано такі задачі:

– систематизувати існуючі методи розв’язання задачі Коші для систем звичайних диференціальних рівнянь першого порядку;

– модифікувати метод Пікара послідовних наближень для його застосування на ЕОМ, довести збіжність такої модифікації, оцінити похибку;

– спроектувати систему розв’язання матричних диференціальних рівнянь за допомогою даної модифікації на ЕОМ;

– здійснити програмну реалізацію розробленої системи, провести її тестування;

– провести експериментальні дослідження.

Об’єктом дослідження є наближені методи знаходження розв’язку систем звичайних диференціальних рівнянь.

Предметом дослідження є алгоритми методів наближеного знаходження розв’язку систем диференціальних рівнянь із заданими початковими умовами у аналітичному вигляді.

Методи дослідження. Для розв’язання поставленої задачі використовувалися такі методи: методи дійсного та функціонального аналізу (для розробки модифікації методу Пікара); методи оптимізації (для чисельного знаходження оцінки похибки); методи теорії алгоритмів та програмування (для програмної реалізації розроблених алгоритмів); методи теорії диференціальних рівнянь (для розробки математичного апарату та для проведення експериментів).

Наукова новизна одержаних результатів складається з таких положень:

– уперше представлено модифікацію методу Пікара та доведено її збіжність, яка, на відміну від існуючих, передбачає заміну підінтегральних функцій першими членами їх рядів Тейлора, що дає змогу спростити обчислення і реалізувати програмно цей метод на ЕОМ;

– удосконалено програмну реалізацію методу Пікара, яка, на відміну від існуючих, опирається на інтегрування виключно поліномів з натуральними степенями, що дає змогу реалізувати цей метод без необхідності імплементації

алгоритмів символьного інтегрування чи диференціювання, адже інтегрування поліномів є, відносно, простою операцією;

– удосконалено процедуру оцінки похибки для методу Пікара, яка відрізняються від існуючих тим, що вона опирається на чисельні методи оптимізації, що дає можливість застосовувати емпіричні методи оптимізації, які є алгоритмічно простими.

Практичне значення одержаних результатів. На основі запропонованої модифікації методу Пікара, побудовано веб-застосунок для наближеного розв'язання систем звичайних диференціальних рівнянь першого порядку.

Апробація результатів дисертації. Основні положення й результати роботи представлено та опубліковано на конференції ПМК 2022.

Публікації.

- Лось В.М., Герасименко В.Р., Копичко С.М. (2022). Модифікований метод ітерацій для систем диференціальних рівнянь. ПМК-2022.

Ключові слова: задача Коші, диференціальні рівняння, теорема Банаха, принцип стискаючих відображень, символьне інтегрування, система диференціальних рівнянь, метод Пікара, метод Рунге-Кутта, ряд Тейлора

ABSTRACT

The thesis is presented in 102 pages. It contains 2 appendixes and bibliography of 9 references. 33 figures and 10 tables are given in the thesis.

Topic relevance. It is well known that differential equations, and in particular systems of such equations, represent an extremely important class of problems that arise in various branches of science and in applied problems of many industries, including the military. There are a large number of numerical and accurate methods for solving problems involving differential equations, including finding a solution to the Cauchy problem for a system of m differential equations. One of these methods is the Picard iterative procedure, which is an approximate method, i.e. a method that provides a solution in analytical form. Unfortunately, the use of this and other approximate methods is usually limited to illustrative examples in numerical methods courses, because its implementation in practice, and even more so on a computer, is significantly complicated by the need to repeatedly integrate potentially non-polynomial functions, for this reason approximate methods, and in particular the Picard method, are almost never implemented in software. That is why the development of new and improvement of existing approximate methods for their application on computers is a very urgent problem today.

Thesis connection to scientific programs, plans, and topics. The thesis was prepared according to the scientific research plan of the Applied Mathematics Department of the National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute.”

Research goal and objectives. The goal of this thesis is to develop a mathematical apparatus and software for the approximate solution of matrix differential equations in analytical form.

To accomplish this goal, the following objectives were reached:

– systematise existing methods for solving the Cauchy problem for systems of ordinary differential equations of the first order;

– modify the Picard’s method of successive approximations for its application on a computer, prove the convergence of such a modification, and estimate the error;

– design a system for solving matrix differential equations using this modification on a computer;

– to implement the software implementation of the developed system, to test the system.

– to conduct experimental research with systems of differential equations.

Object of research is approximate methods for finding solutions to systems of ordinary differential equations.

Subject of research is the algorithms of methods for approximate solution of systems of differential equations with given initial conditions in analytical form.

Methods of research. To solve the task, the following methods were used: methods of real and functional analysis (to develop a modification of Picard's method); optimisation methods (to numerically find the error estimate); methods of algorithm theory and programming (for software implementation of the developed algorithms); methods of differential equations theory (to develop mathematical software and to conduct experiments).

Scientific contribution consists of the following:

– for the first time presented a modification of Picard's method and proved its convergence, which, unlike the existing ones, involves the replacement of subintegral functions, which allows to implement this method on a computer;

– improved software implementation of the Picard method, which, unlike the existing ones, is based on the integration of polynomials with natural degrees only, which makes it possible to implement this method without the need to implement symbolic integration or differentiation algorithms, since the integration of polynomials is a relatively simple operation;

– improved the error estimation procedure for the Picard method, which differs from the existing ones in that it is based on numerical optimisation methods, which makes it possible to apply empirical optimisation methods that are algorithmically very simple.

Practical value of obtained results. On the basis of the proposed modification of Picard's method, a web application for the approximate solution of systems of ordinary differential equations of the first order is built.

Approbation of the thesis results. Basic ideas and results of the research were presented at the 2022 PMK conference.

Publications.

- Los Valeriy, Herasimenko Vladyslav, Kopychko Sergei (2022). A modified iteration method for systems of differential equations. PMK-2022.

Keywords: Cauchy problem, differential equations, Banach theorem, principle of compressive mappings, symbolic integration, system of differential equations, Picard method, Runge-Kutta method, Taylor series

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ЕОМ – електронно-обчислювальна машина

ЗДР – звичайне диференціальне рівняння

∇f^i – кінцева різниця i-го порядку функції f

$\equiv$ – дорівнює за означенням

ВСТУП

В багатьох наукових дисциплінах виникають задачі, пов'язані з диференціальними рівняннями, а зокрема із системами диференціальних рівнянь. Вони є надважливим класом задач, тому не дивно, що на даний момент існує дуже багато точних та чисельних методів для їх розв'язання, в тому числі і для розв’язання задачі Коші для систем з m звичайних диференціальних рівнянь першого порядку.

Одним з таких методів є ітеративна процедура Пікара послідовних наближень, проте її застосування на практиці, на жаль, часто обмежується ілюстративними прикладами в межах курсів чисельних методів, що пов'язано, серед іншого, із складністю імплементації цього методу на ЕОМ, через потребу багаторазового інтегрування потенційно неполіноміальних функцій, що може призвести до надзвичайно складних квадратур, які можуть навіть і не виражатися в елементарних функціях.

Вирішенню цієї проблеми, власне, і присвячена ця робота. Особлива увага приділяється розробленню математичного та програмного забезпечення для ефективного застосування метода Пікара послідовних наближень на ЕОМ для систем диференціальних рівнянь. Додатковою вимогою є знаходження розв’язків у аналітичному вигляді, на відміну від табличних методів, таких як метод Рунге-Кутта, які потребують додаткового знаходження інтерполяційного многочлена між вузлами сітки.

Кінцевий продукт може бути використаний у різних інженерних середовищах для наближеного розв'язування диференціальних рівнянь, а також в якості демонстраційного інструменту для поліпшення засвоєння методу Пікара студентами-математиками.

1 ОГЛЯД ІСНУЮЧИХ МЕТОДІВ ТА ПРОГРАМНИХ ЗАСОБІВ

1.1 Математичні методи знаходження розв'язку задачі Коші для системи з m звичайних диференціальних рівнянь

Почнемо з того, що нагадаємо формулювання задачі Коші для системи з m звичайних диференціальних рівнянь першого порядку, які розглядаються в рамках цієї роботи. Скористаємось визначенням, наданим у праці [2]:

«Для заданої точки $(\tau_0, y_1(\tau_0), \dots, y_m(\tau_0)) \in R^{m+1}$, необхідно знайти такі функції $y_1(\tau), \dots, y_m(\tau)$, що визначені у певному околі точки x_0, і які задовольняють систему рівнянь (1.1)-(1.2)».

$$\begin{cases} \boldsymbol{y_1'} = \boldsymbol{f_1}(\boldsymbol{\tau}, \boldsymbol{y_1}(\boldsymbol{\tau}) \dots \boldsymbol{y_m}(\boldsymbol{\tau})) \\ \dots \\ \boldsymbol{y_m'} = \boldsymbol{f_m}(\boldsymbol{\tau}, \boldsymbol{y_1}(\boldsymbol{\tau}) \dots \boldsymbol{y_m}(\boldsymbol{\tau})) \end{cases} \quad (1.1)$$

За умов:

$$\boldsymbol{y_1}(\boldsymbol{\tau_0}) = \boldsymbol{y_{0,1}} \dots \boldsymbol{y_m}(\boldsymbol{\tau_0}) = \boldsymbol{y_{0,m}} \quad (1.2)$$

Або, у векторному варіанті: для заданої точки $(\tau_0, \boldsymbol{y}(\tau_0)) \in R^{m+1}$, знайти вектор-функцію $\boldsymbol{y}(\boldsymbol{x})$, що є визначеною у певному околі точки x_0, і яка задовольняє (1.3)-(1.4):

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{\tau}, \boldsymbol{y}) \quad (1.3)$$

За умови:

$$\boldsymbol{y}(\tau_0) = \boldsymbol{y}_0 \quad (1.4)$$

Надалі ми часто користуватимемось векторним записом у силу його компактності.

Існує багато методів для вирішення таких задач Коші, які можуть бути розділені на такі категорії:

- точні (чи «аналітичні») методи: вони дають результат у аналітичному вигляді.
- Чисельні (чи «табличні») методи, які дають результат у вигляді таблиці значень наближеного рішення задачі Коші
- та методи, які на виході дають результат у вигляді послідовності елементарних функцій. Ці методи також зазвичай називають «чисельними», але щоб уникнути двозначності, ми називатимемо їх «наближеними»

Чисельні методи також часто поділяють на два підкласи: «однокрокові» та «багатокрокові», залежно від того, скільки попередніх значень використовують для знаходження чергового наближення. Однокрокові методи, як можна здогадатися з назви, використовують лише одне наближене значення розв’язку (знайдене на попередньому кроці) для знаходження наступного наближення, тоді як багатокрокові методи – використовують декілька попередніх наближень.

Точні методи диференціальних рівнянь вивчаються у курсах теорії диференціальних рівнянь.

Найбільш поширеними однокроковими чисельними методами є, перш за все, методи Ейлера та модифікований метод Ейлера, також варто відзначити методи Рунге-Кутта. До популярних багатокрокових методів можна віднести метод забігання вперед, метод Адамса, і метод Мілна.

Наближені методи не є дуже поширеними через складність їх застосування та реалізації на ЕОМ, але серед них можна виділити деякі популярні методи, такі як метод Чаплигіна, метод узагальненого степеневого ряду та метод Пікара послідовних наближень.

Кожен з цих методів має свої переваги та недоліки, які ми розглянемо у наступних підрозділах.

1.1.1 Однокрокові чисельні методи

Метод Ейлера був, історично, найпершим і найпримітивнішим методом розв'язання задачі Коші для звичайних диференціальних рівнянь. На практиці, він фактично не використовується, адже як метод Ейлера, так і його модифікація : уточнений метод Ейлера мають дуже низький порядок точність – відповідно, 1-й та 2-й. Тим не менш, ці методи часто виявляються на диво корисними для доведення різних теорем, у тому числі, наприклад, теорему Пікара існування і єдиності розв'язку задачі Коші можна довести опираючись на [1].

Ці два методи легко виводяться, як в одновимірному, так і в багатовимірному випадках (при чому, використовуючи векторне представлення, доведення є таким же, як в одновимірному випадку), прямо із задачі Коші (1.3)-(1.4), описаної в попередньому підрозділі [4, 5, 7].

Спочатку інтервал інтегрування розбивається на l рівних частин $[\tau_k, \tau_{k+1}]$, кожна з яких має довжину h. Після чого рівняння (1.3) інтегрується на інтервалі $[\tau_i, \tau_{i+1}]$ і отримаємо (1.5):

$$\boldsymbol{y}(\tau_{i+1}) - \boldsymbol{y}(\tau_i) = \int_{\tau_i}^{\tau_{i+1}} \boldsymbol{f}\big(x, \boldsymbol{y}(\tau)\big)\, d\tau \tag{2.5}$$

Одразу відмітимо, що під відніманням у лівій частині мається на увазі векторне (поелементне) віднімання $\boldsymbol{y}(\tau_i)$ від $\boldsymbol{y}(\tau_{i+1})$, а інтеграл у правій частині застосовується до кожного елемента вектор-функції $\boldsymbol{f}\big(x, \boldsymbol{y}(\tau)\big)$.

Далі, чисельно наблизимо інтеграл у правій частині за методом прямокутників і отримаємо (1.6):

$$\boldsymbol{y}(\tau_{i+1}) = h\boldsymbol{f}\big(\tau_i, \boldsymbol{y}(\tau_i)\big) + \boldsymbol{y}(\tau_i), i = \overline{0, l-1} \tag{2.6}$$

У якості початкового наближення $(\tau_0, \boldsymbol{y}(\tau_i))$ береться умова (2.4).

Модифікований (або також «уточнений») метод Ейлера отримується із тієї ж формули (1.5). Тільки цього разу наближення інтеграла у правій частині (1.7) шукається згідно методу трапецій:

$$\boldsymbol{y}(\tau_{i+1}) = \frac{h}{2}\Big(\boldsymbol{f}\big(\tau_i, \boldsymbol{y}(\tau_i)\big) + \boldsymbol{f}\big(\tau_{i+1}, \boldsymbol{y}(\tau_{i+1})\big)\Big) + \boldsymbol{y}(\tau_i), i = \overline{0, l-1} \tag{1.7}$$

Цікавою особливістю даного методу є той факт, що на кожному кроці необхідно обчислювати значення $\boldsymbol{f}\big(\tau_{i+1}, y(\tau_{i+1})\big)$. Зазвичай, це роблять за допомогою звичайного метода Ейлера (саме тому цей метод – називається «уточненим»)

Іншим класом алгоритмів є методи Рунге-Кутта, що мають набагато вищу точність, порівняно з методами Ейлера (зазвичай, 3-го чи 4-го порядку, але можна вивести методи і значно вищих порядків) і саме тому це є, ймовірно, найбільш популярні методи розв'язання диференціальних рівнянь і їх систем на ЕОМ. Тим не менш, ці методи не без недоліків і за високу точність доводиться платити набагато більшим часом обчислень, у порівнянні з методами Ейлера.

Іншим підкласом чисельних методів є так звані методи Рунге-Кутта, що знову ж таки, отримуються з тієї ж формули (1.5), цього разу, наближуючи інтеграл за формулою Сімпсона. Зважаючи на те, що для застосування цієї формули наближеного обчислення інтеграла, необхідно мати три точки, то сам інтеграл рахується трохи інакше, а саме: вводиться проміжна точки $\tau_{i+1/2}$, яка лежить посередині між точками τ_i та τ_{i+1}, що дозволяє переписати формулу (1.5) у вигляді (1.8):

$$\boldsymbol{y}(\tau_{i+1}) - \boldsymbol{y}(\tau_i) = \int\limits_{\tau_{i+1/2}-h/2}^{\tau_{i+1/2}+h/2} \boldsymbol{f}\big(\tau, \boldsymbol{y}(\tau)\big)\, d\tau \tag{1.8}$$

І нарешті, ми можемо застосувати метод Сімпсона із кроком інтегрування $h/2$ та отримаємо запис (1.9):

$$\boldsymbol{y}(\tau_{i+1}) = \frac{h}{6}\left(\boldsymbol{f}\left(\tau_i, \boldsymbol{y}(\tau_i)\right) + 4\boldsymbol{f}\left(\tau_{i+\frac{1}{2}}, \boldsymbol{y}\left(\tau_{i+\frac{1}{2}}\right)\right) + f\left(\tau_{i+1}, \boldsymbol{y}(\tau_{i+1})\right)\right) + \boldsymbol{y}(\tau_i) \quad (1.9)$$

Що дає нам так зване «неявне» представлення, яке так називається, бо $y\left(\tau_{i+\frac{1}{2}}\right)$ та $y(\tau_{i+1})$ у правій частині – невідомі. Формула (1.7) далі може бути використана для отримання різних методів Рунге-Кутта з різними порядками точності. Зокрема, в рамках цієї роботи, ми використовуємо метод Рунге-Кутта 4-го порядку точності для порівняння з методом Пікара. Детальніше, цей метод описано у розділі 2.

1.1.2 Багатокрокові методи

Як вже було відмічено, багатокрокові методи є іншим підкласом чисельних методів. Вони відрізняються від однокрокових методів, таких як Рунге-Кутта тим, що для знаходження наступного наближення, вони використовують знайдені значення розв’язку не тільки з останнього, а з декількох попередніх наближень.

Часто, це дозволяє отримати набагато більш точне наближення розв’язку за ту ж кількість ітерацій, адже на кожному кроці ці методи використовують більшу кількість інформації. Тим не менш, ці методи страждають від декількох недоліків. Одним з них є наприклад той факт, що декілька початкових наближень потрібно знаходити, користуючись іншими методами, як от, методом Рунге-Кутта.

Власне, в цьому розглянемо більш детально деякі з популярних багатокрокових методів.

І почнемо з методу Адамса, який є чи не найбільш поширеним багатокроковим методом, що опирається на формулу інтерполяції Ньютона назад [4]. Цей метод теж

легко отримується з рівняння (1.5). Припустимо, як і раніше, що увесь наш інтервал інтегрування розбито на *l* підінтервалів і нехай ми маємо наближення розв'язку на перших *k* точках розбиття цього інтервалу $(\tau_0, \tau_1, \dots, \tau_k)$, тоді запишемо знову рівняння (1.5).

$$\boldsymbol{y}(\tau_{k+1}) - \boldsymbol{y}(\tau_k) = \int_{\tau_k}^{\tau_{k+1}} \boldsymbol{f}\big(\tau, \boldsymbol{y}(\tau)\big)\, d\tau \qquad (1.10)$$

Після чого здійснимо заміну у формулі (1.1) підінтегральної функції $\boldsymbol{f}\big(\tau, \boldsymbol{y}(\tau)\big)$ на її інтерполяційний многочлен Ньютона інтерполяції назад. Звідки ми й отримуємо формулу Адамса. Вона є відносно громіздкою у своєму загальному вигляді, тому тут наведемо лише формулу, коли нам відомо, наприклад, останні 3 наближення (а не *k*):

$$\boldsymbol{y}(\tau_{k+1}) = \\ +h\left(\frac{5}{12}\boldsymbol{f}\big(\tau_{k+1}, \boldsymbol{y}(\tau_{k+1})\big) + \frac{2}{3}\boldsymbol{f}\big(\tau_k, \boldsymbol{y}(\tau_k)\big) - \frac{1}{12}\boldsymbol{f}\big(\tau_{k-1}, \boldsymbol{y}(\tau_{k-1})\big)\right) + \boldsymbol{y}(\tau_k) \qquad (1.11)$$

Подібним багатокроковим методом є також метод Мілна, який є досить поширеним та, у той же час, відносно простим у реалізації. Відмінність від методу Адамса тут полягає лише у тому, що замість інтерполяційного многочлена Ньютона для інтерполяції назад, тут використовується інтерполяційний многочлен Ньютона для інтерполяції вперед [4].

Щоб отримати основну формулу метода, проінтегруємо початкове рівняння (1.3) на інтервалі $[\tau_{k-3}, \tau_{k+1}]$ і отримаємо:

$$\boldsymbol{y}(\tau_{k+1}) = \int_{\tau_{k-3}}^{\tau_{k+1}} \boldsymbol{f}\big(\tau, \boldsymbol{y}(\tau)\big)\, d\tau + \boldsymbol{y}(\tau_{k-3}) \qquad (1.12)$$

Підінтегральну функцію $\boldsymbol{f}\big(\tau, \boldsymbol{y}(\tau)\big) = \boldsymbol{f}(\tau)$ замінимо тоді таким виразом:

$$\boldsymbol{f}(\tau) \cong \boldsymbol{a_0} + (\tau - \tau_m)\boldsymbol{a_1} + (\tau - \tau_m)(\tau - \tau_{m+1})\boldsymbol{a_2} + \cdots + \\ +(\tau - \tau_m) \ldots (\tau - \tau_{m+l-1})\boldsymbol{a_m} \quad (1.13)$$

Це і є інтерполяційний многочлен Ньютона, інтерполяції «вперед», на сітці $\tau_m, \tau_{m+1} \ldots \tau_{m+l-1}$. Нагадаємо також, що у формулі (1.13), коефіцієнти $\boldsymbol{a}_i$ обчислюються за допомогою кінцевих різниць *i*-го порядку для функції $\boldsymbol{f}$, що мають вигляд (1.14).

$$\boldsymbol{a_i} = \frac{\nabla^i \boldsymbol{f}\left(\tau_m, \boldsymbol{y}(\tau_m)\right)}{h^i i!} \quad (1.14)$$

Підставивши цей вираз у (2.12) та поклавши $m = k - 3$ і $l = 3$, після відносно нескладних спрощень і розрахунків, отримаємо «першу формулу Мілна» (1.15).

$$\boldsymbol{y}(\tau_{k+1}) = \\ +\frac{4h}{3}\left(2\boldsymbol{f}\left(\tau_k, \boldsymbol{y}(\tau_k)\right) - \boldsymbol{f}\left(\tau_{k-1}, \boldsymbol{y}(\tau_{k-1})\right) + 2\boldsymbol{f}\left(\tau_{k-2}, \boldsymbol{y}(\tau_{k-2})\right)\right) + \boldsymbol{y}(\tau_{k-3}) \quad (1.15)$$

Аналогічно, інтегруючи знову рівняння (1.3), тепер на інтервалі $[\tau_{k-1}, \tau_{k+1}]$, й замінюючи у записі (1.13) інтерполяційного многочлена $m = k - 1$ та $l = 3$, отримаємо тоді «другу формулу Мілна» (1.16).

$$\boldsymbol{y}(\tau_{k+1}) = \frac{h}{3}\left(\boldsymbol{f}\left(\tau_{k+1}, \boldsymbol{y}(\tau_{k+1})\right) + 4\boldsymbol{f}\left(\tau_k, \boldsymbol{y}(\tau_k)\right) + \boldsymbol{f}\left(\tau_{k-1}, \boldsymbol{y}(\tau_{k-1})\right)\right) + \boldsymbol{y}(\tau_{k-1}) \quad (1.16)$$

Одразу відмітимо тут, що першу й другу формулу Мілна використовують, як правило, за схемою предиктор-коректор, відповідно (див., наприклад, [4] для деталей).

Є й багато інших різних багатокрокових методів, серед яких можна також відмітити метод «з забіганням вперед», але в більшості випадків, використовується точно така ж ідея, як і у всіх попередньо-розглянутих методах : проінтегрувати

рівняння (1.3) на певному інтервалі та наблизити інтеграл у правій частині певною інтерполяційною схемою.

1.1.3 Наближені методи

Так звані «наближенні методи» фундаментально відрізняються від чисельних методів, що були розглянуті у попередніх підрозділах, у тому числі тим, що на виході вони дають, як правило, якусь комбінацію елементарних функцій (або їх послідовність, яка в решті решт збігається до точного розв'язку), що дає наближення розв'язку на інтервалі інтегрування. Перевагою таких методів є те, що ми можемо одразу отримати значення розв'язку у будь-якій обраній точці області збіжності, без необхідності здійснювати інтерполяцію табличних значень, як це потрібно робити при застосуванні чисельних методів.

Іншою безумовною перевагою є можливість відносно простого керування похибкою : у переважній більшості методів, достатньо лише прийняти до уваги оцінку похибки та обрати відповідну кількість ітерацій. Варто відзначити, що розглянуті до цього «чисельні» методи теж можуть бути уточнені за допомогою, наприклад, відомої процедури Рунге-Кутта. Її недоліком, тим не менш, залишається той факт, що проте вона потребує досить складних й громіздких обчислень.

На жаль, ці методи, як вже було відмічено, досить рідко використовуються на практиці через їх алгоритмічну складність та складність їх імплементації на ЕОМ; радше, вони представляють теоретичний інтерес. Тим не менш, в окремих випадках застосування цих методів є цілком практичним, як правило це означає, що праві частини рівнянь системи повинні мати певний спеціальний вигляд : наприклад, як ми побачимо далі у розділі 2, якщо усі праві частини представлені поліномами, то застосування методу Пікара послідовних наближень, який також є представником наближених методів, стає досить тривіальним, а обчислення – простими.

У цьому розділі, ми розглянемо деякі з найвідоміших «наближених» методів.

Почнемо з методу Чаплигіна, який базується на однойменній теоремі і хоч оригінально він застосовується для наближеного вирішення одновимірних ЗДР, його легко узагальнити на випадок системи диференціальних рівнянь [4, 5, 7]. Сама теорема заслуговує окремої роботи і є досить складної, щоб можна було її описати коротко, тому тут наведемо лише алгоритм роботи методу, який сам по собі є відносно простим (знову використовуватимемо векторне представлення системи для спрощення викладок):

1. Спочатку знаходяться такі дві вектор-функції $\boldsymbol{z_0}$ і $\boldsymbol{u_0}$, що обидві з них вони задовольняють рівняння (2.3) у певній точці x_0 за умови (1.4) і що для них на деякому напівінтервалі $(\tau, \mathrm{b}]$ справджується:

$$\boldsymbol{u_0'} - \boldsymbol{f}(x, \boldsymbol{u_0}) < 0 \tag{1.17}$$

$$\boldsymbol{z_0'} - \boldsymbol{f}(x, \boldsymbol{z_0}) > 0 \tag{1.18}$$

   По суті, ці функції являють собою певного роду “бар’єри”, всередині яких лежить функція справжнього розв’язку $y(\tau)$. Таким чином, ці функції, виступають у ролі чергового наближення розв’язку.
2. Надалі ітераційна процедура будується таким чином: якщо нам відомі якісь наближення розв’язку u_i та z_i, то наступне наближення визначається за формулами [6]:

$$\boldsymbol{h}(\tau) = \boldsymbol{u_i'}(\tau) - \boldsymbol{f}\big(\tau, \boldsymbol{u_i}(\tau)\big) \tag{1.19}$$

$$\boldsymbol{u_{i+1}}(\tau) = \boldsymbol{u}_i(\tau) - \int_{\tau_0}^{\tau} e^{-L(\tau - t)} \boldsymbol{h}(t) dt \tag{1.20}$$

$$\boldsymbol{g}(\tau) = \boldsymbol{z_i'}(\tau) - \boldsymbol{f}\big(\tau, \boldsymbol{z_i}(\tau)\big) \tag{1.21}$$

$$\boldsymbol{z_{i+1}}(\tau) = \boldsymbol{z_i}(\tau) - \int_{\tau_0}^{\tau} e^{-L(\tau - t)} \boldsymbol{g}(t) dt \tag{1.22}$$

Тут, L – константа Ліпшиця для функції $f(\tau, y(\tau))$, адже, згідно умови, вона є Ліпшицевою.

Тут важливо підкреслити той факт, що так як метод Чаплигіна є фактично модифікацією методу Ньютона вирішення ЗДР, то швидкість збіжності може бути оцінена таким чином [6]: $z_n - u_n \leq C/2^{2^n}$, для певного n, що відповідає кроку метода.

В решті решт, є також метод Пікара послідовних наближень, якому повністю і присвячена ця робота, цей наближений метод часто можна зустріти в літературі метод, хоча він, як уже неодноразово відмічалося, дуже рідко використовується на практиці. Він базується на ітеративній процедурі, яка прямо випливає із теореми Банаха про нерухому точку, яка також гарантує лінійну швидкість збіжності цього методу (іншими словами, збіжність «гарантована» за нескінченну кількість ітерацій). До поганої швидкості збіжності, серед недоліків, можна також віднести складність його реалізації на ЕОМ, адже на кожному кроці необхідно знаходити інтеграл правої частини диференціального рівняння, що може, у загальному випадку, приводити до надзвичайно складних квадратур, що можуть навіть і не виражатися у елементарних функціях (це ж саме, насправді кажучи, стосується і метода Чаплигіна й багатьох інших наближених методів, що покладаються на пряме інтегрування для знаходження чергового наближення) [6]. Більш детально цей метод розглядається у наступних розділах (зокрема, у розділі 2), тому не повторюватимемо ту ж саму інформацію тут.

Існує й багато інших наближених методів: можна навести у приклад метод узагальненого степеневого ряду, але він використовується зазвичай для спеціальних типів систем диференціальних рівнянь, тому тут не надаватимемо більше деталей, а залишимо це для третього розділу.

1.2 Існуючі програмні засоби для вирішення задачі Коші для систем звичайних диференціальних рівнянь першого порядку

На сьогоднішній день, існує величезна кількість програмних продуктів для розв’язання, фактично, будь-яких математичних задач, що виникають на практиці, у тому числі і задач Коші для систем звичайних диференціальних рівнянь.

Ці інженерні системи реалізовані у багатьох різних форматах, але ймовірно найпопулярнішими залишаються різноманітні веб-сервіси такі як «Wolfram Alpha», що позиціонує себе як «науковий онлайн-калькулятор» та здатен розв’язувати дуже широкий клас задач як аналітичними, так і чисельними методами. Зокрема, у ньому реалізовано метод Рунге-Кутта для систем диференціальних рівнянь, а також, у відносно обмеженому варіанті, є функціонал для застосування методу Пікара для одновимірної задачі Коші.

Серед інших онлайн-сервісів вирішення диференціальних рівнянь та їх систем, можна навести такі відомі веб-застосунки, як «dcode.fr», «symbolab.com», «emathhelp.net» та багато інших.

Велика кількість таких застосунків пояснюється, звичайно, простотою та прямолінійністю імплементації чисельних методів, як метод Рунге-Кутта, на ЕОМ, що дають відносно точні результати, не вимагаючи при цьому значних обчислень.

Варто також відмітити появу новітніх рішень для розв’язання складних математичних задач (власне, у тому числі і задач Коші) за допомогою методів штучного інтелекту. Наприклад, за останні декілька років, з’явилась певна кількість систем обробки природної мови, таких як ChatGPT, що здатні розв’язувати примітивні системи диференціальних рівнянь у текстовому форматі. Існують також спеціалізовані системи штучного інтелекту, призначені для розв’язання диференціальних рівнянь, що опираються, серед іншого, на архітектуру ODE-RNN, яка використовує для обчислень той факт, що, як виявляється, існує пряма відповідність між рекурсивними диференціальними рівняннями та звичайними диференціальними рівняннями.

У той же час, програмних рішень, що б використовували метод ітерацій Пікара, доступних широкому загалу є неймовірно мала кількість, що, в свою чергу, легко пояснюється значною складністю реалізації цього методу на ЕОМ, а також з кількістю обчислень необхідних для знаходження розв'язку із достатньою точністю. У рамках пошуку подібних систем, у вільному доступі було знайдено буквально лише одну «демонстрацій» на базі «Wolfram Alpha», про яку уже було загадано раніше.

1.3 Висновки до розділу

За результатами цього розділу, було описано в грубих лініях основні сучасні методи вирішення задачі Коші для систем звичайних диференціальних рівнянь першого порядку. Також було проведено порівняльний аналіз їх головних переваг та недоліків, зокрема у Пікара.

Результати цього аналізу зібрані в таблиці 1.1.

Таблиця 1.1 – порівняння методів розв'язання задачі Коші для систем звичайних диференціальних рівнянь першого порядку

Метод	Точність*	Складність імплементації	Тип розв'язку
Точні методи	Гарантовано точний розв'язок	Висока	Аналітичний
Метод Ейлера	$O(h^2)$	Дуже низька	Сітковий (Табличний)
Модифікований метод Ейлера	$O(h^2)$	Дуже низька	Сітковий (Табличний)
Метод Рунге-Кутта	$O(h^n)$, де $n=4$ як правило	Дуже низька	Сітковий (Табличний)

Продовження таблиці 1.1

Метод	Точність*	Складність імплементації	Тип розв’язку
Метод Адамса	$O(h^n)$, де n залежить від кількості попередніх наближень	Низька	Сітковий (Табличний)
Метод Мілна	$O(h^n)$, де n залежить від кількості попередніх наближень	Низька	Сітковий (Табличний)
Метод Чаплигіна	Як у методі Ньютона	Висока	Аналітичний
Метод Пікара	$O(a^n)$, де a – коефіцієнт стиску	Висока	Аналітичний

Ремарка *, аналіз точності у даній таблиці, виконувався у припущенні того, що розв’язок існує, є стійким і розглядається у області збіжності.

Зрештою, основна увага у даній роботі, звісно, приділяється методу Пікара послідовних наближень. Основною перевагою цього методу безсумнівно є аналітичний вигляд розв’язку, тобто, як було відмічено, немає, наприклад, необхідності знаходити інтерполяційну функцію між вузлами сітки, щоб знайти значення розв’язку між значенням таблиці, як це робиться у табличних методах. Також, іншою перевагою, є простота уточнення розв’язку: достатньо просто збільшити кількість ітерацій методу. До недоліків можна віднести повільну збіжність: теорема Банаха про стискуючий оператор гарантує лише лінійну збіжність (до речі, у цьому, ми матимемо змогу переконатись у розділі 3). Іншим недоліком є також

складність операції інтегрування, на якій будується ітеративна процедура, що значно ускладнює імплементацію цього методу на ЕОМ.

2 ОПИС МАТЕМАТИЧНИХ МЕТОДІВ

В цьому розділі, ми, спочатку, наведемо деякі з необхідних нам для подальших викладок понять та визначень, у тому числі для формулювання принципу Пікара-Банаха про нерухому точку, на якому фактично повністю і базується метод Пікара, що розглядається у рамках цієї роботи. У цих викладках ми також використовуватимо уже отримані результати, зокрема ті, що стосуються доведення збіжності методу Пікара [1, 2, 8]. Після цього, ми пояснимо у чому, зі строго математичної точки зору, полягає модифікація методу Пікара, що пропонується нами для застосування на ЕОМ, а також доведемо її збіжність. Для неї ж буде отримано «глобальну» оцінку похибки та буде показано, як побудувати розв'язок на інтервалі, більшому за область збіжності. В решті решт, ми також пропонуємо прийом для економії обчислювальних ресурсів та оперативної пам'яті ЕОМ, при застосуванні згаданої модифікації.

2.1 Основні поняття з теорії диференціальних рівнянь

В цьому підрозділі, ми зазначимо лише найбільш важливі визначення та поняття з теорії диференціальних рівнянь, що знадобляться нам надалі. Ці та інші визначення запозичені із відомого підручника [2].

І почнемо ми із визначення того виду систем диференціальних рівнянь, які будуть розглядатися у цій роботі.

Нехай D – певна область в просторі R^{m+1} і $\left(x, y_1(x), \dots, y_m(x)\right)$ – є координати у даному просторі. Тоді, кожна вектор-функція $\boldsymbol{f}: D \to R^m$ визначає у цій області систему диференціальних рівнянь (2.1).

$$\boldsymbol{y}' = \boldsymbol{f}(x, \boldsymbol{y}) \tag{2.1}$$

Відзначимо, що тут ми користуємося «векторним» позначенням, введеним у розділі 1, тобто : виділені жирним змінні та функції є, насправді, відповідно векторами та вектор-функціями. Дійсно, у випадку рівняння (2.1), воно може бути переписано у вигляді (2.2).

$$\begin{cases} y_1' = f_1\big(x, y_1(x) \dots y_m(x)\big) \\ \dots \\ y_m' = f_m\big(x, y_1(x) \dots y_m(x)\big) \end{cases} \tag{2.2}$$

Проте так як такий запис є досить громіздким, ми надаватимемо перевагу запису виду (2.1).

Перейдемо тепер до центрально означення даної роботи.

Визначення 2.1: диференційована у кожній точці x певного інтервалу I вектор-функція $\boldsymbol{y}(x)$, що належить області D та яка задовольняє систему (3.1) у всіх точках інтервалу I, називається розв'язком рівняння (2.1) на цьому інтервалі [2].

Нагадаємо одразу, що під диференційованістю вектор-функції, мається на увазі диференційованість кожної компоненти цієї вектор-функції. Іншими словами, для того, щоб вектор-функція була диференційованою, кожна з її складових повинна бути диференційованою, тобто мати похідну.

Визначення 2.2: графіки розв'язку системи (2.1) називаються інтегральними кривими цієї системи [2].

Відзначимо тут, що як правило диференціальні рівняння та їх системи мають нескінченну кількість розв'язків, що представляються у вигляді певної параметричної сім'ї. Поняття задачі Коші, яке ми уже використовували в розділі 1, але яке ми введемо в наступному визначенні більш строго, покликано виділити з цієї нескінченної сім'ї розв'язків лише один розв'язок, або принаймні певну конкретну підмножину розв'язків, що задовольнятимуть задану початкову умову.

Визначення 2.3: Якщо на додачу до системи (2.1) також задано певні початкові умови (2.3).

$$y_1(x_0) = y_{0,1} \dots y_m(x_0) = y_{0,m} \quad (2.3)$$

Або, у компактному вигляді (2.4),

$$\boldsymbol{y}(x_0) = \boldsymbol{y_0} \quad (2.4)$$

І де $(x_0;\ \boldsymbol{y_0}) \in D$, то кажуть, що задано задачу Коші для системи (3.1) та з початковими умовами (2.4).

Власне, задача Коші полягає у знаходженні такого розв'язку цієї системи, що задовольняв би умову (2.4) та був би визначеним принаймні у деякому околі точки $(x_0;\ \boldsymbol{y_0})$ [2].

З геометричної точки зору, задачу Коші можливо інтерпретувати, як задачу знаходження таких інтегральних кривих системи, що проходять через точку $(x_0;\ \boldsymbol{y_0})$, задану у умові (2.4).

В решті решт, введемо надважливе для методу Пікара (та й узагалі для теорії систем диференціальних рівнянь) визначення.

Визначення 2.4: Точка $(x_0;\ \boldsymbol{y_0})$ із області D називається точкою єдиності розв'язку задачі Коші (2.1)-(2.4), якщо для всяких двох розв'язків $\boldsymbol{y_m}(x)$ і $\boldsymbol{y_n}(x)$ цієї задачі Коші, існує такий інтервал Ĩ, що на якому $\boldsymbol{y_m}(x) = \boldsymbol{y_n}(x)$, і при чому x_0 теж належить Ĩ.

Визначення 2.5: Кажуть, що розв'язок задачі Коші є єдиним на певному інтервалі I, якщо кожна точка цього розв'язку на інтервалі I є точкою єдиності задачі Коші [2].

2.2 Важливі поняття теорії функціонального аналізу

В даному розділі ми представимо необхідні поняття з функціонального аналізу, які потрібні для формулювання теореми Банаха. Ми також надамо доведення цієї теореми, оскільки сам метод послідовних наближень базується на ній. Для ознайомлення з визначеннями цього розділу можна звернутися до підручника [1].

Розгляд ми почнемо з метричного простору, який можна розглядати як узагальнення множини дійсних чисел. У метричному просторі визначена відстань між будь-якими двома елементами множини.

Означення метричного простору – пара (M, ρ), де M - певна множина (простір), а $\rho(\varphi, \xi)$ - невід'ємна та однозначна дійсна функція (метрика), яка є визначеною для всяких двох елементів φ та ξ із множини M. Метрика задовольняє трьом аксіомам.

Трійка аксіом, які повинна задовольняти метрика, включає: невід'ємність, симетричність та нерівність трикутника. Ці аксіоми необхідні для забезпечення відповідних властивостей метричного простору. Власне кажучи, вони формулюються наступним чином :

а) $\rho(\varphi, \xi) = 0$ якщо і тільки якщо $\varphi = \xi$

б) $\rho(\varphi, \xi) = \rho(\xi, \varphi)$ $\qquad \forall\, \varphi, \xi, \eta \in P$

в) $\rho(\varphi, \eta) \leq \rho(\varphi, \xi) + \rho(\xi, \eta)$

Хоча простір дійсних чисел може служити прикладом метричного простору (у цьому дуже легко переконатися, скориставшись тим фактом, що, звичайно, для випадку дійсних чисел, аксіома в) є нерівністю трикутника), існує безліч інших множин, які можуть задовольняти аксіомам метричного простору. Наприклад, ми можемо розглянути простір елементами якого є вектори $\varphi = (\varphi_1, \dots, \varphi_m)$ з m функцій, визначених, неперервних на певному сегменті $|\tau - \tau_0| \leq d$ і таких, що $|\varphi_i - y_{0,i}| \leq Kd$, де константу K визначено з умови $|f_i(\tau, y_1 \dots y_m)| \leq K$. Цей простір ми

позначаємо C_m^* і ще повернемося до нього у наступному підрозділі. Але поки що у рамках демонстрації, для цього простору можемо просто ввести метрику (2.5), що задовольняє аксіомам метричного простору [1]:

$$\rho(\varphi, \psi) = max_{\tau,i}|\varphi_i - \psi_i| \qquad (2.5)$$

Легко перевірити, що всі три аксіоми виконуються для такого вибору метрики. Детально ми зупинятися на цьому не будемо, але відзначимо, що це добре продемонстровано, наприклад, у підручнику [1].

Далі введемо поняття околу елементу («точки») метричного простору.

Визначення 2.6: сукупність точок τ певного метричного простору P, які задовольняють умову (3.6):

$$U_\varepsilon(\tau_0) = \rho(\tau, \tau_0) < \varepsilon \text{ , де } \varepsilon \in \mathbb{R} \qquad (2.6)$$

Називають $\varepsilon -$ околом точки x_0 цього простору (також, її називають відкритою кулею, згідно геометричної інтерпретації цього об'єкту у просторі $\mathbb{R}^3$) [3].

Нам також буде потрібно визначення збіжності у метричному просторі. Його у підручнику [1] формулюють наступним чином:

Визначення 2.7 [1]: «послідовність $\{x_n\}$ елементів певного метричного простору P називають збіжною до елементу x^* цього простору, якщо виконується (3.7)»

$$lim_{n\to\infty}\rho(x_n, x^*) = 0 \qquad (2.7)$$

Ясно, що це визначення можна переформулювати також іншим чином [1]: «послідовність $\{x_n\}$ збігається до x^*, якщо будь-який окіл $U_\varepsilon(x^*)$ точки x^* містить усі точки послідовності $\{x_n\}$, починаючи з деякої. Точку x^* тоді називають границею $\{x_n\}$».

Перейдемо до чергового важливого для нас поняття: повного метричного простору.

Визначення 2.8 [1]: «послідовність точок $\{x_n\}$ з простору P називають фундаментальною, якщо $\forall\, \varepsilon > 0\ \exists\, N(\varepsilon) \in \mathbb{N}$, що $\forall\, n > N,\ m > N$ виконується умова (3.8)».

$$\rho(x_n, x_m) < \varepsilon \tag{2.8}$$

Це є аналогом критерію Коші з дійсного випадку.

Із третьої аксіоми метричного простору випливає безпосереднім чином, що всяка збіжна послідовність – є фундаментальна. І дійсно, нехай маємо послідовність x_n – що збіжна до x, тоді для заданого ε завжди можливо знайти певний номер N, такий що для всяких $n > N,\ m > N$: $\rho(x, x_n) < \frac{\varepsilon}{2}$ і $\rho(x, x_m) < \frac{\varepsilon}{2}$ і матимемо:

$$\rho(x_m, x_n) = \rho(x_m, x) + \rho(x, x_n) = \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon \tag{2.9}$$

Однак, обернене твердження уже не завжди вірне, в чому не важко переконатися. Що приводить нас до наступного визначення.

Визначення 2.9 [1]: Метричні простори, в яких всяка фундаментальна послідовність є збіжна, називаються повними.

Доведемо, наприклад, що простір неперервних на відрізку $[\alpha,\ \beta]$ функцій $C_{[a,b]}$ – є повний (що є, до речі, лише частковим випадком простору C_m^*).

Нехай, маємо повну фундаментальну послідовність $\{f_n\}$ цього простору, тоді, за визначенням, матимемо $\forall\, \varepsilon > 0\ \exists\, N(\varepsilon) \in \mathbb{N}$, такі, що матимемо $\forall\, n > N,\ m > N$:

$$max_{t \in [\alpha,\beta]} |f_m(t) - f_n(t)| < \varepsilon \tag{2.10}$$

Цілком ясно, що так як:

$$|f_m(t) - f_n(t)| \leq max_{t \in [\alpha,\beta]}|f_m(t) - f_n(t)| \quad \forall t \in [\alpha, \beta] \tag{2.11}$$

$$\text{то } |f_n(t) - f_m(t)| < \varepsilon \quad \forall t \in [\alpha, \beta] \tag{2.12}$$

А це, якраз, і є визначенням рівномірної збіжності функціональної послідовності. Тоді, із наслідку теореми про граничний перехід, а саме : «границя рівномірно збіжної послідовності функцій є неперервною функцією», прямо слідує той факт, що границя f цієї послідовності є неперервна. Перейдемо тоді до границі при $m \to \infty$ у попередній нерівності і матимемо:

$$|f_n(t) - f(t)| \leq \varepsilon \tag{2.13}$$

Для усіх значень t, а отже і для максимума при $n > N$. Це й доводить збіжність послідовності $\{f_n\}$ у сенсі метрики цього простору.

Цей же результат легко узагальнюється на випадок простору C_m^*, введеного трохи раніше, але викладки для нього є трохи більш громіздкими (хоч і мало у чому відрізняються від розглянутого випадку). Власне тому доведення для цього простору ми не наводитимемо тут.

Розглянемо тепер інші два дуже важливих визначення. Нехай P – метричний простір із ρ – метрикою у цьому просторі.

Визначення 2.10 [1]: «Відображення A: $P \to P$ називають стискуючим, якщо знайдеться дійсно число α < 1, таке що для всяких двох векторів y_1 і y_2 цього простору P, виконується така нерівність (2.14)»

$$\rho(Ay_1, Ay_2) \leq \alpha \cdot \rho(y_1, y_2) \tag{2.14}$$

Визначення 3.11 [1]: «Елемент $\hat{\tau}$ простору P називають нерухомою точкою відображення A, якщо $A\hat{\tau} = \hat{\tau}$. Інакше кажучи, нерухомі точки – це, по суті, розв'язки рівняння $A\hat{\tau} = \hat{\tau}$».

Теорема Банаха, або принцип стискаючих відображень: Будь-яке стискаюче відображення, визначене у повному метричному просторі, має одну і лише одну нерухому точку.

Доведення: Для початку, відмітимо, що будь-яке стискаюче відображення є також і неперервним.

І справді, якщо $\tau^* \to \tau$, то за визначенням оператора стиску, вірно також, що $\mathrm{A}\tau^* \to \mathrm{A}\tau$.

Взявши далі довільну точку τ^0 із даного метричного простору (повного) у якості початкової точки ітераційної процедури, побудуємо послідовність наступного вигляду (2.15):

$$\tau^1 = \mathrm{A}\tau^0, \tau^2 = \mathrm{A}\tau^1 = \mathrm{A}^2\tau^0 \dots \tau^n = \mathrm{A}\tau^{n-1} = \mathrm{A}^n\tau^0 \qquad (2.15)$$

Видно, що побудована послідовність – є фундаментальна. Дійсно, припустивши, що n ≤ m, матимемо (2.16):

$$\begin{gathered}\rho(\tau^n, \tau^m) = \ \rho(\mathrm{A}^n\tau^0, \mathrm{A}^m\tau^0) \leq \ \alpha^n \cdot \rho(\tau^0, \tau^{m-n}) \leq \\ \leq \alpha^n \cdot \left(\rho(\tau^0, \tau^1) + \rho(\tau^1, \tau^2) + \cdots + \rho(\tau^{m-n-1}, \tau^{m-n})\right) \leq \\ \leq \alpha^n \cdot \rho(\tau^0, \tau^1) \cdot (1 + \alpha + \alpha^2 + \cdots + \alpha^{m-n-1}) \leq \alpha^n \cdot \rho(\tau^0, \tau^1) \cdot \frac{1}{1-\alpha}\end{gathered} \qquad (2.16)$$

І так як $\alpha < 1$, то взявши досить велике значення n, цю величину можемо зробити якзавгодно малою. Таким чином виходить, що послідовність й справді фундаментальна, а отже у даному метричному просторі вона має границю.

Нехай, наприклад:

$$lim(\tau^n) \ = \ \tau^*, n \to \infty \qquad (2.17)$$

Тоді використовуючи той факт, що відображення А є неперервним, ми матимемо наступний ланцюг (2.18):

$$A\,\tau^* = A\,lim_{n\to\infty}(\tau^n)\; = lim_{n\to\infty}(A\tau^n)\; = lim_{n\to\infty}(\tau^{n+1})\; = \tau^* \qquad (2.18)$$

Який доводить існування нерухомої точки. Тепер достатньо довести єдиність такої цієї точки:

Якщо $Ax = x$ і $Ay = y$, то за визначенням стискаючого оператора, маємо (2.19):

$$\rho(x,y) = \;\rho(Ax,Ay) \leq \;\alpha\rho(x,y) \qquad (2.19)$$

І так як $\alpha < 1$, це означає, що єдине можливе значення для $\rho(x,y) -$ це 0 (отже $\rho(x,y) = 0$), тобто $x = \;y$.

Теорема доведена.

2.3 Класичний Метод Пікара послідовних наближень для систем диференціальних рівнянь

Нарешті, ввівши усі необхідні нам визначення та надавши доведення теореми Банаха, можемо перейти до самої суті цієї роботи – розгляду методу Пікара. І почнемо ми, звичайно, з короткого нагадування того, у чому полягає класичний метод Пікара послідовних наближень. Заодно, ми безпосередньо пояснимо у чому полягає дослідницька задача у даній роботі.

Отже, нехай маємо задачу Коші для системи рівнянь. У якості винятка, у цьому розділі користуватимемося «розлогим» записом (а не векторним), щоб спростити розуміння викладок і запишемо, власне, систему як (2.20):

$$\begin{cases} y_1' = f_1\big(x, y_1(x) \dots y_m(x)\big) \\ \dots \\ y_m' = f_m\big(x, y_1(x) \dots y_m(x)\big) \end{cases} \qquad (2.20)$$

За умов:

$$y_1(x_0) = y_{0,1} \dots y_m(x_0) = y_{0,m} \quad (2.21)$$

Нехай тепер знову D – певна область, визначена в $\mathbb{R}^{m+1}$. Розглянемо визначені, неперервні та ліпшицеві у цій області функції $f_i(x, y_1 \dots y_m)$, $i = \overline{1,m}$.

Одразу тут же нагадаємо, що умова Ліпшиця в випадку систем диференціальних рівнянь формулюється [2] наступним чином.

Вектор-функція $\bar{f}(x, \bar{y}) : D \to \mathbb{R}^m$ називається локально ліпшицевою, щодо $\bar{y}$, якщо для кожної точки $(x_0, \bar{y}_0)$ із області D знайдуться такі числа $r = r(x_0, \bar{y}_0) > 0$ та $L = L(x_0, \bar{y}_0) > 0$, що матимемо (2.22):

$$\left|\left|\bar{f}(x, \bar{y}') - \bar{f}(x, \bar{y}'')\right|\right| \le L\left|\left|\bar{y}' - \bar{y}''\right|\right| \; \forall (x, \bar{y}'), (x, \bar{y}'') \in B_r(x_0, \bar{y}_0) \subset D \quad (2.22)$$

Тут $B_r(x_0, \bar{y}_0)$ – позначає $(m+1)$-вимірну кулю, радіуса r із центром у точці $(x_0, \bar{y}_0)$.

Також відомо, що на певному інтервалі $|x - x_0| \le d$ існуватиме єдиний розв'язок $y_i = \varphi_i(x), i = \overline{1,m}$ такої задачі Коші [2, с. 392].

Слідуючи [1, с. 80] і як уже було згадано, розглянемо простір C_m^*, елементами якого є вектори $\varphi = (\varphi_1, \dots, \varphi_m)$ з m функцій, визначених, неперервних на певному сегменті $|x - x_0| \le d$ і таких, що $\left|\varphi_i - y_{0,i}\right| \le Kd$, де константу K визначено з умови $|f_i(x, y_1 \dots y_m)| \le K$.

Метрика у цьому просторі визначається наступним чином :

$$\rho(\varphi, \psi) = max_{x,i}|\varphi_i - \psi_i| \quad (2.23)$$

Як ми уже відзначали, цей метричний простір є повним і до того ж, наступний оператор є стискуючим оператором у цьому просторі :

$$A\big(\varphi_1(x) \dots \varphi_m(x)\big) : \{y_{0,i} + \int_{x_0}^{x} f_i\big(\tau, \varphi_1(\tau), \dots, \varphi_m(\tau)\big)d\tau \,; i = \overline{1,m}\} \quad (2.24)$$

Таким чином, згідно принципу Пікара-Банаха стискаючих відображень, ми можемо побудувати ітераційну процедуру знаходження наближеного розв'язку $\varphi_i{}^{(n)}(x), i = \overline{1,m}$ задачі Коші (2.20)-(2.21) :

$$\varphi_i{}^{(j)}(x) = \; y_{0,i} + \int_{x_0}^{x} f_i\big(\tau, \varphi_1{}^{(j-1)}(\tau), \dots, \varphi_m{}^{(j-1)}\big)d\tau \qquad (2.25)$$

Тут за допомогою j – позначається черговий крок ітерації, а i – звичайно, позначає індекс рівняння, для відповідного розв'язку.

Відзначимо також, що у якості $(\varphi_1{}^{(0)}, \dots, \varphi_m{}^{(0)})$ можна брати абсолютно будь-які неперервні в D функції, але здебільшого беруться константи (часто хорошим початковим наближенням є $\varphi_i{}^{(0)} = y_i$, адже згідно початкової умови, розв'язки рівняння повинні проходити через ці точки, тому ми одразу починаємо «там де потрібно»).

Тут же одразу і видно, що, у випадку, коли функції $f_i\big(\tau, \varphi_1(\tau), \dots, \varphi_m(\tau)\big)$ є неполіноміальними, то інтегрування у ітераційній процедурі може привести до надзвичайно складних квадратур, які можуть навіть і не виражаються у елементарних функціях. Це, звичайно, значним чином ускладнює застосування цього методу на практиці, зокрема на ЕОМ. Саме тому, ми пропонуємо у цій роботі, модифікувати цей метод так, щоб уникнути інтегрування «складних» функцій та, в певному сенсі, обмежитися лише інтегруванням поліномів, що спрощує розрахунки на ЕОМ. Ця модифікація описана у наступному підрозділі.

2.4 Опис модифікації методу

Перейдемо тепер до опису модифікації методу Пікара послідовних наближень, який дозволяє спростити його застосування на ЕОМ і який є центральною частиною цієї роботи. У цьому розділі також використаємо певні з викладок, описаних у роботах [1, 8].

Отже, як вже було відмічено у попередньому розділі, наша мета полягає в уникненні повторного інтегрування складних функцій у процесі ітерацій класичного методу Пікара. Один зі способів вирішення цієї проблеми полягає в наближенні підінтегральних функцій поліномами. Для цього ми можемо застосувати формулу Тейлора для функцій однієї змінної до функцій $f_i\big(\tau, \varphi_1(\tau), \dots, \varphi_m(\tau)\big)$, розклавши їх до k-го порядку в точці a (нехай розглядається інтервал [a, b]) та відкинути залишковий член. Припускається, що функції $f_i\big(\tau, \varphi_1(\tau), \dots, \varphi_m(\tau)\big)$ мають необхідну кількість частинних похідних по всім змінним. Для нульового наближення ми також використовуємо достатньо гладку функцію, таким чином :

$$f_i\big(\tau, \varphi_1(\tau), \dots, \varphi_m(\tau)\big) \approx \sum_{n=0}^{k} \frac{{f_i}^{(n)}(a, \varphi_1(a), \dots, \varphi_m(a))}{n!} (\tau - a)^n \qquad (2.26)$$

Уточнимо, що тут, під ${f_i}^{(n)}$ ми розуміємо повну n-ту похідну функції f_i, як похідну складної функції аргументу τ, тобто наприклад

$$f'_{i,\tau}\big(\tau, \varphi(\tau)\big) = \frac{\partial f_i}{\partial \tau} + \frac{\partial f_i}{\partial \varphi_1}\frac{d\varphi_1}{d\tau} + \dots + \frac{\partial f_i}{\partial \varphi_m}\frac{d\varphi_m}{d\tau} \qquad (2.27)$$

При заміні підінтегральних функцій їх частковими сумами ряду Тейлора, оператор (2.25), який для уникнення плутанини ми в подальших викладках позначатимемо як $\tilde{A}\varphi(x)$, приймає наступний вигляд :

$$\tilde{A}\big(\varphi_1(x),\dots,\varphi_m(x)\big):\{y_{0,i}+\int_a^x\sum_{n=0}^{k}\frac{f_i^{(n)}\big(a,\varphi_1(a),\dots,\varphi_m(a)\big)}{n!}(\tau-a)^n\ d\tau\ ;i=\overline{1,m}\}=$$

$$=\left\{y_{0,i}+\sum_{n=0}^{k}\frac{f_i^{(n)}\big(a,\varphi_1(a),\dots,\varphi_m(a)\big)}{(n+1)!}(x-a)^{n+1}\ ;i=\overline{1,m}\right\} \qquad (2.28)$$

Звісно, виникає питання про те, чи збігається метод при такому підході до наближення підінтегральної функції і, відповідно, до нової форми оператора. Ми розглянемо це питання детальніше у наступному розділі.

2.5 Збіжність модифікації методу Пікара

Для того, щоб довести збіжність цієї модифікації, ми покажемо, що оператор (2.28), при такій заміні підінтегральної функції, залишається оператором стиску.

Дійсно, розглянемо для цього відстань між $\tilde{A}\varphi^1$ та $\tilde{A}\varphi^2$:

$$\rho\big(\tilde{A}\varphi^2,\tilde{A}\varphi^1\big)=$$

$$=max_{x,i}\left|\sum_{n=0}^{k}\frac{f_i^{(n)}\big(a,\varphi_1^2(a),\dots,\varphi_m^2(a)\big)-f_i^{(n)}\big(a,\varphi_1^1(a),\dots,\varphi_m^1(a)\big)}{(n+1)!}(x-a)^{n+1}\right|\leq$$

$$\leq(*)$$

Тоді зафіксуємо значення n, і розглянемо отриману різницю під знаком суми.

Приймаючи до уваги той факт, що $f_i^{(n)}\big(a,\varphi_1(a),\dots,\varphi_m(a)\big)$ є, фактично, функціями багатьох змінних $(a,\varphi_1,\dots,\varphi_m)$, то можемо застосувати до них теорему

Лагранжа про скінченні прирости для випадку функцій багатьох змінних і для зручності запису, далі позначатимемо $f^{(n)}(\tau, y_1, \dots, y_m) \equiv F(\tau, y_1, \dots, y_m)$):

$$|f_i^{(n)}\left(a, \varphi_1^2(a), \dots, \varphi_m^2(a)\right) - f_i^{(n)}\left(a, \varphi_1^1(a), \dots, \varphi_m^1(a)\right)| \equiv$$
$$\equiv |F\left(a, \varphi_1^2(a), \dots, \varphi_m^2(a)\right) - F\left(a, \varphi_1^1(a), \dots, \varphi_m^1(a)\right)| =$$
$$= |0 + F'_{y_1}(\zeta,\ \xi_1, \dots,\ \xi_m\ )(\varphi_1^2(a) - \varphi_1^1(a)) + \dots + F'_{y_m}(\zeta,\ \xi_1, \dots,\ \xi_m\ )(\varphi_m^2(a) - \varphi_m^1(a))|$$
$$\leq |\text{Позначимо } M_i = \max_k |F'_{y_k}(\zeta,\ \xi_1, \dots,\ \xi_m\ )| \leq$$
$$\leq M_i \cdot m \cdot max_{x,i}\left(\varphi_i^2(a) - \varphi_i^1(a)\right) =$$
$$= M_i \cdot m \cdot \rho(\varphi^2, \varphi^1) \qquad (2.29)$$

Важливою відмітити, що M_i – завжди визначене та існує, в сенсі максимума неперервної на відрізку функції.

Підберемо тепер кінець інтервалу інтегрування b таким чином, щоб виконувалась наступна умова:

$$\begin{cases} k \cdot M_i \cdot m \cdot |b - a|^{k+1} < 1, \text{якщо } b > a + 1 \\ \text{або } k \cdot M_i \cdot m \cdot |b - a|\ < 1, \text{якщо } b < a + 1 \end{cases}$$

Матимемо, тоді (2.30) :

$$(*) \leq max_{x,i}\left|\sum_{n=0}^{k} M_i \cdot m \cdot \rho(\varphi^2, \varphi^1) \cdot \frac{(x-a)^{n+1}}{(n+1)!}\right| \leq$$
$$\leq |\text{Для визначеності припустимо, що } b < a + 1;\ \text{і нехай } M = \ max_i(M_i)| \leq$$
$$\leq k \cdot M \cdot m \cdot (b - a)\ \cdot \rho(\varphi^2, \varphi^1) \leq \alpha \cdot \rho(\varphi^2, \varphi^1)$$
$$M \cdot m \cdot (b - x0)\ = a \qquad (2.30)$$

Так як кінець інтервалу інтегрування b ми можемо вибирати на наш розсуд, то це дозволяє зробити $\alpha < 1$.

Отже, маємо (3.31):

$$\rho\left(\tilde{A}\varphi^2, \tilde{A}\varphi^1\right) \leq \alpha \cdot \rho(\varphi^2, \varphi^1) \qquad (2.31)$$

Де $\alpha < 1$, це і доводить факт того, що оператор $\tilde{A}$ є стискуючим оператором.

2.6 Оцінка похибки модифікації методу

Хоч ми і довели, що метод збіжний, нам це, насправді, мало про що каже. Адже далі нам ще потрібно довести, що при застосуванні такої модифікації методу, вона дасть результат близький до того, який ми би отримали, якби не застосовували розкладення підінтегральної в ряд Тейлора.

Для цього, ми можемо оцінити різницю, у сенсі метрики простору C_m^*, , між наближенням розв'язку системи, яке ми б отримали з класичним методом Пікара і наближенням, яке ми отримаємо про описаній модифікації методу.

Знову будемо позначати як $\tilde{A}\varphi(x)$ оператор $A\varphi(x)$, де підінтегральну функцію розкладаємо у ряд Тейлора, тоді достатньо оцінити різницю між діями цих двох операторів $\tilde{A}$ і A на одну і ту ж вектор-функцію $\varphi(x)$.

Цілком зрозуміло, що так як різниця між операторами $\tilde{A}$ і A полягає тільки в тому, що в одному з них підінтегральну функцію розкладають у ряд Тейлора, то різниця їх дії на одну і ту ж вектор-функцію виражатиметься інтегралом від залишкового члену цього ряду Тейлора, а саме матимемо (2.32):

$$\rho\big(A\varphi,\tilde{A}\varphi\big)=max_{i,x}\left|\int_a^x\frac{(\tau-a)^{k+1}}{(k+1)!}f_i^{(n+1)}\big(\xi,\varphi_1(\xi),\dots,\varphi_m(\xi)\ \big)d\tau\right|=$$

$$=max_{i,x}\left|\frac{(x-a)^{k+2}}{(k+2)!}f_i^{(n+1)}\big(\xi,\varphi_1(\xi),\dots,\varphi_m(\xi)\ \big)\right|\le$$

$$\le\left|\text{Позначимо }\widetilde{M}\equiv max_{i,x}\left(f_i^{(n+1)}\big(\xi,\varphi_1(x),\dots,\varphi_m(x)\ \big)\right)\right|\le$$

$$\le\frac{\widetilde{M}\cdot max_{x,i}|(x-a)^{k+2}|}{(k+2)!}\le\frac{\widetilde{M}\cdot(b-a)^{k+2}}{(k+2)!}\qquad(2.32)$$

У цій формулі, ми можемо підібрати кінець інтервалу інтегрування b таким чином, щоб отримане значення було меншим за певне задане число Δ_1.

$$\frac{\widetilde{M}\cdot(b-a)^{k+2}}{(k+2)!}<\Delta_1\qquad(2.33)$$

Тепер розглянемо різницю:

$$\rho\big(A^2\varphi,\tilde{A}^2\varphi\big)=\rho\left(A(A\varphi),\tilde{A}\big(\tilde{A}\varphi\big)\right)=$$

$$=max_{x,i}\left|\int_a^x[f_i\ (\tau,A\varphi_1(\tau),\dots,A\varphi_m(\tau)\ )\right.$$

$$\left.-\sum_{n=0}^{k}\frac{f_i^{(n)}\left(a,\tilde{A}\varphi_1(a),\dots,\tilde{A}\varphi_2(a)\right)}{n!}(x-a)^n]d\tau\right|=$$

$$=|\text{розкладемо перший доданок у ряд Тейлора у точці }a|=$$

$$=max_{x,i}\left|\int_a^x\sum_{n=0}^{k}\frac{f_i^{(n)}\big(a,A\varphi_1(a),\dots,A\varphi_2(a)\big)-f_i^{(n)}\left(a,\tilde{A}\varphi_1(a),\dots,\tilde{A}\varphi_2(a)\right)}{n!}(x-a)^n\right.$$

$$\left.+R_n(\tau)d\tau\right|\le$$

$$\leq |\text{Викор. т. Лагранжа; так само, як і при доведенні збіжності}|$$

$$\leq max_{x,i}\left|\int_a^x \sum_{n=0}^{k} \frac{m \cdot M \cdot max_{x,i}|\tilde{A}\varphi(\eta) - A\varphi(\eta)|}{n!}(x-a)^n + R_n(\tau)d\tau\right| \leq$$

$$\leq max_x\left|\Delta_1 + \int_a^x \sum_{n=0}^{k} \frac{m \cdot M \cdot \delta_1}{n!}(x-a)^n d\tau\right| \leq$$

$$\leq |\text{Маємо коефіцієнт } \alpha \text{ із попереднього доведення (після інтегрування)}| \leq$$

$$\leq \Delta_1 + \alpha \cdot \Delta_1 = \Delta_1(\alpha+1) \equiv \Delta_2 \qquad (2.34)$$

Точно таким же чином, ми можемо отримати наступну нерівність:

$$\rho\left(A^3\varphi, \tilde{A}^3\varphi\right) < \Delta_2(\alpha+1) = \Delta_1(\alpha+1)^2 \qquad (2.35)$$

І так далі до s-того наближення:

$$\rho\left(\tilde{A}^s\varphi, A^s\varphi\right) < \Delta_{s-1}(\alpha+1) = \Delta_1(\alpha+1)^{s-1} = \frac{\widetilde{M} \cdot (\alpha+1)^{m-1} \cdot (b-a)^{k+2}}{(k+2)!} \qquad (2.36)$$

Зважаючи на те, що факторіал зростає швидше за будь-який скінченний поліном, то для забезпечення збіжності модифікації методу до справжнього розв’язку, достатньо просто розкладати підінтегральну функцію до «досить високого» порядку (звичайно, треба також зважати на вибір кінця інтервалу інтегрування b).

Отриману формулу, разом з похибкою метода Пікара, можемо вважати загальною похибкою застосування цього методу при розкладенні підінтегральної функції у ряд Тейлора.

Таким чином, нами доведено, що метод Пікара для систем диференціальних рівнянь, при розкладенні у ньому підінтегральної функції у ряд Тейлора, дійсно буде збіжним до справжнього розв’язку.

2.7 Прийом для спрощення розрахунків на ЕОМ

У цьому підрозділі, ми пропонуємо спеціальний прийом для спрощення розрахунків та економії оперативної пам’яті при застосуванні цього алгоритму на ЕОМ.

Приймаючи до уваги те, як працюють сучасні ЕОМ, при практичних розрахунках, не завжди зручно зберігати у пам’яті усі попередні наближення, які ми знайшли. Дійсно, для знаходження наступного наближення розв’язку системи, при реалізації цього алгоритму на ЕОМ, оператор $\tilde{A}$ застосовується до правих частин рівнянь не одночасно, а «по порядку», тобто спочатку знаходимо наступне наближення ${\varphi_1}^{(l+1)}(x)$, далі ${\varphi_2}^{(l+1)}(x)$ і так далі аж до ${\varphi_m}^{(l+1)}(x)$. Звичайно, ми припускаємо, що код не є паралелізованим, але паралелізація і векторизація цього алгоритму виходить за рамки даної роботи і потребує подальших досліджень

Власне, сам прийом опирається на ту ж ідею, що й метод Гауса-Зейделя, і полягає у тому, що так як для знаходження наступного наближення використовуються, власне кажучи, попередні наближення розв’язків, при знаходженні ${\varphi_i}^{(l+1)}(x)$ ми можемо використати уже знайдені наближення ${\varphi_1}^{(l+1)}(x), \dots, {\varphi_i}^{(l+1)}(x)$, замість ${\varphi_1}^{(l)}(x), \dots, {\varphi_i}^{(l)}(x)$. Це може трохи пришвидшити збіжність алгоритму, а також дозволяє ефективніше використовувати оперативну пам’ять ЕОМ.

До того ж, така модифікація є цілком обґрунтованою з точки зору збіжності самого алгоритму, адже як ${\varphi_i}^{(l+1)}$ так і ${\varphi_i}^{(l)}$ є поліномами (тобто належать C_m^*), а отже можемо застосувати $\tilde{A}$ до ${\varphi_1}^{(l+1)}(x), \dots, {\varphi_i}^{(l+1)}(x), {\varphi_i}^{(l)}(x), \dots, {\varphi_m}^{(l)}(x)$, а не до ${\varphi_1}^{(l)}(x), \dots, {\varphi_i}^{(l)}(x), {\varphi_i}^{(l)}(x), \dots, {\varphi_m}^{(l)}(x)$ і так як $\tilde{A}$ є оператором стиску у цьому просторі C_m^*, то алгоритм дійсно буде збіжним. Звичайно, тип збіжності так і залишиться лінійним, тобто нам усе ще теоретично необхідно було б зробити нескінченну кількість ітерацій, щоб отримати точний розв’язок, але емпірично при

такій модифікації алгоритму, він швидше збігається до наближеного розв'язку із заданою точністю (див. результати у відповідному розділі).

2.8 Процедура спряження локальних розв'язків на краях області збіжності

Отже, в попередніх розділах нами було доведено збіжність модифікації методу Пікара, при заміні підінтегральних функцій їх рядами Тейлора, до справжнього розв'язку, але разом із цим, нам довелося також накласти деякі додаткові умови на вектор-функції в правій частині системи при доведенні. До того ж, у цьому випадку, збіжність методу Пікара значним чином залежить від вибору кінця інтервалу інтегрування. Це і не дивно, адже ряд Тейлора дає «непогане» наближення лише локально.

Зважаючи на це, ще раз трохи модифікуємо алгоритм методу, щоб прийняти до уваги ці обмеження та мати змогу інтегрувати систему на інтервалі більшому, ніж початкова область збіжності. Введемо процедуру спряження локальних розв'язків на краях області збіжності. Не обмежуючи загальності, ми можемо припустити, що початкові умови задано на початку інтервалу інтегрування, а саме : $\boldsymbol{y_0} = \boldsymbol{y}(\alpha)$, при чому сам інтервал інтегрування I – $[\alpha, \beta]$.

Розіб'ємо тоді інтервал I на n частин, які позначатимемо $[\alpha_i, \beta_i]$, де відповідно $\alpha_1 = \alpha, \beta_n = \beta, i = 1 \dots n$. І при чому $b_{i-1} = \alpha_i$.

Далі розкладемо кожну компоненту вектор-функції $\boldsymbol{f}(\tau, \boldsymbol{\varphi^{n-1}}(\tau))$ у її ряд Тейлора в певній точці α_1 (тобто там, де задана початкова умова) і знайдемо наближення розв'язку за ітераційною процедурою на першому із підінтервалів $[\alpha_1, b_1]$, у припущенні, що інтервал розбито таким чином, що кінець інтервалу b_1 було підібрано так, що умови збіжності, отримані вище, були виконані.

Можемо тепер знайти чергове наближення $\boldsymbol{y}(\beta_1) = \boldsymbol{y}(\alpha_2) \approx \boldsymbol{\varphi^n_{[\alpha_1, \beta_1]}}(x)$. Приймаючи до уваги, що $\boldsymbol{y}(\alpha_1)$ лежить на тих же інтегральних кривих, що і $\boldsymbol{y}(\alpha_2)$, то

розв’язок задачі Коші для системи із початковими умовами $\boldsymbol{y}(\alpha_1) = \boldsymbol{y_0}$ (якщо виконані умови збіжності, тобто він існує і єдиний) проходитиме через $\boldsymbol{y}(\alpha_2)$, тому можемо повторити всі ті ж самі викладки для точки α_2. І так далі аж до α_n. В результаті матимемо n вектор-функцій, які на відповідному інтервалі наближають одні й ті самі інтегральні криві.

Суть описаного алгоритму проілюструємо на рисунку 2.1. Для простоти зобразимо випадок з лише одним рівнянням:

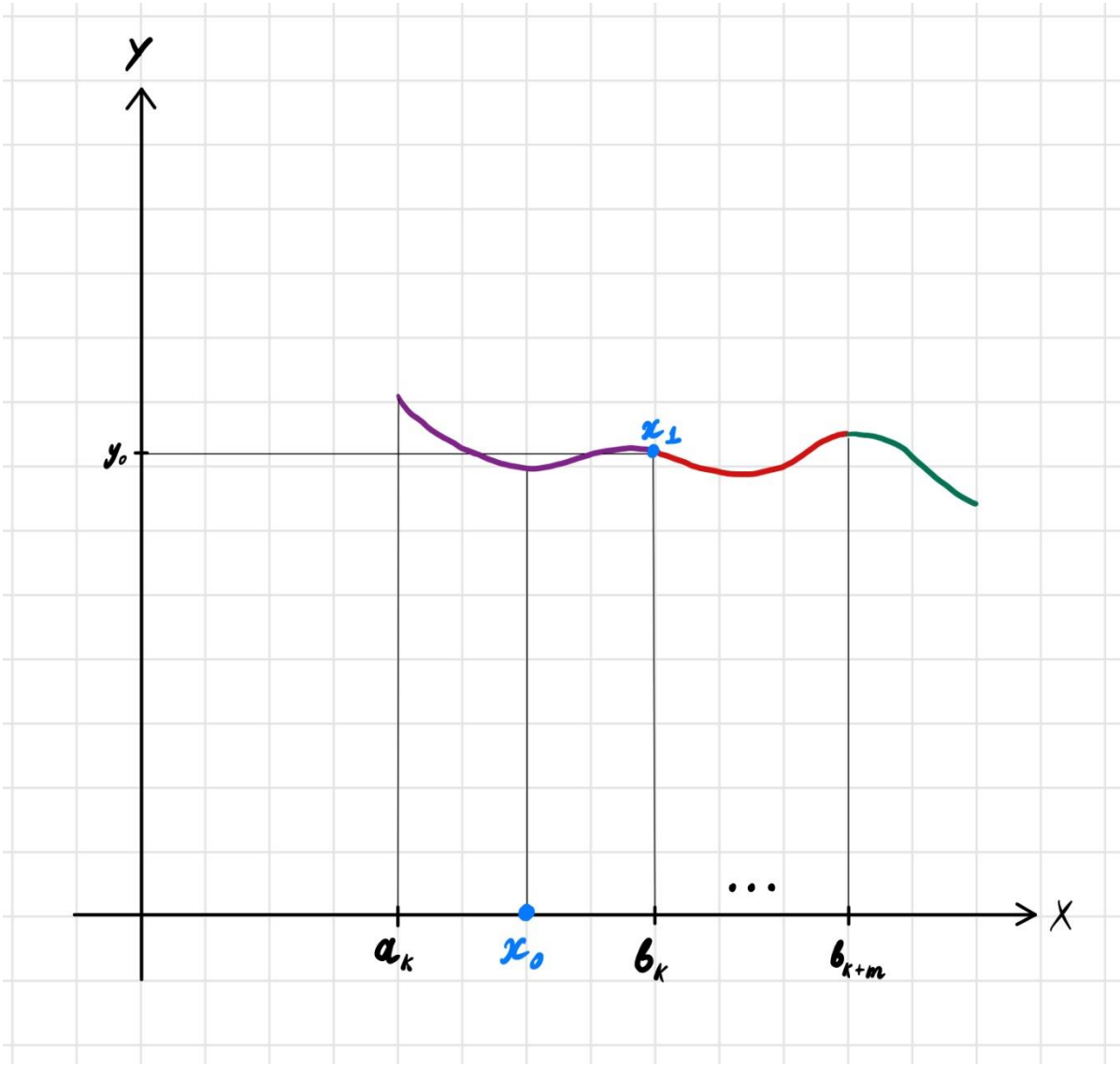


Рисунок 2.1 - Схема знаходження наближення розв’язку на інтервалі, більшому за початкову область збіжності, для одновимірного випадку

Тут різними кольорами позначено розв’язки, знайдені у локальних областях збіжності.

Варто також відзначити, що чим більшим обрати значення n й чим до вищого порядку ми розкладаємо підінтегральні функції, тим точніше матимемо наближення розв’язку задачі Коші для системи на усьому інтервалі інтегрування.

2.9 Метод Рунге-Кутта 4-го порядку

Ми уже частково розглядали методи Рунге-Кутта у розділі 1, проте ми включаємо більш детальний опис цього методу тут, адже він безпосередньо буде використаний у системі розв’язання систем диференціальних рівнянь, для порівняння результатів із методом Пікара.

Таким чином, з основною ідеєю методу ми уже ознайомилися, тепер отримаємо метод Рунге-Кутта 4-го порядку точності із неявної формули, отриманої у розділі 1, так само як це запропоновано у праці [4].

Почнемо, власне кажучи, із розгляду неявної формули (знову ж таки, називається вона неявною, адже нам невідомі значення $y\left(\tau_{i+\frac{1}{2}}\right)$ та $y(\tau_{i+1})$) :

$$y(\tau_{i+1}) = y(\tau_i) + \frac{h}{6}\left(f\big(\tau_i, y(\tau_i)\big) + 4f\left(\tau_{i+\frac{1}{2}}, y\left(\tau_{i+\frac{1}{2}}\right)\right) + f\big(\tau_{i+1}, y(\tau_{i+1})\big)\right) \qquad (2.36)$$

Для простоти запису, позначимо $y(\tau_k) \equiv y_k$.

Неявну частину можна обчислити наступним чином:

$$4f\left(\tau_{i+\frac{1}{2}}, y_{i+\frac{1}{2}}\right) + f(\tau_{i+1}, \tau_{i+1}) =$$

$$= 2f\left(\tau_{i+\frac{1}{2}}, y_i + \frac{h}{2}f(\tau_i, y_i)\right) + 2f(\tau_{i+1}, y_i + \frac{y_i + hf(\tau_i, y_i)/2}{2}) +$$

$$+ f(\tau_{i+1}, y_i + \frac{y_i}{2} + \frac{y_i + hf(\tau_i, y_i)/2}{4}) \quad (2.37)$$

Отриманий метод можемо записати у вигляді такої ітеративної процедури:

$$\begin{cases} \boldsymbol{y_{i+1}} = \Delta \boldsymbol{y_i} + \boldsymbol{y_i}, \\ \Delta \boldsymbol{y_i} = \frac{1}{6}\left[\boldsymbol{k_i^1} + 2\boldsymbol{k_i^2} + 2\boldsymbol{k_i^3} + \boldsymbol{k_i^4}\right], \\ \boldsymbol{k_i^1} = h\boldsymbol{f}(\tau_i, \boldsymbol{y_i}), \\ \boldsymbol{k_i^2} = h\boldsymbol{f}\left(\tau_{i+\frac{1}{2}}, \boldsymbol{y_i} + \frac{\boldsymbol{k_i^1}}{2}\right), \\ \boldsymbol{k_i^3} = h\boldsymbol{f}\left(\tau_{i+\frac{1}{2}}, \boldsymbol{y_i} + \frac{\boldsymbol{k_i^2}}{2}\right), \\ \boldsymbol{k_i^4} = h\boldsymbol{f}\left(\tau_{i+1}, \boldsymbol{y_i} + \boldsymbol{k_i^3}\right). \end{cases} \quad (2.38)$$

Зважаючи на те, що метод Рунге-Кутта опирається на інтерполяційний метод Сімпсона, що у свою чергу має 4-й порядок точності відносно кроку, то і метод Рунге-Кутта матиме також 4-й порядок точності [детальніше, 4, 5, 7]. Метод дає досить точні результати, якщо обраний крок є відносно невеликим, тому, зважаючи на простоту реалізації цього методу, його було доцільно використати для порівняння результатів, отриманих за методом Пікара послідовних наближень, що і зроблено у даній роботі.

2.10 Висновки до розділу

За результатами цього розділу, було досліджено математичне забезпечення для системи розв'язання матричних диференціальних рівнянь методом Пікара послідовних наближень. В першу чергу, були розглянуті фундаментальні для

викладок, поняття з теорії диференціальних рівнянь, такі як поняття розв'язку системи, задача Коші для систем, існування розв'язку та постановка задачі. Також, нами було розглянуто деякі з основних понять з теорії функціонального аналізу, у тому числі метричні простори, повні метричні простори, збіжність і фундаментальність у метричних просторах, доведення принципу Пікара-Банаха стискаючих відображень.

Основною суттю даного розділу, тим не менш, було введення модифікації класичного методу Пікара, шляхом заміни у ньому підінтегральних функцій, їх рядами Тейлора, що покликано уникнути інтегрування складних неполіноміальних функцій і таким чином спростити розрахунки на практиці, а зокрема на ЕОМ. Було доведено, що така модифікація буде збіжна і саме до справжнього розв'язку. Для цієї модифікації було також оцінено "глобальну" похибку. Таким чином, застосування на ЕОМ стало значно більш ефективним і простим.

Для подальшого спрощення розрахунків на ЕОМ був введений спеціальний прийом обчислення наступного. Було показано, що такий прийом дозволяє підвищити точність розрахунків, зменшити час їх виконання та зекономити оперативну пам’ять ЕОМ.

Ми також продемонстрували яким чином можливо провести спряжування розв'язків на краях області збіжності таким чином, щоб можна було проводити інтегрування систем диференціальних рівнянь за нашим методом не тільки у області збіжності, а й на інтервалах, більших за неї. Тим не менш, таке спряжування, звичайно, тягне за собою накопичення похибки, дослідження якої виходить за рамки цієї роботи.

На сам кінець, ми коротко описали метод Рунге-Кутта 4-го порядку точності, відносно кроку, для його подальшого порівняння з модифікованим методом Пікара.

Підводячи підсумки, результати даного дослідження є важливими як з практичної, так і з наукової точок зору. З одного боку, вони дозволяють зменшити час розрахунків на ЕОМ та зробити їх більш ефективними. А з іншого, представляють теоретичний інтерес з перспективи продовження досліджень, стосовно збіжності

наближених методів розв’язання систем диференціальних рівнянь та оцінки їх похибок.

3 ОПИС ПРОГРАМНИХ ЗАСОБІВ

3.1 Опис розроблених алгоритмів

У цьому розділі будуть описані розроблені, на основі результатів отриманих у розділі 2, програмні засоби. Ми почнемо із загального огляду побудованої системи та розглядатимемо кожен з підмодулів у більших деталях в подальших розділах. Ми також наведемо певні теоретичні результати, які стосуватимуться алгоритмічної складності розробленого модифікованого алгоритму методу Пікара послідовних наближень (її ми порівняємо зі складністю метода Рунге-Кутта). Насамкінець будуть наведені тестові випадки для, власне кажучи, тестування коректного функціонування системи, які будуть виконані та їх результати наведені у розділі 4 цієї роботи. Отже, перейдемо нарешті до розгляду системи.

Розроблена програма представлена у вигляді веб-орієнтованого застосунку, розміщеного на безкоштовному хостигну pythonenywhere. Користувач взаємодіє з даним застосунком за допомогою графічного інтерфейсу головної веб-сторінки застосунку, вводячи дані, як кількість рівнянь у системі, самі рівняння, початкові умови тощо у відповідні поля.

Користувач має змогу розв’язати введену систему диференціальних рівнянь модифікованим методом Пікара, що був описаний в деталях у розділі 2, або методом Рунге-Кутта четвертого порядку точності (або, звичайно, обома методами для порівняння).

Веб-застосунок може бути проконсультований за наступною веб-адресою: http://picardsolver.pythonanywhere.com/

Також, на додачу до лістингу програми, наведеного у додатках, увесь програмний код доступний за наступною адресою: https://github.com/vladherasymenko/PicardSolver. Код отримуватиме регулярні оновлення саме на платформі GitHub, тому перевагу, при виборі вихідного коду для

реімплементації на власній ЕОМ, варто надавати саме наведеному веб-посиланню на GitHub репозиторій.

Відмітимо одразу, що звичайно, основним функціоналом системи – є розв’язання задачі Коші для систем диференціальних рівнянь модифікованим методом Пікара, а метод Рунге-Кутта було імплементовано лише для порівняння з методом Пікара. Останній, тим не менш, фундаментально відрізняється від методу Пікара, адже він є табличним («сітковим») методом, у той час як метод Пікара – це метод «наближений», що дає розв’язок у вигляді певної аналітичної функції. Саме тому порівняння між двома методами пропонується проводити візуально, адже не існує достатньо інформативної метрики, яка могла б виміряти різницю точності між функцією і таблицею значень.

Переходячи до розгляду самих алгоритмів, реалізованих у рамках розробки системи, наведемо на рисунку 3.1 загальну схему функціонування системи, а у подальших підрозділах розглянемо кожну з підсистем більш детально.

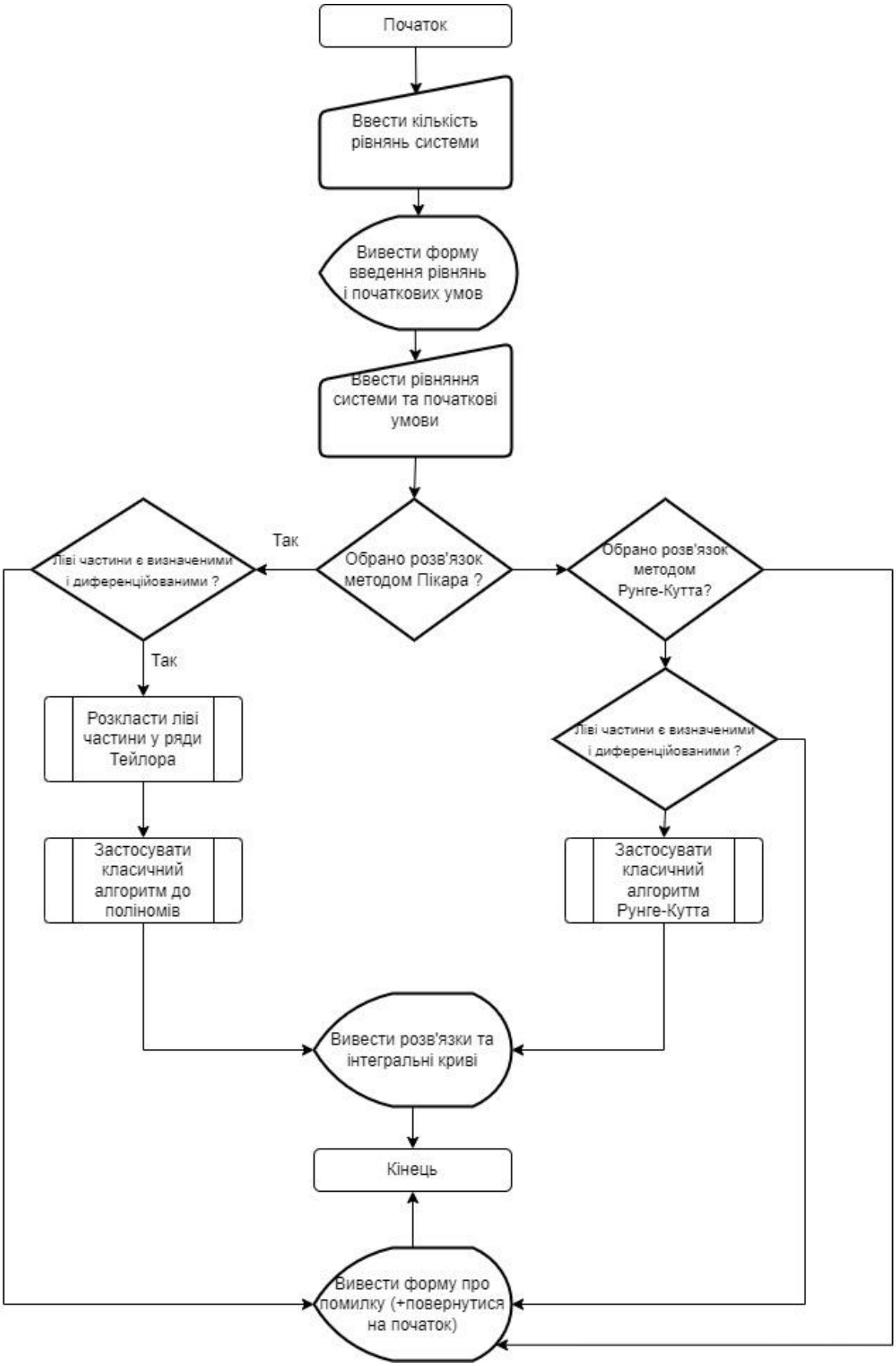


Рисунок 3.1 - Загальний алгоритм функціонування системи

Далі детальніше розглянемо кожну із розроблених підсистем. Почнемо із способу знаходження похідних, що використовується у роботі, після чого перейдемо до розгляду модуля розкладання функцій в ряди Тейлора, адже на нього опирається модифікований алгоритм методу Пікара.

3.1.1 Знаходження похідних

Фундаментально, є два основних підходи до програмного знаходження похідних диференційованих функцій: символьний та чисельний [9].

Символьне диференціювання полягає в знаходженні аналітичної формули для похідної функції. Це може бути корисно, коли потрібно отримати точну формулу для подальшого аналізу, або коли потрібно отримати точні значення похідної для всіх точок в заданому діапазоні. Пакунки для символьного диференціювання, такі як sciру або diff, використовують символьні маніпуляції для обчислення похідної функції.

Однак символьне диференціювання має свої недоліки. Деякі складні функції можуть бути важко або навіть неможливо аналітично диференціювати. Крім того, символьне диференціювання може бути дуже обчислювально-витратним, особливо для складних функцій з багатьма змінними.

З іншого боку, чисельне диференціювання полягає в апроксимації похідної функції, використовуючи числові методи. Це зазвичай швидше, ніж аналітичне диференціювання, і може бути використане для обчислення похідної функції в будь-якій точці з заданою точністю. Наприклад, можна використовувати метод скінченних різниць, який ми і використовуємо у даній роботі, щоб обчислити похідну функції у точці, наблизивши його до лівого та правого сусідів.

Незважаючи на ці переваги, чисельне диференціювання має свої недоліки. Чисельні апроксимації можуть бути неточними, особливо якщо функція має сильні зміни у своїй похідній. Крім того, чисельне диференціювання є операцією погано

визначеною, особливо для функцій, які мають велику похідну або які є неперервними. Це може призвести до помилкових результатів або недооцінки точності. Тому, перед використанням чисельних методів, необхідно ретельно вивчити їх характеристики та обмеження, щоб забезпечити точність і надійність результатів.

У підсумку, обидва методи диференціювання мають свої переваги та недоліки, і їх вибір залежить від потреб користувача та особливостей задачі. Як правило, символьне диференціювання корисне для отримання точних аналітичних формул, тоді як чисельне диференціювання корисне для швидкого і точного обчислення значень похідних у конкретних точках.

Тим не менш, наш вибір зупинився, як було уже сказано, на чисельному диференціюванні і його реалізовано у програмі за допомогою готового модулю scipy.misc.derivative(), який використовує класичну різницеву схему.

На рисунку 3.2 наведемо спрощений алгоритм знаходження похідної.

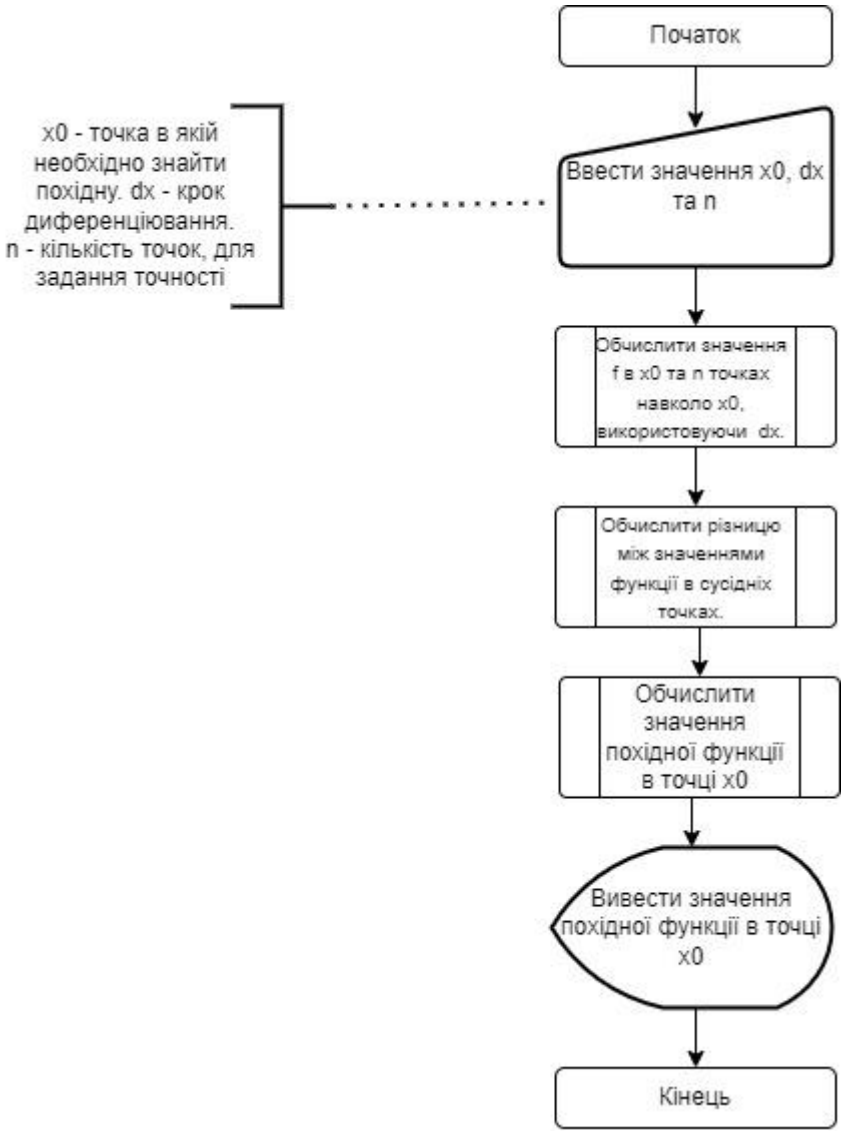


Рисунок 3.2 - Спрощений алгоритм різницевої схеми

Зазначимо ще, що функція scipy.misc.derivative() дозволяє одразу знайти похідну *n*-го порядку, що буде дуже корисно для нас при реалізації алгоритму розкладення у ряд Тейлора.

Варто також відзначити, що безсумнівною перевагою цієї функції є те, що вона дозволяє знайти чисельне значення похідної від, фактично будь-якої диференційованої у заданій точці функції, адже вона опирається на вбудовану функцію eval() мови програмування python, яка дозволяє знайти чисельне значення будь-якого виразу чи будь-якої функції у заданій точці, що є єдиною вимогою до функціонування різницевої схеми.

3.1.2 Розкладення в ряд Тейлора

Модуль розкладу заданої функції у ряд Тейлора (або, вірніше, модуль знаходження перших n членів ряду Тейлора) імплементовано у коді функцією to_taylor_series(), яка отримує на вхід рядок-функцію для розкладу, точку розкладу, порядок розкладу, а також попередні поліноміальні наближення для $y_1 \dots y_m$, які одразу підставляються замість відповідних значень.

Перед використанням цієї функції, тим не менш, потрібно знати, як працює функція factorial(), яка необхідна для знаходження факторіалу натурального числа.

Власне, на рисунку 3.3 наведено блок-схему алгоритму роботи функції factorial(). Вона є допоміжною функцією, яка використовується при обчисленні формули Тейлора.

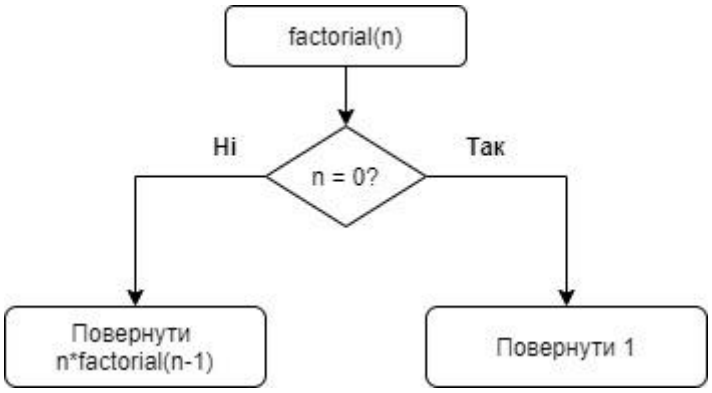


Рисунок 3.3 – Схема роботи алгоритму знаходження факторіалу

Таким чином, це просто класичний рекурсивний алгоритм для знаходження факторіалу заданого натурального числа.

Нарешті, можемо безпосередньо перейти до розгляду модуля, що реалізовує розклад заданої функції (правої частини диференціального рівняння системи) у ряд Тейлора. Ясно, що даний модуль повністю базується на відомій формулі Тейлора (3.1) для розкладу функції $f_i(x)$ у точці x_0 в степеневий ряд:

$$f_i(x) = \sum_{n=0}^{\infty} \frac{f_i^{(n)}(x_0)}{n!}(x - x_0)^n \tag{3.1}$$

Звичайно, неможливо повністю представити нескінчений степеневий ряд на ЕОМ, тому у рамках даної роботи, ми обмежуємося першими k членами цього ряду та відкидаємо його залишковий член. Таким чином:

$$f_i(x) \approx \sum_{n=0}^{k} \frac{f_i^{(n)}(x_0)}{n!}(x - x_0)^n +$$

$$= f_i(x_0) + (x - x_0) \cdot f_i'(x_0) + \cdots + (x - x_0)^k \cdot \frac{f_i^{(k)}(x_0)}{k!} \tag{3.2}$$

Так як тепер ми маємо «на руках» функції обчислення похідних та факторіалів, нескладно імплементувати формулу (3.2) програмно. Власне, наведемо на рисунку 3.4 блок-схему алгоритму функції, що і реалізовує обчислення цієї формули.

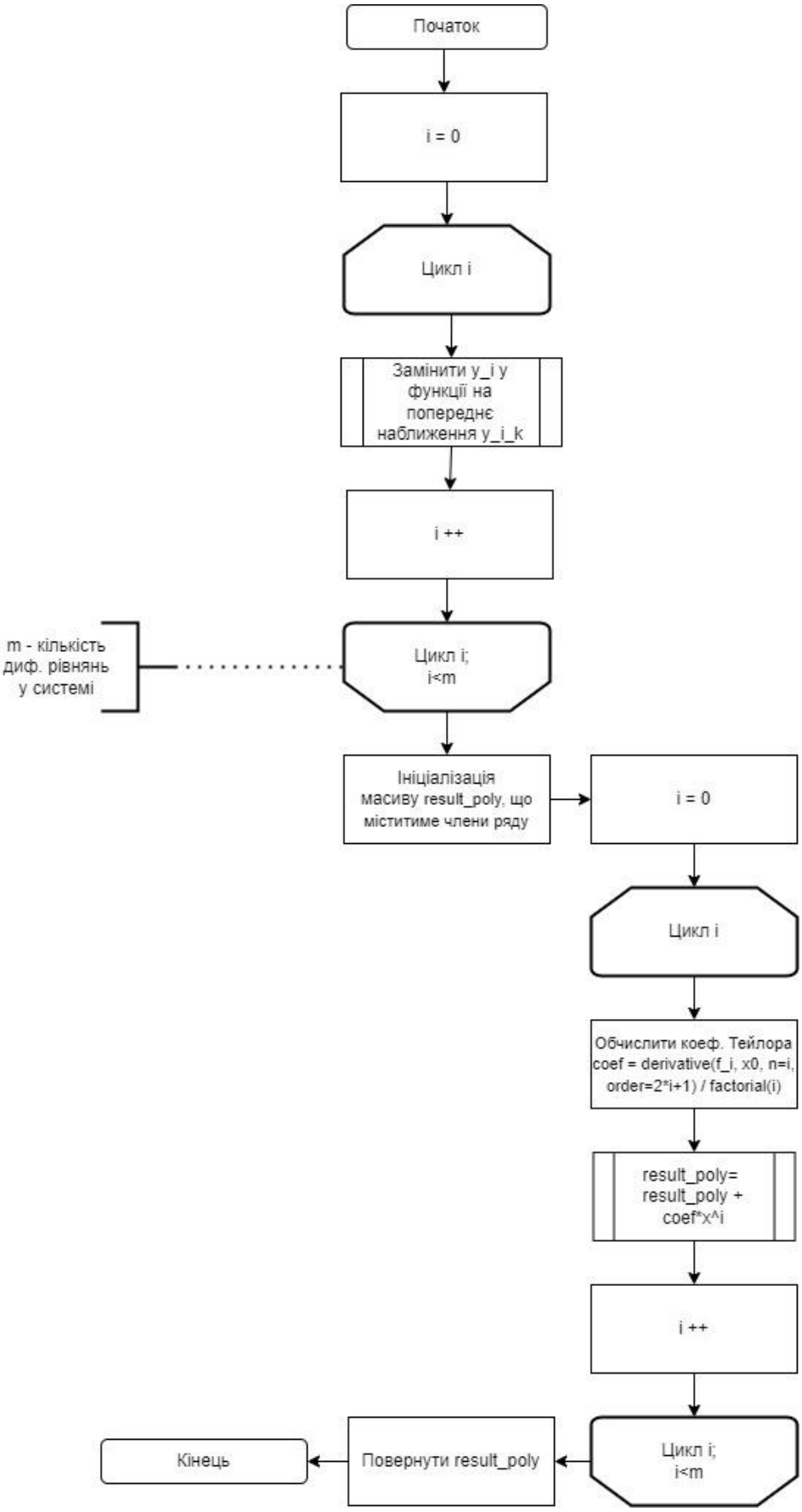


Рисунок 3.4 - Схема алгоритму роботи функції знаходження перших *n* членів ряду Тейлора

3.1.3 Знаходження розв’язку задачі Коші для систем звичайних диференціальних рівнянь першого порядку

Як було показано на блок-схемі алгоритму роботи розробленої системи на самому початку цього розділу, у даному веб-додатку імплементовано два методи розв’язання задачі Коші для систем звичайних диференціальних рівнянь першого порядку, а саме модифікована процедура Пікара-Банаха, а також метод Рунге-Кутта 4-го порядку точності для порівняння результатів

Наведемо ще раз запис ітеративної процедури Пікара, що була отримана у розділі 2:

$$\varphi_i{}^{(j)}(x) = y_{0,i} + \int_{x_0}^{x} f_i\left(\tau, \varphi_1{}^{(j-1)}(\tau), \dots, \varphi_m{}^{(j-1)}\right) d\tau \qquad (3.3)$$

Тут $\varphi_i{}^{(j)}(x)$ – це j-те наближення інтегральної кривої i-го диференціального рівняння системи. У якості $\varphi_i{}^{(0)}(x)$ можна брати будь-яку неперервну функцію на інтервалі I інтегрування. Як уже було обговорено, часто у якості $\varphi_i{}^{(0)}(x)$ доцільно брати константу $y_{0,i}$ з початкової умови ($y_i(x_0) = y_{0,i}$) задачі Коші для системи.

З самого запису (3.3) добре видно і практичне правило реалізації цього алгоритму, яке полягає у інтегруванні функцію $f_i\left(\tau, \varphi_1{}^{(j-1)}(\tau), \dots, \varphi_m{}^{(j-1)}\right)$ на підінтервалі $[x_0, x]$, тобто інтегрування правої частини даного диференціального рівняння, де замість $y_1 \dots y_m$ підставляється останнє наближення розв’язку.

Видно також, що найважчою частиною тут є необхідність інтегрування функції $f_i\left(\tau, \varphi_1{}^{(j-1)}(\tau), \dots, \varphi_m{}^{(j-1)}\right)$. Проте, інтегрування поліномів відносно просто імплементувати програмно і, власне, саме у наближенні підінтегральних функцій поліномами і їх подальшому інтегруванні і полягає модифікація методу Пікара, що розглядається в рамках даної роботи, як описано у розділі 2.

Отже, ми вже маємо модуль to_taylor_series(), що дозволяє наблизити підінтегральну функцію у правій частині (3.3) поліномом. Залишається лише розглянути сам процес інтегрування, який, звичайно, реалізовано окремою функцією. Інтегрування поліномів є досить простою операцією алгоритмічно, тому можемо обійтися без використання чисельних методів, як метод трапецій чи Сімпсона, а одразу імплементувати . До того ж, так як результатом інтегрування полінома є інший поліном, а, власне кажучи, поліноми (з натуральними степенями) можна програмно цілком представити одним масивом його коефіцієнтів, то наш модуль інтегрування оперуватиме не символьними представленнями, а масивами, що значно спрощує написання коду та обчислення, уникає необхідності парсингу, а також економить оперативну пам’ять. Дійсно, наприклад, поліном $1 + 4x + 3x^2$ можемо представити у вигляді масиву $[1, 4, 3]$, а тоді його квадратура запишеться як $x + 2x^2 + x^3$, масив якох відповідно буде $[0, 1, 2, 1]$.

Для ясності наведемо ще блок-схему алгоритму модуля інтегрування на рисунку 3.5.

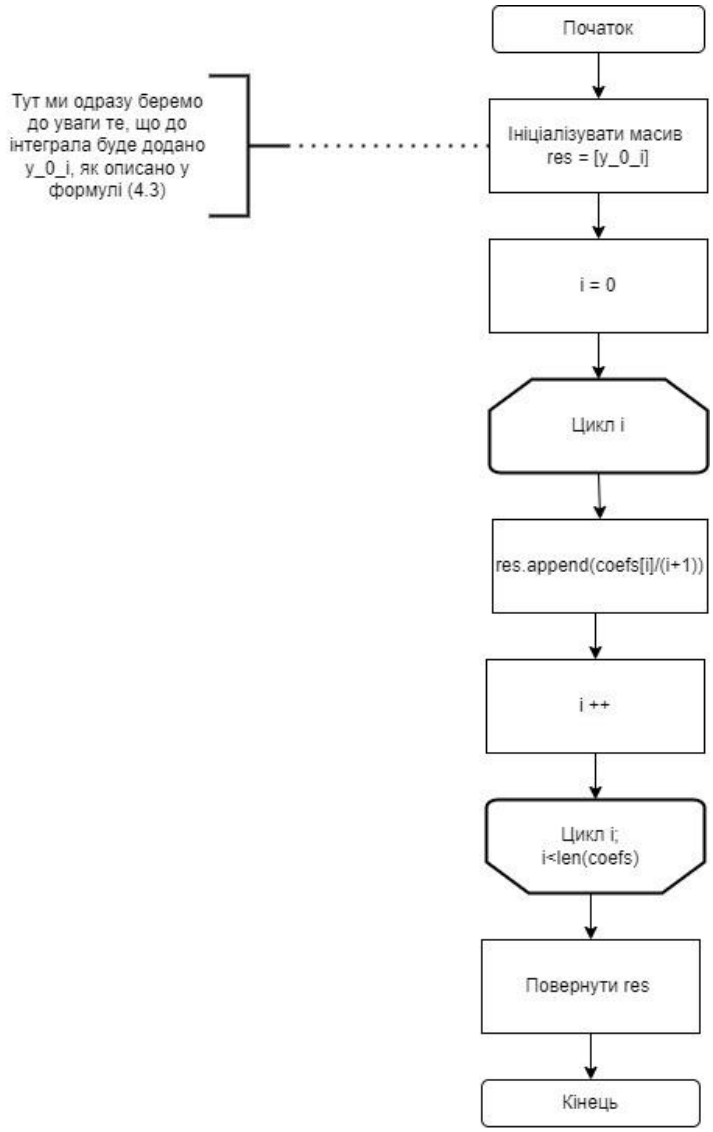


Рисунок 3.5 - Схема алгоритму інтегрування для поліномів

Перейдемо нарешті до розгляду найголовнішої частини розробленої системи, а саме: алгоритму модифікованого методу Пікара. І одразу наведемо його блок-схему на рисунку 3.6.

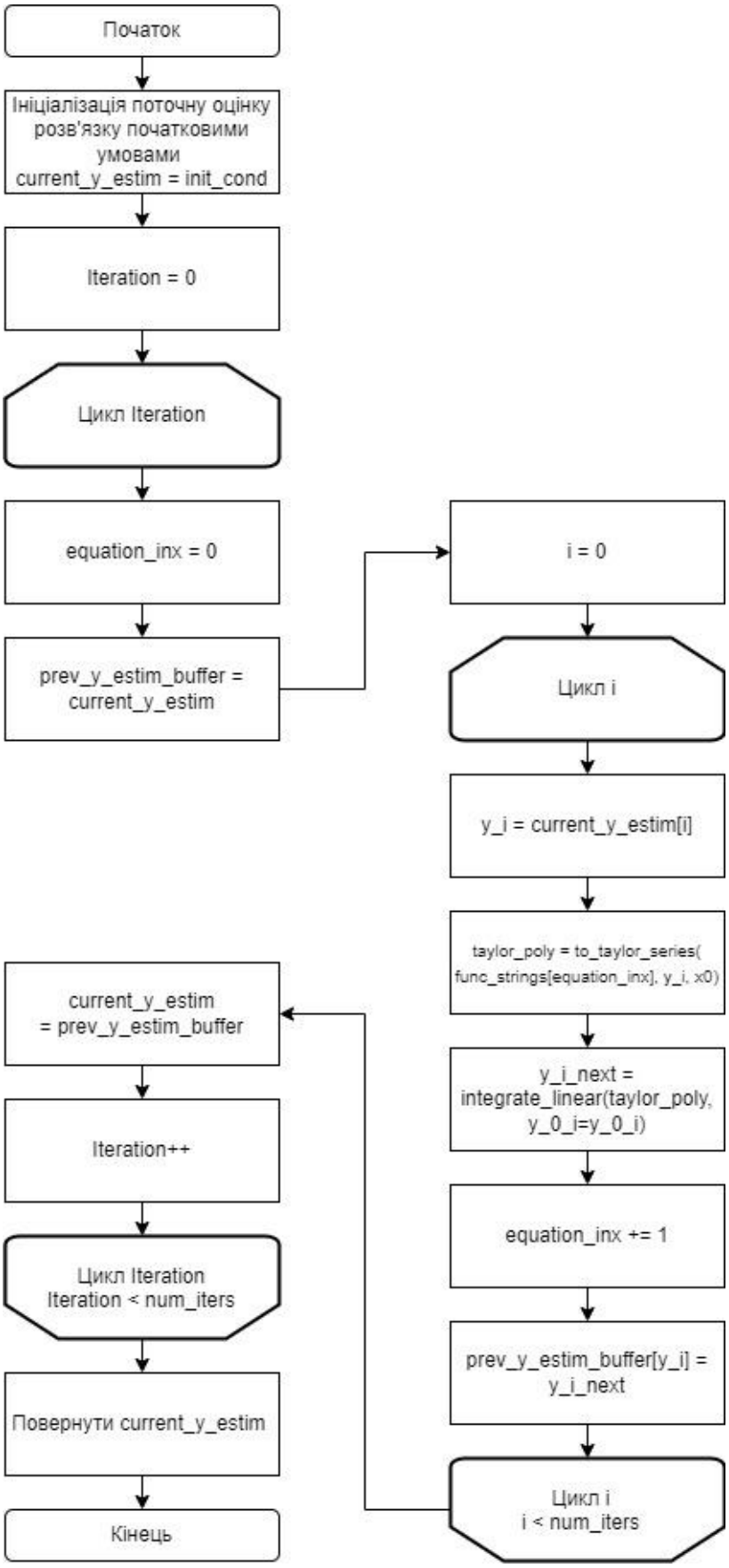


Рисунок 3.6 - Схема алгоритму модифікованого методу Пікара

Нагадаємо далі, що для того, щоб мати змогу інтегрувати систему диференціальних рівнянь на інтервалах більших за початкову область збіжності, ми користуємося спряженням розв'язків. На практиці це зроблено наступним чином. Спочатку, що увесь інтервал інтегрування, який ми позначатимемо $[a, b]$ розбивається на k менших інтервалів, позначимо їх $[a_i, b_i]$. Ми припускаємо, що розбивка є такою, що на цих підінтегралах виконуються умови збіжності модифікованого методу Пікара.

Після цього, власне за модифікованим методом Пікара, знаходиться наближення розв'язку системи диференціальних рівнянь на підінтервалі, що містить точку x_0 із початкової умови. Нехай це буде інтервал $[a_m, b_m]$, для визначеності. Також позначимо розв'язок, що ми отримаємо на цьому підінтервалі, як $\boldsymbol{y}_{[a_m,b_m]}(x)$.

Для отримання наступного наближення, звичайно, на кожному кроці методу, функції $f_i(x,\ {\varphi_1}^{n-1}(x), \dots, {\varphi_m}^{n-1}(x))$ розкладаються у відповідні ряди Тейлора (вірніше, знову, у перші декілька членів їх рядів Тейлора) в точці x_0, щоб отримати наближення цих функцій поліномом, яке можна надалі буде проінтегрувати функцією інтегрування поліномів, описаною вище. Ясно, що таке наближення інтегральних кривих буде відносно неточним, бо до похибки методу Пікара ще додається похибка розкладу функцій у ряди Тейлора (що відповідає відкинутому залишковому члену, детальніше див. доведення у розділі 2). Але, звичайно, чим більшу кількість підінтервалів ми візьмемо, тим менш суттєва буде ця похибка, адже тим меншої довжини буде кожен із інтервалів $[a_i, b_i]$. У розробленій системі, кількість підінтервалів, а також степінь до якої розкладаються функції у ряд Тейлора можуть бути задані користувачем. Інакше приймаються значення обчислені автоматично.

Все те ж саме повторюємо далі для інтервалу справа: $[a_{m+1}, b_{m+1}]$, якщо, звісно, такий є (тобто, якщо інтервал $[a_m, b_m]$ не був останнім справа). Але змінюємо початкову умову: замість $\boldsymbol{y}(x_0) = \boldsymbol{y_0}$ матимемо:

$$x_0 = b_m = a_{m+1}; \tag{3.4}$$

$$\boldsymbol{y_0} = \boldsymbol{y}_{[a_m,b_m]}(b_m); \tag{3.5}$$

І беручи до уваги те, що на інтервалі $[a_m, b_m]$, маємо $\boldsymbol{y}_{[a_m,b_m]} \approx \boldsymbol{y}(x)$, то роблячи припущення, що цей розв'язок належить області існування та єдиності задачі Коші та те, що він є асимптотично стійким, отримаємо:

$$\boldsymbol{y}_{[a_m,b_m]}(b_m) \approx \boldsymbol{y}(b_m) \tag{3.6}$$

Тому ми і беремо його у якості початкової умови.

І так далі аж доки не дійдемо до останнього інтервалу, а саме: $[a_{m+k}, b_{m+k}] = [a_{m+k}, b]$. Далі повторюємо ту ж процедуру, але рухаючись цього разу від інтервалу $[a_m, b_m]$ наліво і у якості початкової умови, взявши такі умови:

$$a_m = b_{m-1} = x_0; \tag{3.7}$$

$$\boldsymbol{y}_{[a_m,b_m]}(a_m) = \boldsymbol{y_0}; \tag{3.8}$$

Знову обмовимося, що такий алгоритм є обґрунтованим лише у області збіжності цього методу, тобто у області існування і єдиності розв'язку даної задачі Коші, бо за її межами, немає жодних гарантій існування інтегральних кривих.

У програмі область збіжності визначається згідно з формулами, які були виведені в теоретичній частині. В окремих формулах для знаходження максимуму певної функції використовується оптимізаційний метод спряжених напрямків Пауелла, що був імплементований у бібліотеці scipy для мови Python. Вибір саме цього методу був зроблений емпірично, оскільки для функцій, які тестувалися у даній роботі, він показував відносно швидкі та надійні результати, тому що рідко залишався у локальних мінімумах. Тим не менш, у розробленій системі процес оптимізації функцій зазвичай займає набагато менше часу, ніж процес знаходження інтегральних кривих, тому для цього можна було б обрати будь-який інший сучасний метод, наприклад BFGS.

Насамкінець додамо, що у програмному коді було імплементовано також інші допоміжні функції, як-от наприклад, функція додавання двох поліномів або

піднесення їх до степеня. Їх програмна реалізація представляє собою технічні деталі, які недоцільно включати у цей розділ, тим не менш, читач, якого цікавить їх програмна імплементація, може, звичайно, проконсультувати лістинги або репозиторій, наданий на самому початку даного розділу. Проте ми ще частково зачепимо ці функції у наступному розділі, де знаходження їх алгоритмічної складності буде важливим для нас у рамках визначення глобальної алгоритмічної складності модифікованого методу Пікара.

3.2 Оцінки алгоритмічної складності модулів

Розглянувши більш детально програмний код, констатуємо, що алгоритмічна складність коду для знаходження розв'язку системи диференціальних рівнянь модифікованим методом Пікара залежить від кількості ітерацій, тобто значення параметру num_iters, та від кількості членів у поліномах, тобто значення параметру taylor_order.

Оцінимо алгоритмічну складність функції integrate_linear - ця функція складається з циклу з len(coefs) ітерацій, тобто має складність O(n), де n = len(coefs).

Функція list_to_formula складається з циклу з len(coefs) ітерацій, тому її складність також O(n), де n = len(coefs).

Алгоритм множення поліномів multiply_polys має вкладений цикл, тому його складність становить O(n^2), де n - довжина списку poly_list1.

Функція poly_to_power множить поліном на самого себе power разів, тому її складність - O(n^2 * k), де n - довжина списку poly_list, а k - значення параметру power.

Функція add_arr має складність O(n), де n - довжина списку a або b, тому що вона містить лише один цикл.

Алгоритм обчислення факторіала factorial також має складність O(n), де n - значення параметру n.

Алгоритм обчислення значення ряду Тейлора у функції to_taylor_series має складність O(n^2), де n - значення параметру n. Використовувані тут функції custom_func та derivative мають складність, яка залежить від точності обчислення похідних.

Отже, загалом алгоритмічна складність даного коду складається з декількох компонент:

Обчислення поліномів - у функції multiply_polys і poly_to_power використовується поділ і множення масивів, що має алгоритмічну складність O(n^2), де n - довжина масиву.

Обчислення інтегралів - у функції integrate_linear обчислюється сума n елементів, де n - ступінь полінома, що має алгоритмічну складність O(n).

Обчислення похідних - у функції to_taylor_series використовується функція derivative з бібліотеки scipy.misc, яка виконує обчислення чисельного значення похідної функції, що має алгоритмічну складність O(n^2), де n - порядок похідної.

Обчислення значень функцій - у функції custom_func використовується функція eval, яка виконує обчислення значення функції, що має алгоритмічну складність O(n), де n - довжина рядка.

В решті решт, теоретично, складність функції picard_general може бути визначена як O(n^2) або O(n^3), де n - кількість ітерацій, що необхідні для досягнення потрібної точності. Однак, на практиці, складність може бути нижчою, якщо метод збігається швидше, або вищою, якщо потрібна більш висока точність.

3.3 Тестування програмного забезпечення

У цьому розділі ми наведемо декілька наборів тестових випадків, які покликані якнайкращим чином покрити увесь код програми, аби забезпечити її коректну та безперебійну роботу.

Тестові випадки наведені у наступних підрозділах. У кожному з, них таблиці спочатку містять тести для коректно введених користувачем даних, а в кінці, відповідно, некоректно введених.

3.3.1 Метод Пікара

Таблиця 3.1 – тестовий випадок М.П. 1

Номер тестового випадку	М.П. 1
Опис	Розв’язати задачу Коші для системи з 2 рівнянь методом Пікара, із заданими початковими умовами
Передумови тесту	Веб-застосунок відкрито
Кроки виконання	Очікуваний результат
Увести кількість рівнянь системи і саму систему у відповідні поля. Наприклад, «2» та y1' = x^3 + y2, y2’=y1 , x0 = 0, y01 = 1, y02 = 1»	Запит відображається у полі
Натиснути кнопку « Розв’язати систему »	Завантажується сторінка із розв’язком і з аналізом точності

Таблиця 3.2 – тестовий випадок М.П. 2

Номер тестового випадку	М.П. 2
Опис	Розв’язати задачу Коші для одного рівняння
Передумови тесту	Веб-застосунок відкрито
Кроки виконання	Очікуваний результат
Увести в головне поле коректний запит, але вказати лише 1 рівняння. Наприклад, «y1' = x^3 + y1 + ½; x0 = 1; y0 = 1»	Запит відображено у відповідному полі
Натиснути кнопку « Розв’язати систему »	Завантажується сторінка із розв’язком і з аналізом точності

Таблиця 3.3 – тестовий випадок М.П. 3

Номер тестового випадку	М.П. 3
Опис	Розв’язати систему з десяти рівнянь методом Пікара
Передумови тесту	Веб-застосунок відкрито
Кроки виконання	Очікуваний результат
В головне поле застосунку, ввести запит, що містить 10 різних рівнянь	Запит відображається у полі
Натиснути кнопку « Розв’язати систему »	Завантажується сторінка із розв’язком і з аналізом точності

Таблиця 3.4 – тестовий випадок М.П. 4

Номер тестового випадку	М.П. 4
Опис	Розв’язати систему, що містить експоненту
Передумови тесту	Веб-застосунок відкрито

Продовження таблиці 3.4

Кроки виконання	Очікуваний результат
Ввести у головне поле такий запит, який містить експоненту. Наприклад, «y1' = e^-x + y1 + 1/2»	Запит відображається у полі
Натиснути кнопку « Розв’язати систему »	Завантажується сторінка із розв’язком і з аналізом точності

Таблиця 3.5 – тестовий випадок М.П. 5

Номер тестового випадку	М.П. 5
Опис	Розв’язати систему, що містить тригонометричні функції
Передумови тесту	Веб-застосунок відкрито
Кроки виконання	Очікуваний результат
В головне поле застосунку ввести запит, що містить тригонометричні функції, наприклад «y1' = sin(-x) + y2 + ½; y2’ = y1»	Запит відображається у полі
Натиснути кнопку « Розв’язати систему »	Завантажується сторінка із розв’язком і з аналізом точності

Таблиця 3.6 – тестовий випадок М.П. 6

Номер тестового випадку	М.П. 6
Опис	Розв’язати систему, що містить логарифмічну функцію
Передумови тесту	Веб-застосунок відкрито
Кроки виконання	Очікуваний результат

Продовження таблиці 3.6

Кроки виконання	Очікуваний результат
Увести в головне поле запит, що містить логарифмічну функцію, наприклад «y1' = log(x) + y1; y2’ = y1»	Запит відображається у полі
Натиснути кнопку « Розв’язати систему »	Завантажується сторінка із розв’язком і з аналізом точності, наведено наближений графік інтегральної кривої та вказано область збіжності

Таблиця 3.7 – тестовий випадок М.П. 7

Номер тестового випадку	М.П. 7
Опис	Розв’язати систему, що складається виключно з нелінійних диф. рівняння методом Пікара
Передумови тесту	Веб-застосунок відкрито
Кроки виконання	Очікуваний результат
Увести в відповідні поля нелінійні диф. рівняння.	Запит відображається у відповідному полі для кожного із рівнянь системи
Натиснути кнопку « Розв’язати систему »	Завантажується сторінка з розв’язком, наведено графіки наближених інтегральних кривих.

Таблиця 3.8 – тестовий випадок М.П. 8

Номер тестового випадку	М.П. 8
Опис	Ввести некоректні дані
Передумови тесту	Веб-застосунок відкрито
Кроки виконання	Очікуваний результат
Увести набір символів, замість кількості рівнянь. Наприклад, «ее». Натиснути кнопку «Розв’язати систему »	Отримано повідомлення про некоректність вводу
Замість диф. рівняння, ввести набір символів. Наприклад, «y1' =r231@». Натиснути кнопку « Розв’язати систему»	Отримано повідомлення про некоректність вводу
Увести набір символів, замість початкової умови. Наприклад, «еее». Натиснути кнопку « Розв’язати систему»	Отримано повідомлення про некоректність вводу
Замість початкової умови, ввести набір символів,. Наприклад, «y01 = абв». Натиснути кнопку « Розв’язати систему »	Отримано повідомлення про некоректність вводу
Увести набір символів, замість кількості ітерацій. Наприклад, «еее». Натиснути кнопку « Розв’язати систему»	Отримано повідомлення про некоректність вводу
Увести набір символів, замість кількості рівнянь, після того, як система була введена. Наприклад, «еее». Натиснути кнопку « Розв’язати систему»	Отримано повідомлення про некоректність вводу

Таблиця 3.9 – тестовий випадок М.П. 9

Номер тестового випадку	М.П. 9
Опис	Ввести систему диф. рівнянь, для якої не виконується умова Ліпшиця
Передумови тесту	Веб-застосунок відкрито
Кроки виконання	Очікуваний результат
В головне поле застосунку, ввести диференціальне рівняння для якого умова Ліпшиця не виконується. Наприклад рівняння, *«y1' =1/x»*. Натиснути кнопку « Розв’язати систему»	Отримано повідомлення про розбіжність методу

3.4 Висновки до розділу

Цей розділ було присвяченено програмній імплементації, описаної у розділі 2, модифікації методу Пікара послідовних наближень. Тут було, зокрема, описано алгоритми розроблених модулів, а також наведено їх блок-схеми, описано також основні функціональні можливості розробленого додатку: розв’язання задачі Коші для систем диференціальних рівнянь методами Пікара та Рунге-Кутта. Для методу Пікара, згідно з теоретичними результатами, отриманими у розділі 2, було реалізовано аналіз збіжності та обчислення похибки. До того ж, значну увагу було приділено до розробки тестових випадків для побудованої системи, що покликані забезпечити якомога стабільнішу роботу розробленої системи.

4 ЕКСПЕРИМЕНТАЛЬНІ РОЗРАХУНКИ

У цьому фінальному розділі ми продемонструємо на конкретних прикладі роботу побудованої системи, а також виконаємо тести, наведені у попередньому розділі, щоб безпосередньо переконатися у коректності роботи розробленої програми. Знову нагадаємо, що доступ до веб-застосунку можна отримати за таким посиланням: http://picardsolver.pythonanywhere.com/

4.1 Демонстрація роботи програми

Для демонстрації коректності роботи імплементованого алгоритму Пікара, а також щоб показати його збіжність, знайдемо за його допомогою розв'язок простої лінійної системи диференціальних рівнянь (тобто розв'яжемо задачу Коші для цієї системи):

$$\begin{cases} y_1' = y_2 + y_3 - y_1 \\ y_2' = -y_1 - y_2 \\ y_3' = -y_1 - 3y_3 \end{cases} \tag{4.1}$$

$$\text{За умов}: \; y_1(0) = 0,\; y_2(0) = 1, y_3(0) = -1 \tag{4.2}$$

Аналітичний розв'язок цієї системи можна досить легко знайти вручну і її частинний розв'язок, що задовольняє умови (4.2) – це:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = e^{-x} \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix} \tag{4.3}$$

Введемо цю систему у веб-застосунок і подивимося на розв’язок, знайдений за методом Пікара. Результати наведено на рисунку 4.1.

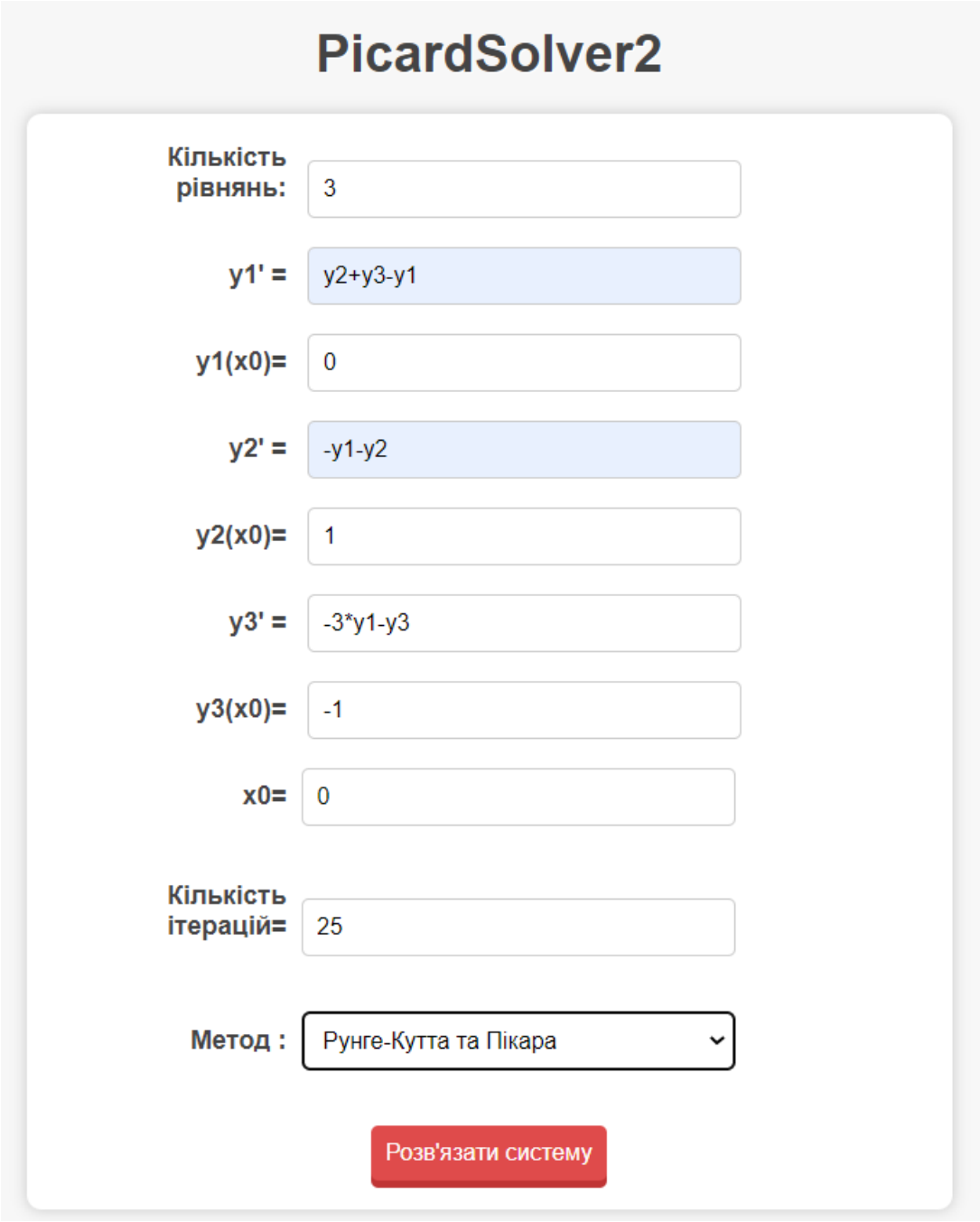


Рисунок 4.1 – Введені дані

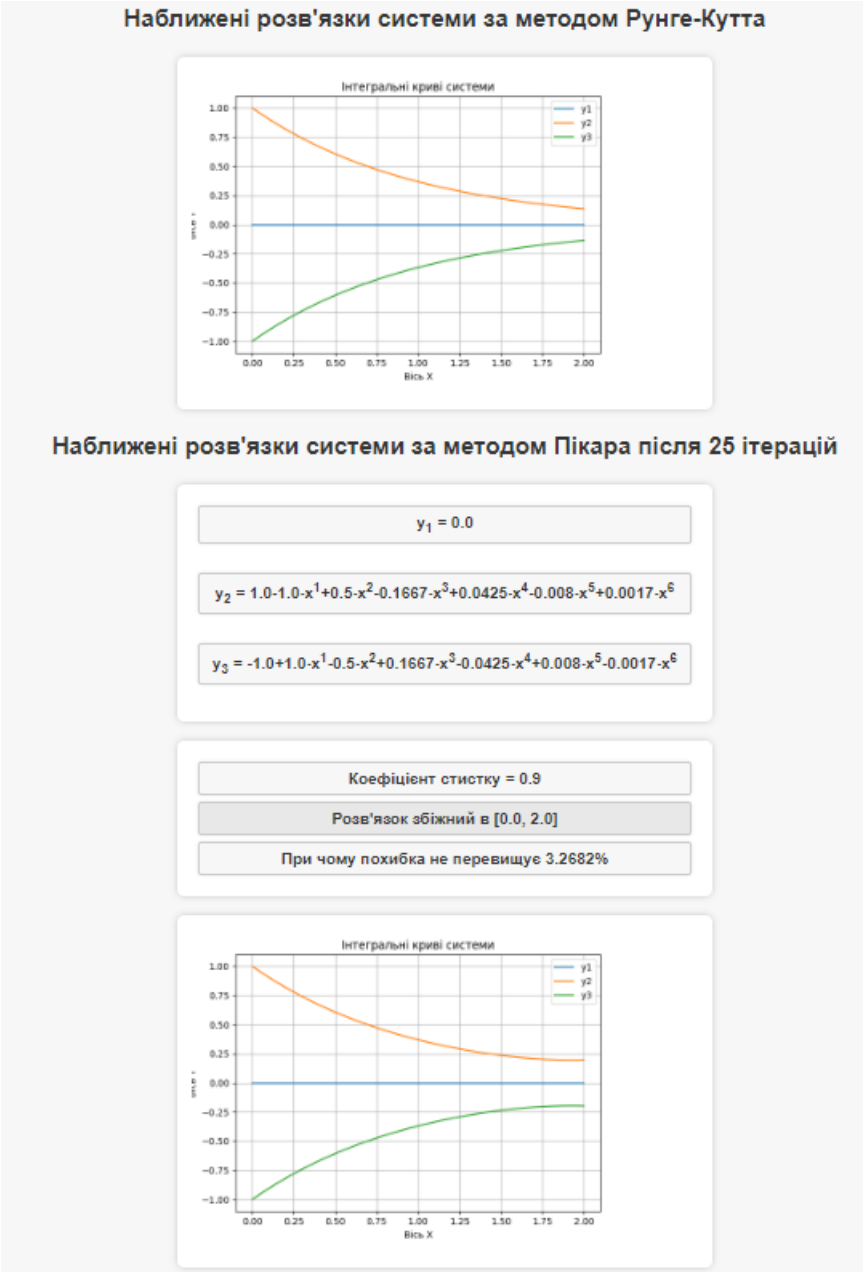


Рисунок 4.2 – Знайдений розв’язок

Нескладно переконатися у правильності отриманого розв’язку. Дійсно, адже отриманий результат для y_2 та y_3 є нічим іншим, як записом перших декількох членів ряду Тейлора (4.4) для e^{-x} та $-e^{-x}$, відповідно, що є правильним аналітичним розв’язком системи.

$$e^{\tau} = \sum_{k=1}^{\infty} \frac{\tau^n}{n!} = 1 + \tau + \frac{1}{2}\tau^2 + \frac{1}{6}\tau^3 + \frac{1}{24}\tau^4 + + \frac{1}{120}\tau^5 + \frac{1}{720}\tau^6 + \cdots =$$

$$= 1 + 1 \cdot \tau + 0.5\tau^2 + 0.166(6)\tau^3 + 0.04166(6)\tau^4 + 0.00833(3)\tau^5 + 0.001388(8) \ldots \quad (4.4)$$

І це дійсно якраз те, що ми і отримали у відповідному полі.

Коректність отриманого результату також підтверджує той факт, що отримані розв'язки візуально збіглися з розв'язками, отриманими за допомогою метода Рунге-Кутта 4-го порядку.

Це також наглядно демонструє той факт, що метод Пікара має лінійну збіжність, адже так як рад Тейлора має нескінченну кількість членів, то і для того, щоб отримати точний розв'язок за методом Пікара потрібно було б зробити нескінченну кількість ітерацій (тим не менш, часто, доволі непогану точність можна досягти за невелику кількість кроків).

Далі, покажемо, як працюватиме розроблений застосунок при введені в неї нелінійної системи. Візьмемо, наприклад, таку систему :

$$\begin{cases} y_1' = y_2 + y_1 \\ y_2' = x \cdot y_1 \end{cases} \quad (4.5)$$

$$\text{За умов}: \; y_1(0) = 1, \; y_2(0) = 0 \quad (4.6)$$

Тоді вводимо ці дані у систему:

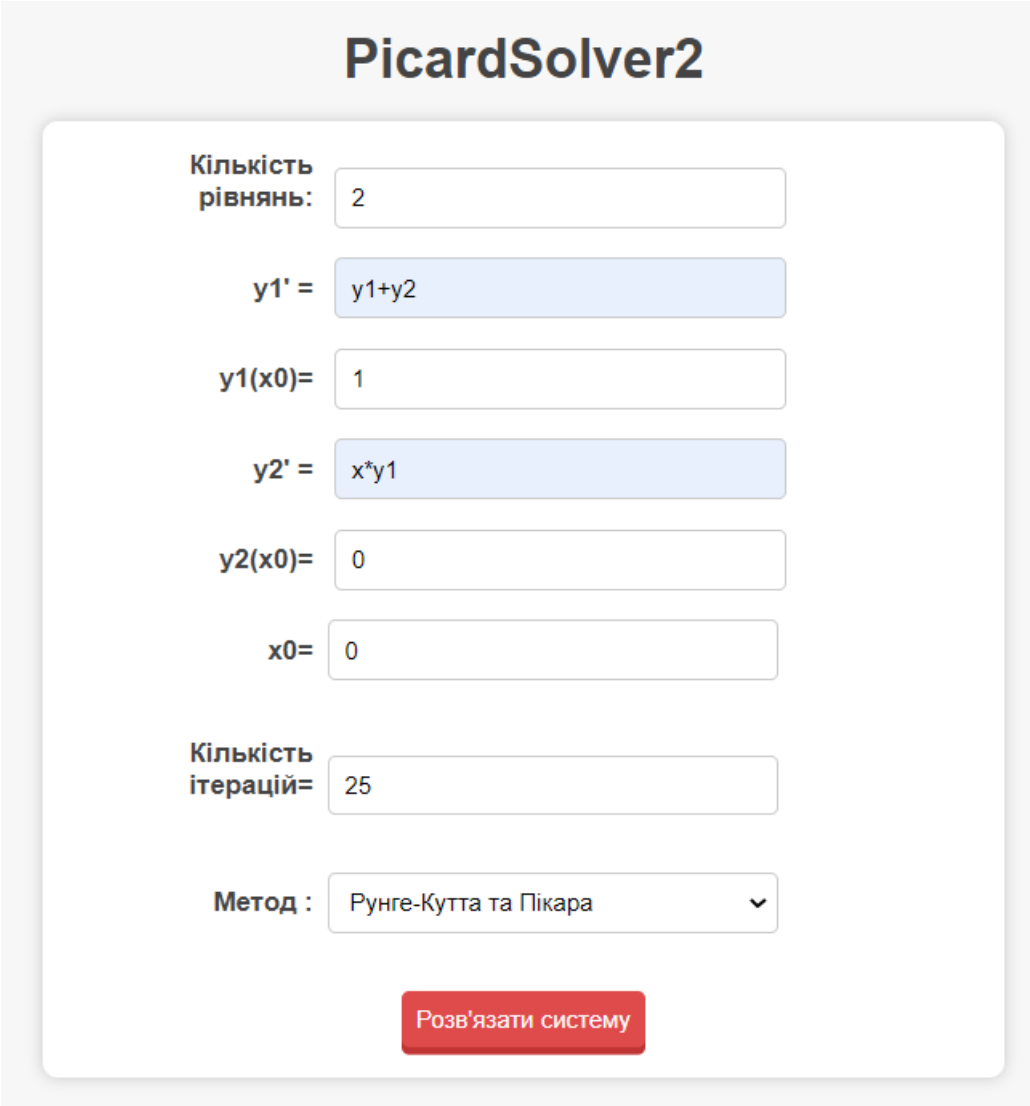


Рисунок 4.3 – Введені дані

Рисунок 4.4 – Знайдений розв’язок

Легко перевірити власноруч, що отриманий розв’язок дійсно збігається з правильним частинним розв’язком. Це також, знову ж таки, підтверджується тим, що отримані за методом Пікара інтегральні криві збігаються з кривими, отриманими за методом Рунге-Кутта.

Далі, продемонструємо роботу системи на більш складному прикладі нелінійної системи і візьмемо для цього систему рівнянь Лотке-Вольтерра, які широко використовуються для моделювання природніх і соціологічних процесів типу хижак-жертва.

Тоді введемо одну з таких систем у відповідні поля веб-додатку.

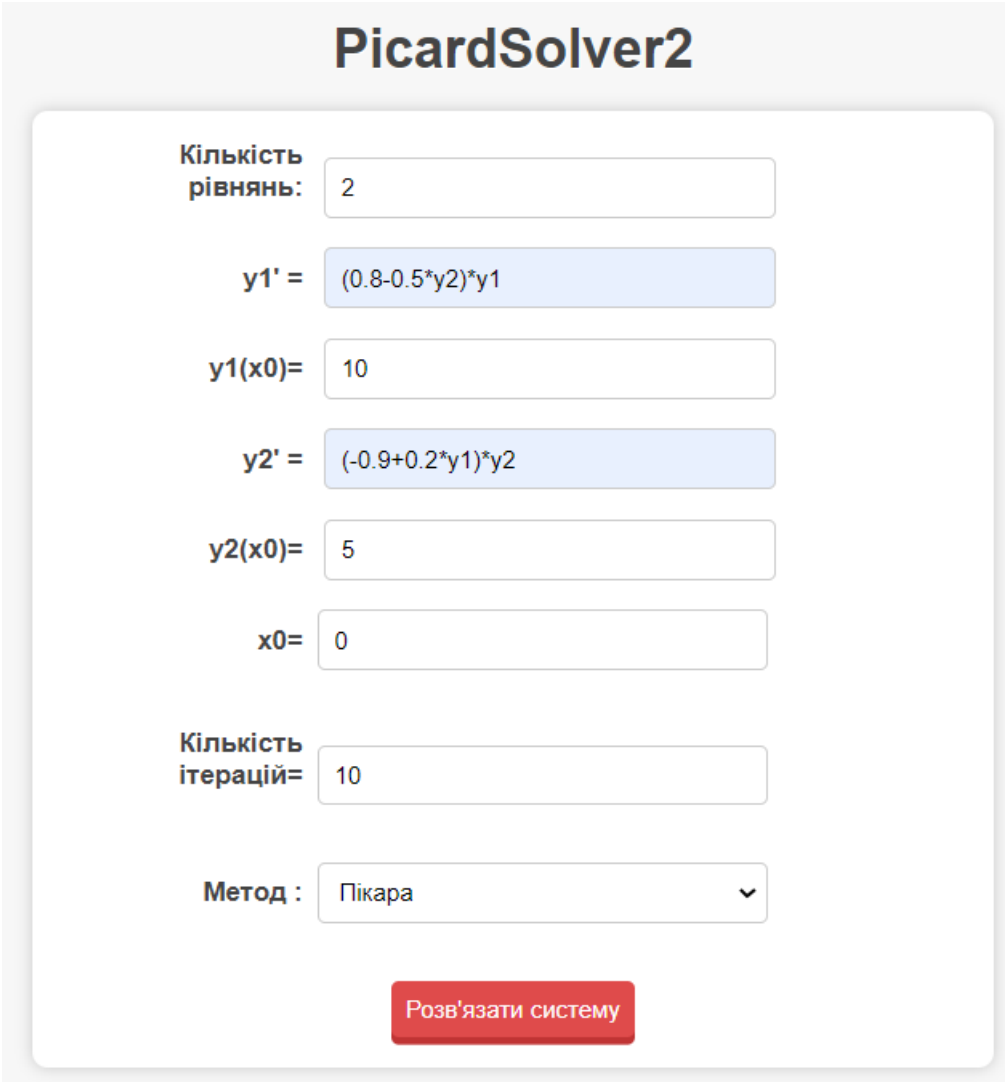


Рисунок 4.5 – Введені дані

І одразу розглянемо отримані інтегральні криві.

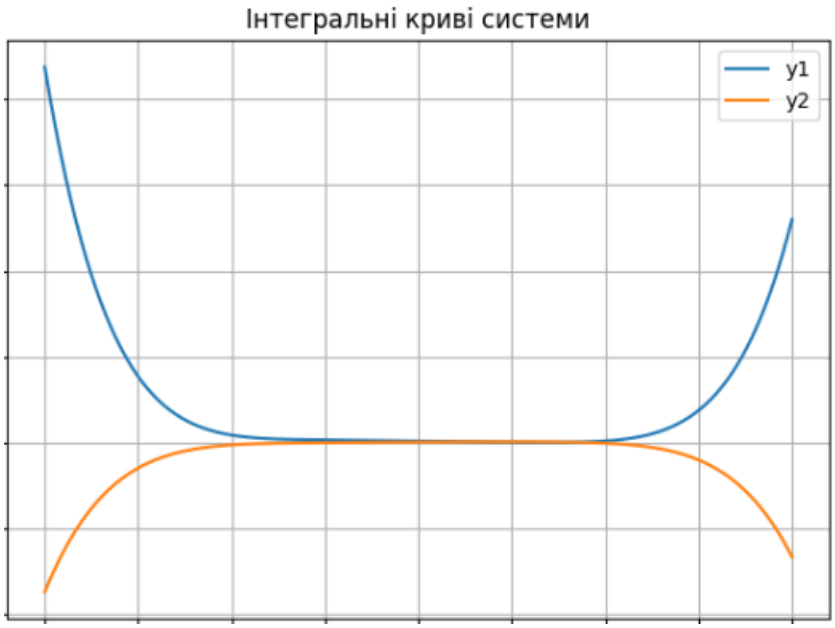


Рисунок 4.6 – Знайдений розв’язок

Ми бачимо на отриманих графіках характерну поведінку для розв’язку систем Лотки-Вольтера, а саме: інтерпретуючи y1, як «жертв», а y2, як «хижаків» ми бачимо, що спочатку кількість хижаків поступово зростає, а кількість жертв – спадає, адже для кожного з хижаків є достатня кількість їжі. Далі, обидві популяції стабілізуються біля якоїсь точки еквілібріуму, де вони знаходяться певний час, після чого розв’язки знову розходяться, адже жертв стає недостатньо для усіх хижаків і вони починають вимирати, даючи можливість популяції жертв збільшити своє число.

Насамкінець, покажемо на прикладі, що система також розрахована на нелінійні системи, що містять не тільки поліноміальні функції, але й, наприклад, тригонометричні функції.

Рисунок 4.7 – Введені дані

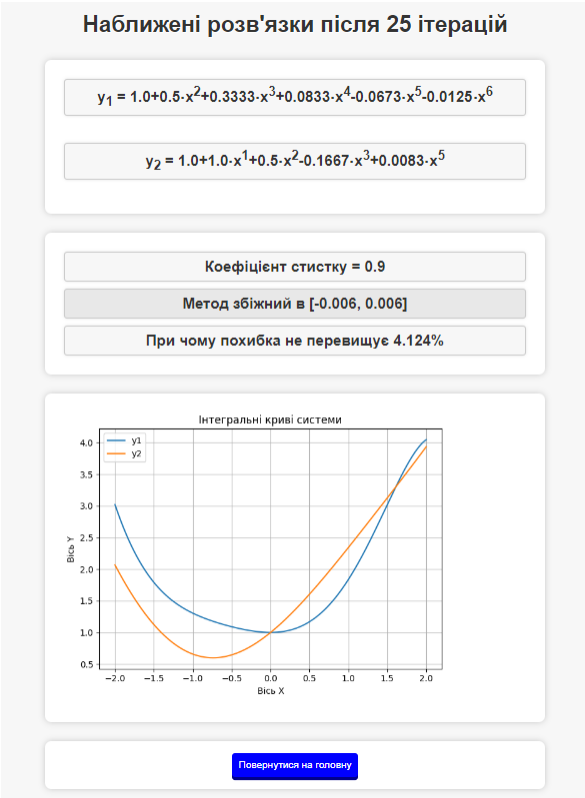


Рисунок 4.8 – Знайдений розв’язок

4.2 Перевірка тестових випадків

Перейдемо тепер до перевірки тестових випадків, наведених у відповідних підрозділах. Будемо їх виконувати тестових в тому порядку, в якому вони наведені. Знімки екрану із тестовими випадками наведені на наступних рисунках.

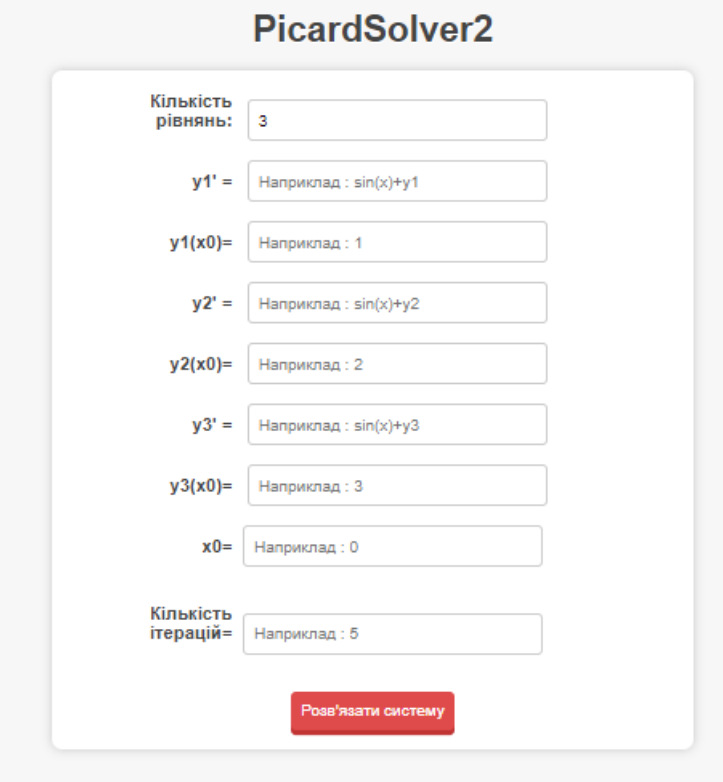


Рисунок 4.9 - Головна сторінка розробленого застосунку. Запити вводяться у відповідні поля у центрі

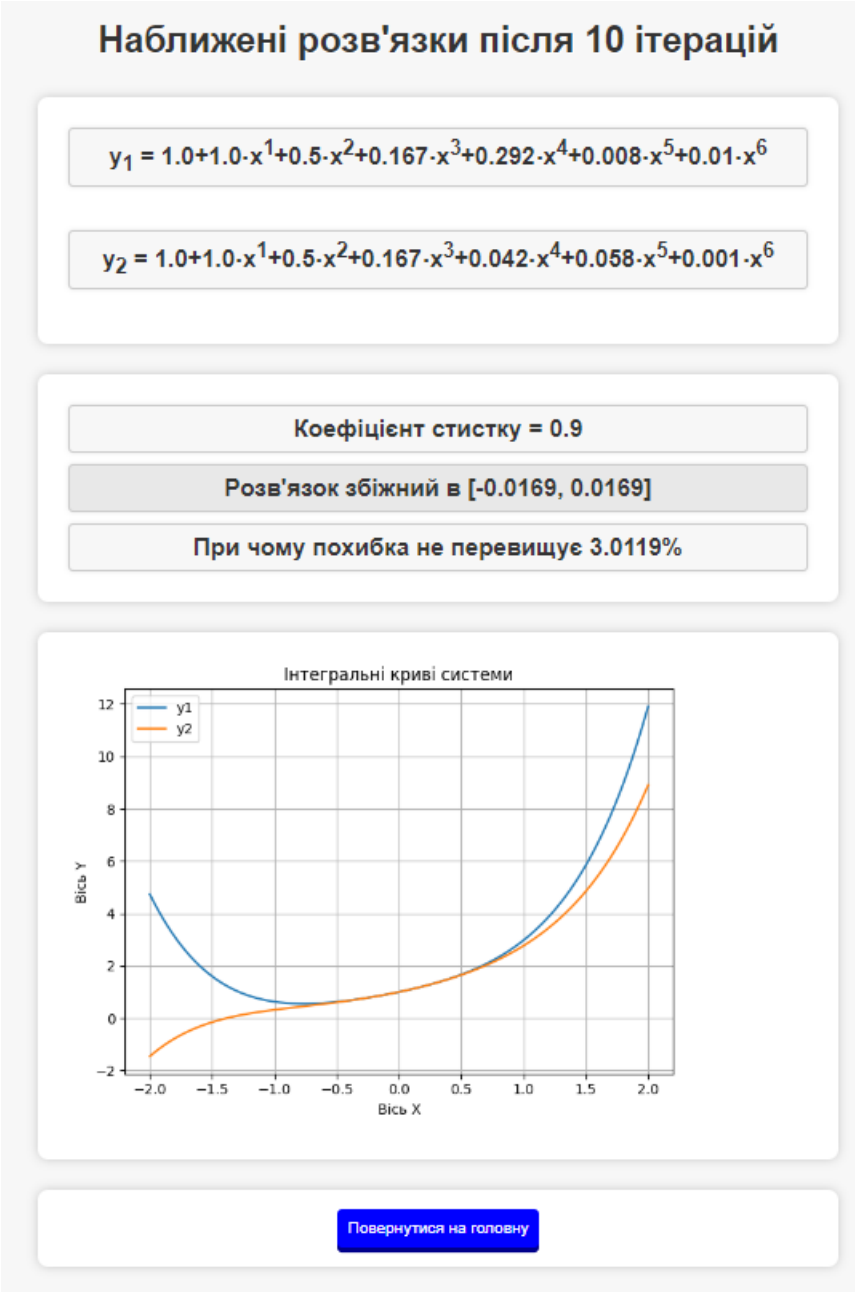


Рисунок 4.10 – Тест М.П.1, відображення розв’язку

Рисунок 4.11 – Тест М.П.2, відображення розв’язку

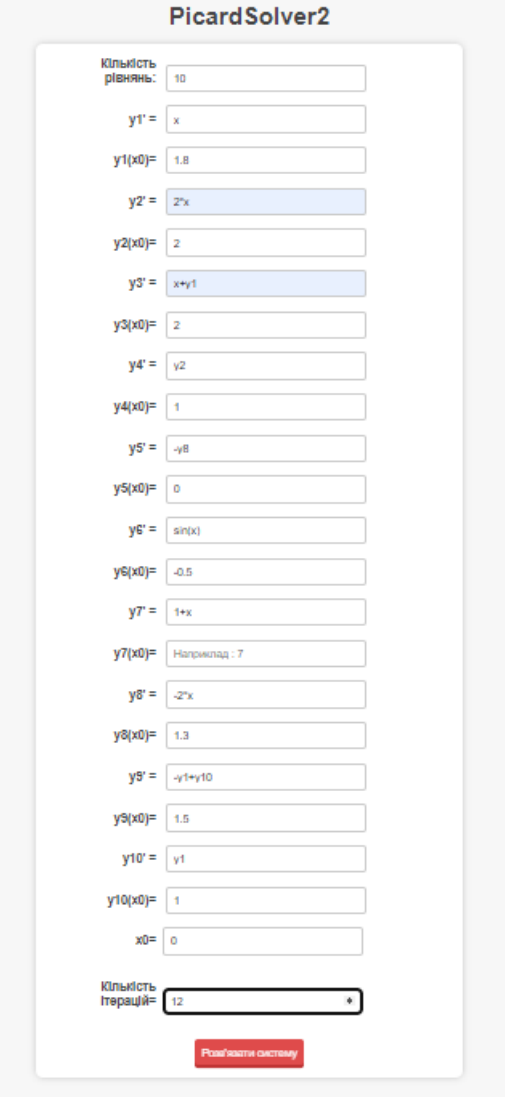


Рисунок 4.12 – Тест М.П.3, відображення запиту

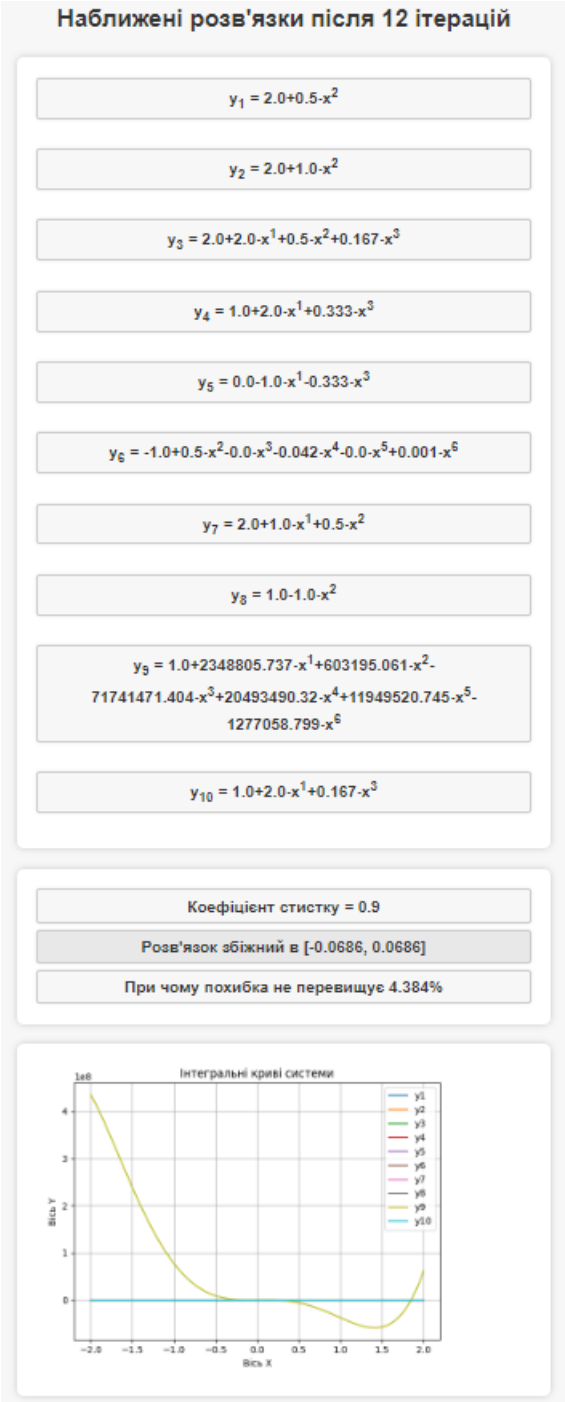


Рисунок 4.13 – Тест М.П.3, відображення розв’язку

Рисунок 4.14 – Тест М.П.4, відображення розв’язку

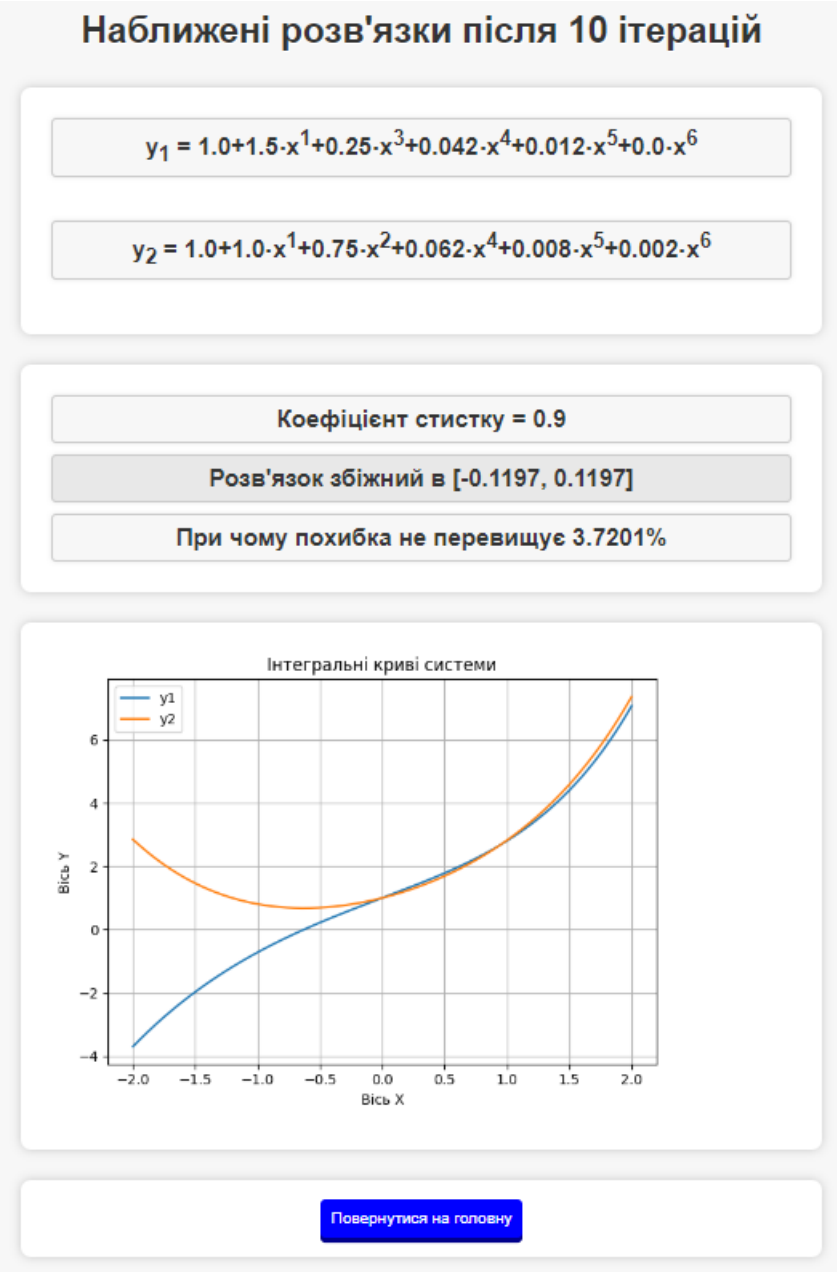


Рисунок 4.15 – Тест М.П.5, відображення розв’язку

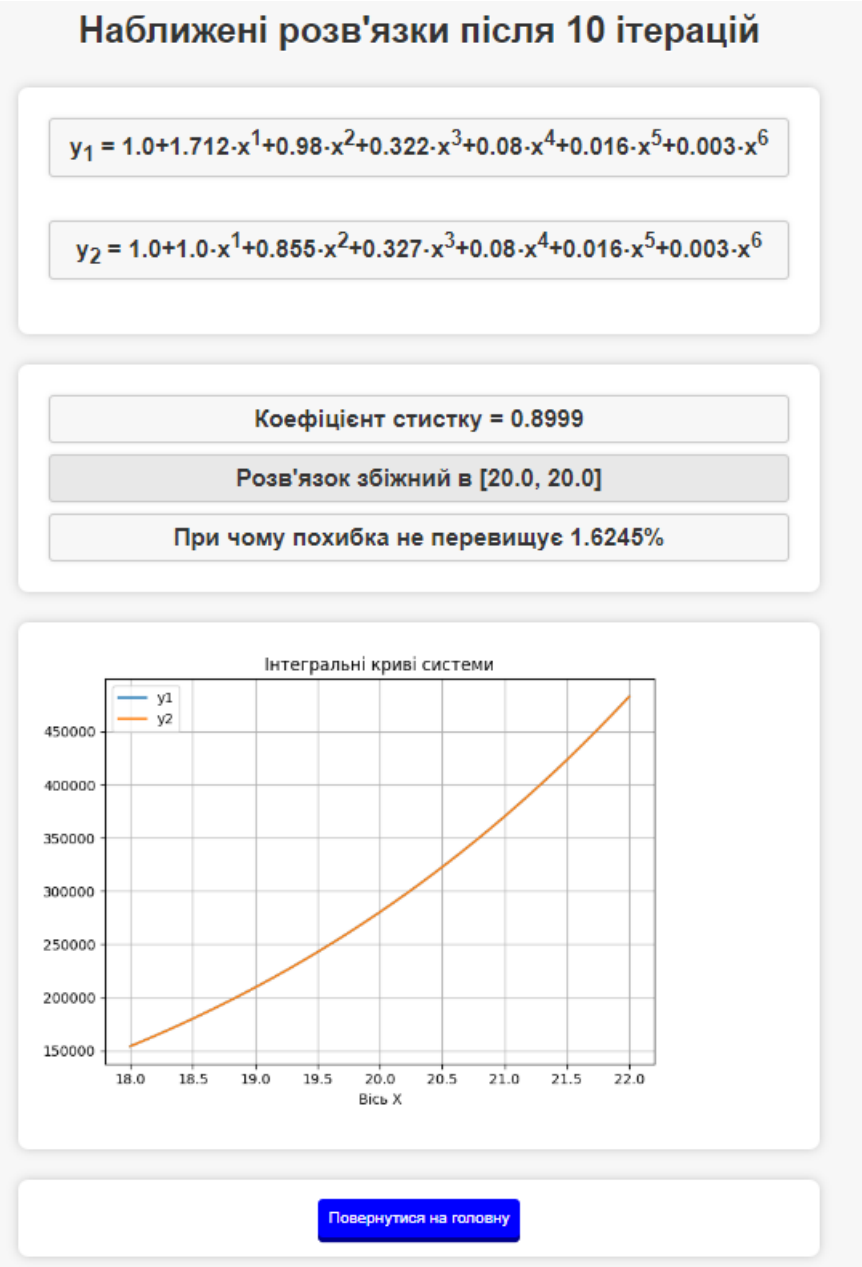


Рисунок 4.16 – Тест М.П.6, відображення розв’язку

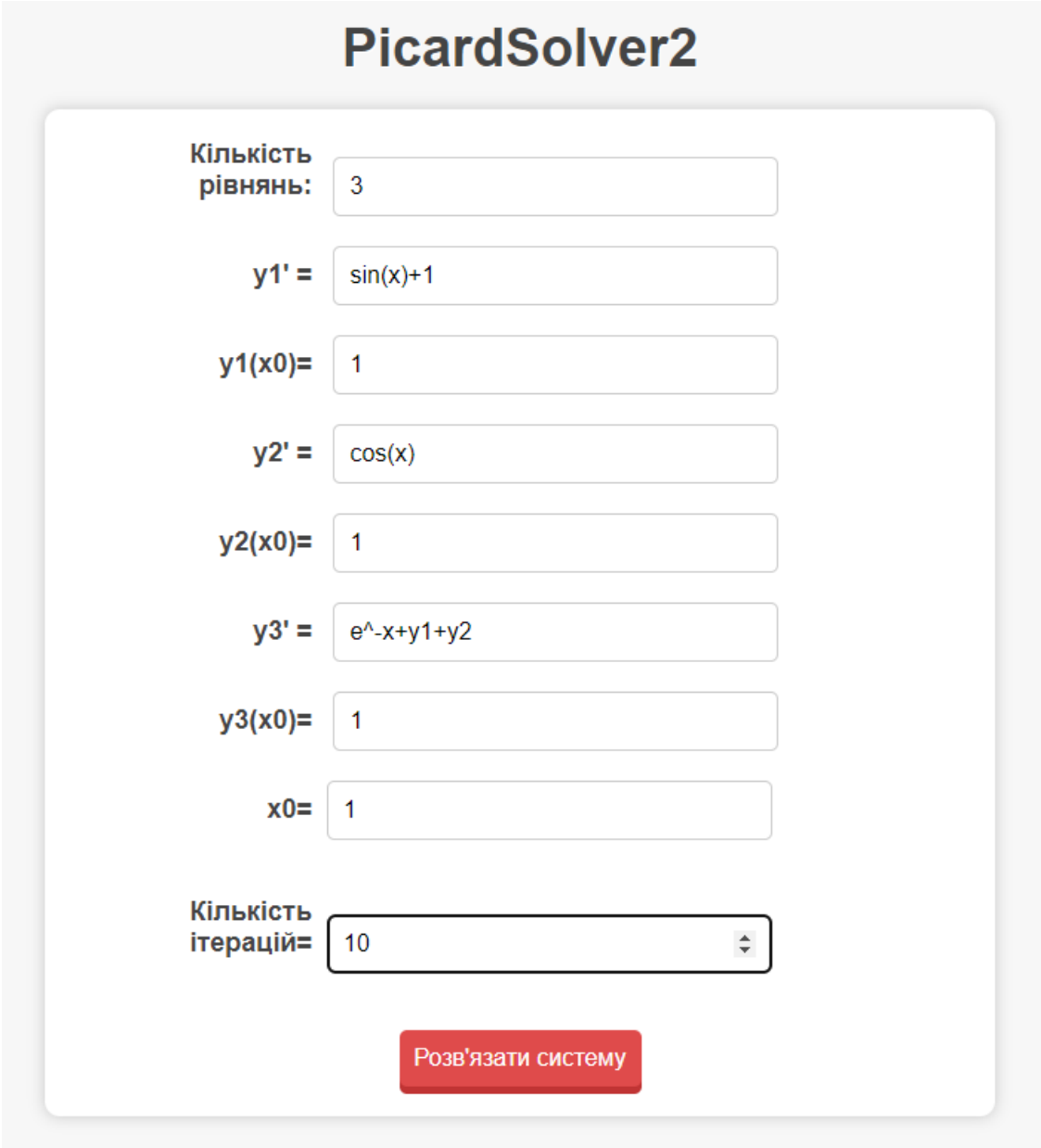


Рисунок 4.17 – Тест М.П.7, відображення введених даних

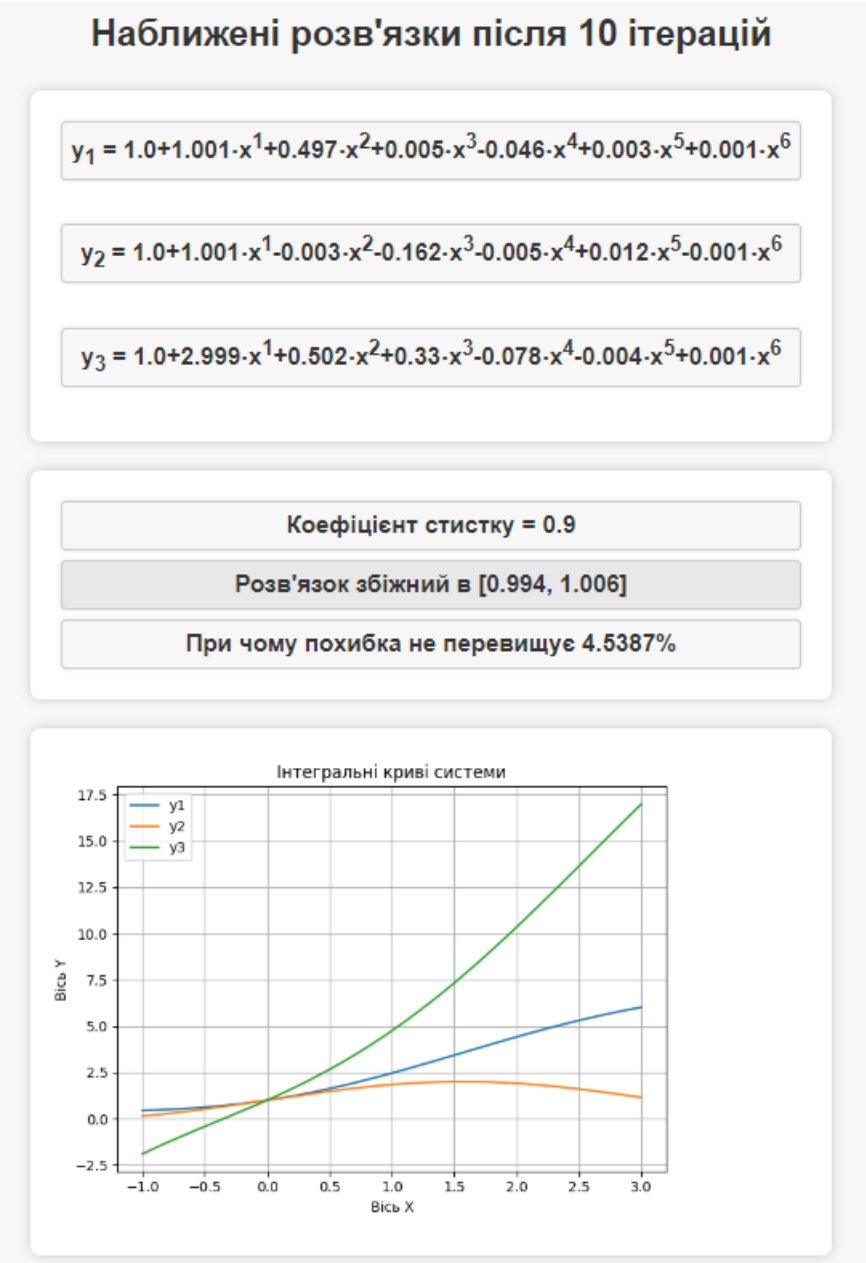


Рисунок 4.18 – Тест М.П.7, відображення розв’язку

Рисунок 4.19 – Тест М.П.8, частина 1

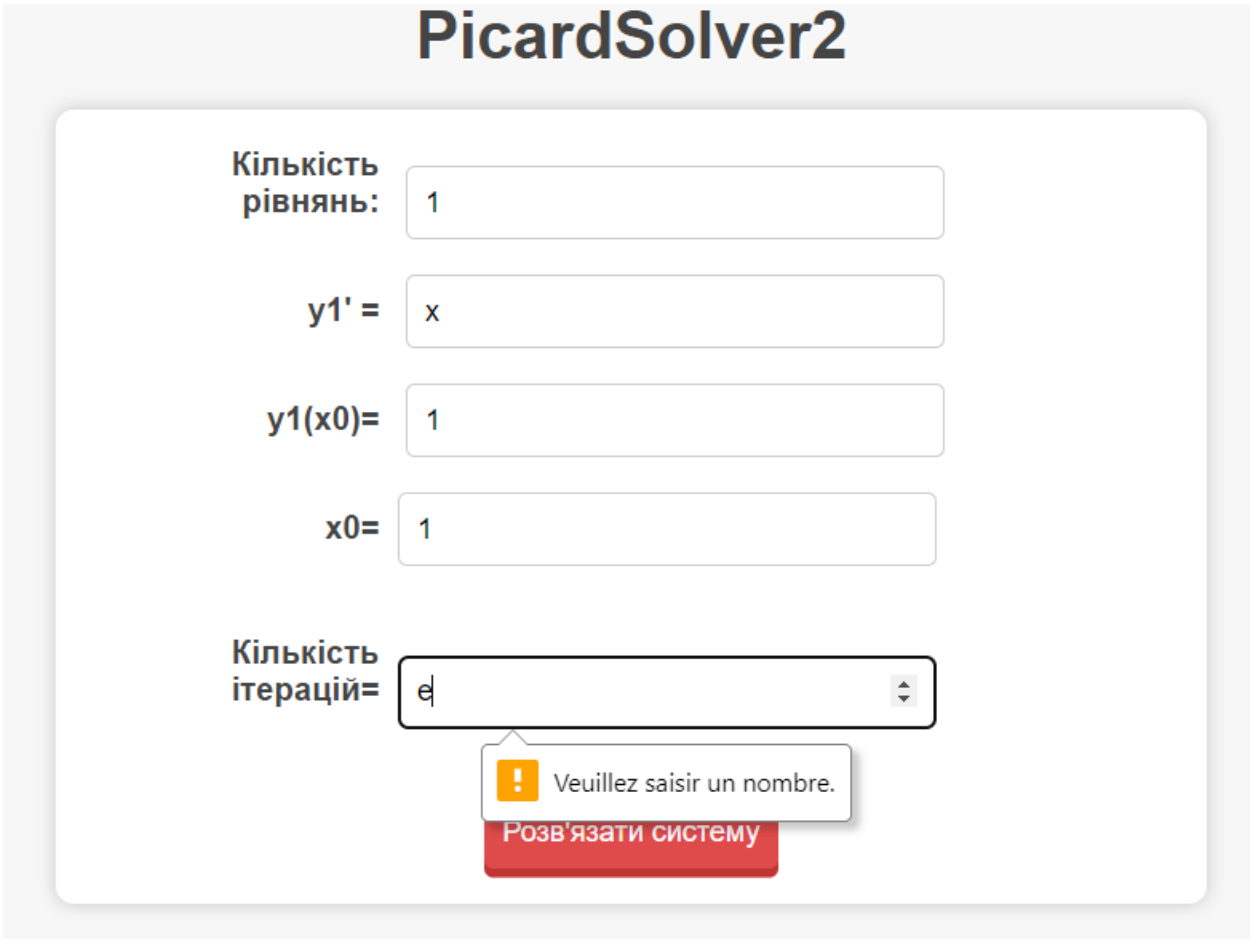


Рисунок 4.20 – Тест М.П.8, частина 2

Рисунок 4.21 – Тест М.П.8, частина 3.1

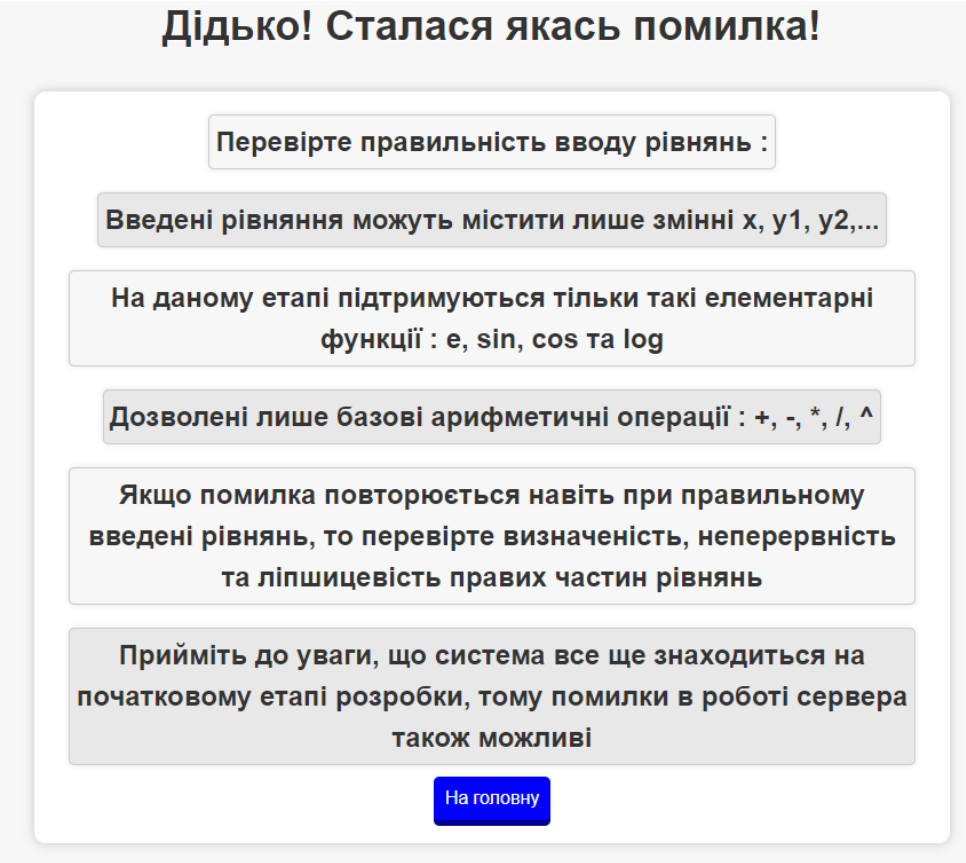


Рисунок 4.22 – Тест М.П.8, частина 3.2

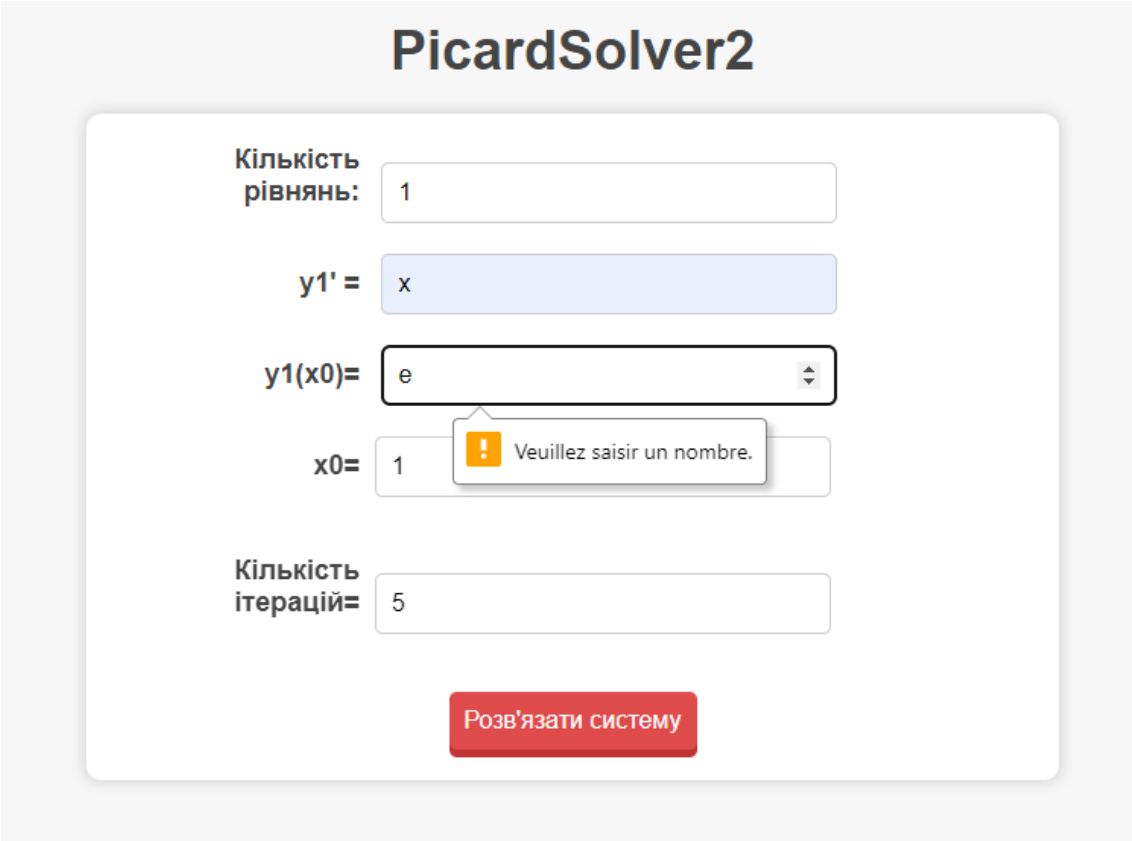


Рисунок 4.23 – Тест М.П.8, частина 4

Рисунок 4.24 – Тест М.П.8, частина 5

Рисунок 4.25 – Тест М.П.9, відображення вводу

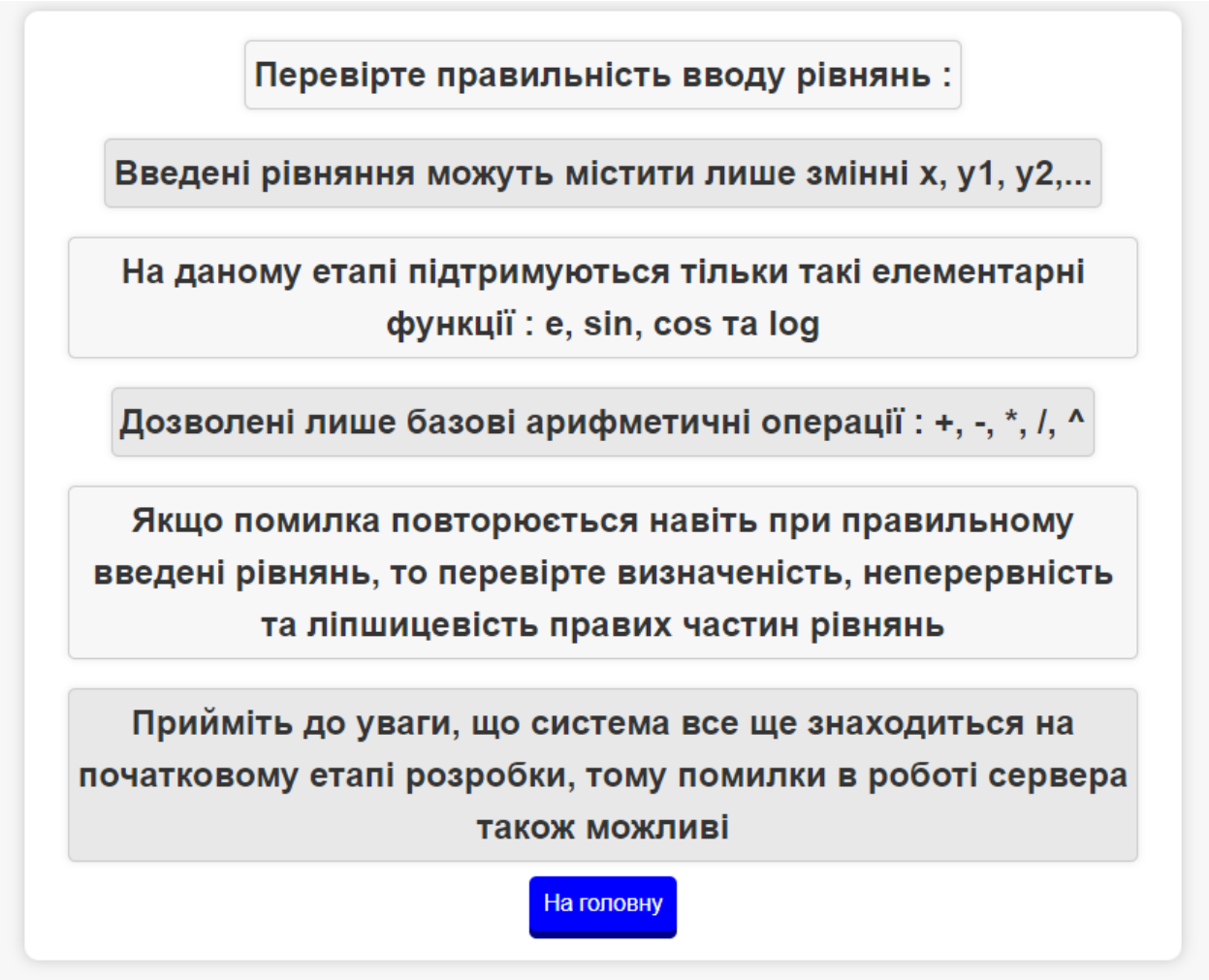


Рисунок 4.26 – Тест М.П.9, відображення результату

4.3 Висновки до розділу

У розглянутому розділі було проведено експериментальні розрахунки, що демонструють роботу розробленої системи. Тести та демонстрації проводилися за допомогою веб-застосунку для доступу до програми. Було знайдено розв'язок простої системи диференціальних рівнянь за допомогою розробленої програми, і результати експерименту співпали з аналітичним розв'язком. Після цього було виконано усі тести, наведені в попередньому розділі, і всі вони були пройдені успішно. Отже, можна зробити висновок, що розроблена система є коректною та може бути

використана для розв'язання складних диференціальних рівнянь з точністю, необхідною для багатьох наукових досліджень.

ВИСНОВКИ

За результатами виконання магістерської роботи було:

а) було виконано порівняльний аналіз існуючих наближених та чисельних методів для розв’язання задачі Коші для систем звичайних диференціальних рівнянь першого порядку. Було, зокрема, описано переваги і недоліки кожного з наведених методів, таких як методи, Рунге-Кутта, Адамса, Мілна, а також наближених методів Пікара та Чаплигіна для систем диференціальних рівнянь.

б) розроблено математичну модель модифікації методу Пікара послідовних наближень для систем звичайних диференціальних рівнянь першого порядку, для спрощення його застосування на ЕОМ. Ця модифікація полягає в заміні підінтегральних функцій систем їх рядами Тейлора та їх подальшому інтегруванні за стандартною процедурою. Для цієї модифікації було доведено збіжність та оцінено похибку. Було також запропоновано прийом подальшого спрощення розрахунків на ЕОМ, який має схожу ідею на метод Гауса-Зейделя для СЛАР. Насамкінець, запропоновано процедуру спряження розв’язків на кінцях локальних областей збіжності для інтегрування систем диференціальних рівнянь на інтервалах більших за початкову область збіжності.

в) описану модифікацію методу було програмно реалізовано у вигляді веб-орієнтовану застосунку, що дозволяє розв’язувати, методом Пікара, системи звичайних диференціальних рівнянь першого порядку, які містять до 10 рівнянь. Заропонована програмна реалізація дає розв’язок системи в аналітичному вигляді, на відміну від переважної більшості чисельних методів, які дають результат в табличному вигляді.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Колмогоров А.Н. Елементи теорії функцій та функціонального аналізу: Підручник / А.Н. Колмогоров, С.В. Фомін. – Київ: Видавниче об'єднання "Вища школа", 1974.
2. Самойленко А.М. Диференціальні рівняння: Підручник / А.М. Самойленко, М.О. Перестюк, І.О. Парасюк. – Київ: Либідь, 2003. – 2-ге вид., перероб. і доп. – 600 с.
3. Зорич В. А. Математичний аналіз. Частина II: Підручник / В. А. Зорич. – Київ: Вища школа, 1984. – 607 с.
4. Чисельні методи : навчальний посібник / В. М. Задачин, І. Г. Конюшенко. – Х.: Вид. ХНЕУ ім. С. Кузнеця, 2014. – 180 с.
5. Чисельні методи: Навчальний посібник. / Волонтир Л.О, Зелінська О.В., Потапова Н.А., Чіков І.А., Вінницький національний аграрний університет. – Вінниця: ВНАУ, 2020 – 322 с.
6. Чаплигін С. А. Новий метод наближеного інтегрування диференціальних рівнянь [Текст] / С. А. Чаплигін ; Під ред. В. К. Гольцмана. — Л.: Державне видавництво техніко-теоретичної літератури, 1950, – 103 с.
7. Єжов С.М. Методи обчислень: Навчальний посібник / С.М. Єжов. – Київ: ВПЦ "Київський університет", 2001, – 140 с.
8. Лось, В. М., Герасименко, В. Р., Копичко С.М. Модифікований метод ітерацій для систем диференціальних рівнянь // Прикладна математика та комп’ютинг, 16-18 листопада 2022 р.
9. Holmes, Mark H. Introduction to numerical methods in differential equations. New York, NY : Springer New York, 2007, – 239 p.

Додаток А. Лістинги програм

Модуль “main.py” :

```
import re
import numpy as np
import scipy.special
from scipy.misc import derivative
from math import sin, cos, e, log
import scipy.optimize as optimize
import sympy as sp
import matplotlib.pyplot as plt
import io
import base64


def list_to_formula_(coefs):
    res = str(coefs[0])
    for i in range(1, len(coefs)):
        if not np.isclose(coefs[i], 0):
            res+=f"+{coefs[i]}*x^{i}"
    return res.replace("+-", "-")


def find_alpha(string_f, x0, k, y_approx, m, alpha):
    M_i_list = []
    for i in range(m):
        # Define the variable x
        x = sp.Symbol('x')
        # Define the string
        string_y = list_to_formula_(y_approx[i])
        string_f[i] = string_f[i].replace("e(", "exp(").replace(f"y{i+1}", string_y)

        # Convert the string to a SymPy expression
        f = sp.sympify(string_f[i])
        f_y = sp.sympify(string_y)

        M_list = []
        for n in range(k):
            # Find the derivative of the expression
            f_prime = sp.diff(f, x, n+1)
            f_y_prime = sp.diff(f_y, x, 1)
            g = f_prime * f_y_prime
            g_func = sp.lambdify(x, g)
            def objective(x):
                return -g_func(x)
            try:
                M_k = optimize.minimize_scalar(objective, method='bounded', bounds=(x0-0.5, x0+0.5)).fun
            except:
```

```
            M_k = 30
        M_list.append(abs(M_k))
    M_i_list.append(max(M_list))
M = max(M_i_list)
end_int = abs(alpha/(M*k) + x0)
alpha = M*k*(end_int-x0)
return alpha, (round(x0-abs(end_int-x0),4), round(end_int,4))


def custom_func(func_str):
    # Функція приймає функцію-рядок
    # І повертає функцію пайтона зі строки.
    # Наприклад для "sin(x)" вона поверне функцію, що приймає 1 аргумент і обчислює значення синуса
    def result_func(x):
        # Це функція, що буде повернута
        nonlocal func_str
        # Обчисоюємо значення "функції", заданої рядком
        func_str = func_str.replace('^', '**')
        try:
            return eval(func_str.replace('x', '(' + str(float(x)) + ')'))
        except OverflowError:
            # Якщо отримане значення завелике - повернемо, як результат, 10 в степені 25
            return 10**+25
    return result_func


def integrate_linear(coefs, y_0_i = 0):
    res = [y_0_i]
    for i in range(len(coefs)):
        res.append(coefs[i]/(i+1))
    return res


def list_to_formula(coefs, precision=2):
    res = str(coefs[0])
    for i in range(1, len(coefs)):
        if not np.isclose(coefs[i], 0):
            res+=f"+{round(coefs[i], precision)}*x^{i}"
    return res.replace("+-", "-")


def multiply_polys(poly_list1, poly_list2):
    p1 = np.array(poly_list1)
    term_mult = np.zeros(len(poly_list1)-1)
    for i in range(len(poly_list2)):
        zeros = np.zeros(i)
        term_mult = np.concatenate((term_mult, np.zeros(1)))
        term = p1 * poly_list2[i]
        term_mult += np.concatenate((zeros, term))
    return np.array(term_mult)
```

```
def poly_to_power(poly_list, power):
    base = poly_list
    for i in range(int(power)-1):
        poly_list = multiply_polys(poly_list, base)
    return np.array(poly_list)


def add_arr(a, b):
    if len(a) < len(b):
        c = b.copy()
        c[:len(a)] += a
    else:
        c = a.copy()
        c[:len(b)] += b
    return c


def factorial(n):
    if n == 0:
        return 1
    return n*factorial(n-1)


def to_taylor_series(func_str, y_i_estimations, x0, n):
    for key in y_i_estimations.keys():
        func_str = func_str.replace(key, "(" + list_to_formula(y_i_estimations[key]) + ")")
    f_i = custom_func(func_str)
    result_poly = np.array([f_i(x0)])
    for i in range(1, n+1):
        coef = derivative(f_i, x0, n=i, order=33)/factorial(i)
        result_poly = add_arr(result_poly, coef*poly_to_power([-x0, 1], i))
    return result_poly


def plot_integral_curves(current_y_estim, interval):
    inter = np.linspace(interval[0], interval[1], 1000)
    fig = plt.figure()
    plt.title('Інтегральні криві системи')
    plt.xlabel('Вісь X')
    plt.ylabel('Вісь Y')
    plt.grid(True)
    plt.tight_layout()
    for y_i in current_y_estim.keys():
        f = custom_func(list_to_formula_(current_y_estim[y_i]))
        y_vals = [f(x) for x in inter]
        plt.plot(inter, y_vals, label = f"{y_i}")
        plt.legend()
    plt.show()
    # Save the plot to a PNG image in memory
    output = io.BytesIO()
    fig.savefig(output, format='png')
    output.seek(0)
```

```
    # Encode the PNG image in Base64 and convert to a string
    encoded_img = base64.b64encode(output.getvalue()).decode('utf-8')
    return encoded_img


def plot_integral_curves_RK(t, y):
    fig = plt.figure()
    plt.title('Інтегральні криві системи')
    plt.xlabel('Вісь X')
    plt.ylabel('Вісь Y')
    plt.grid(True)
    plt.tight_layout()
    for i in range(y.shape[1]):
        plt.plot(t, y[:, i], label=f'y{i+1}')

    plt.legend()

    # Save the plot to a PNG image in memory
    output = io.BytesIO()
    fig.savefig(output, format='png')
    output.seek(0)

    # Encode the PNG image in Base64 and convert to a string
    encoded_img = base64.b64encode(output.getvalue()).decode('utf-8')
    return encoded_img


def add_superscript(string):
    # Функція для прикрашання рядка, додаванням тегів для відображення степенів як суперскрипт у HTML
    # Приймає рядок
    # і повертає розставлені теги у цьому ж рядку
    # Наприклад, 'x^1' -> 'x<sup>1</sup>'
    # Це невеликий костиль, щоб щось типу x^-1 не розглядалось як 2 доданки
    string = string.replace('^-', '^%')
    # Якщо немає піднесення у степінь, то не потрібно нічого і робити
    if '^' in string:
        # Це лічильник кількості степенів, що були зустріті підряд. Напр,якщо пройдено "x^x^", то counter_sups == 2
        counter_sups = 1
        # Булева змінна, що позначає, чи знаходимось ми "у степені"
        in_power = True
        # Знаходимо індекс першого піднесення у степінь
        inx = string.index('^')+1
        # Копіюємо у результуючу змінну все до піднесення у степінь і додаємо відкритий тег
        new_string = string[:inx-1] + '<sup>'
        # Семафор відслідковує кількість дужок.
        # Наприклад, якщо ми пройшли частину "sin(x" рядку sin(x), то semaphore == 1
        semaphore = 0
        # Це головний цикл, де ми проходимось по рядку
        for char in string[inx:]:
            # Обробка семафору
            if char == '(' and in_power is True:
```

```
                semaphore += 1
                new_string += char
            elif char == ')' and in_power is True and semaphore > 0:
                semaphore -= 1
                new_string += char
            elif char == ')' and in_power is True and semaphore == 0:
                # Якщо дужка закривається і це остання дужка у степені,
                # то закриваємо тег стільки разів, скільки зустріли піднесення у степінь
                new_string += '</sup>'*counter_sups
                counter_sups = 0
                new_string += char
            elif char in ['+', '-', '*', '/'] and semaphore == 0 and in_power is True:
                # Якщо ми поза дужками у степені і зустрічаємо операції, то зариваємо тег
                in_power = False
                new_string += '</sup>'*counter_sups
                counter_sups = 0
                new_string += char
            elif char == '^' and semaphore == 0 and in_power is True:
                # Якщо зустрічаємо піднесення у степінь, коли ми вже "у степені", то додаємо новий тег і counter_sups=+1
                counter_sups += 1
                new_string += '<sup>'
            elif char == '^' and in_power is False:
                # Якщо ми не у піднесенні у степінь, але зустрічаємо його, то знову відкриваємо тег
                in_power = True
                counter_sups += 1
                new_string += '<sup>'
            else:
                new_string += char
        if in_power is True:
            # Знову ж таки, якщо закінчився цикл і ми все ще у степені, то рядок закінчився піднесенням у степінь
            new_string += '</sup>'*counter_sups
    else:
        new_string = string
    # Див. початок, щоб зрозуміти заміну
    return new_string.replace('%', '-')


def runge_kutta_system(f, t_span, y0, h):
    # f is a list of strings representing the system of differential equations
    # t_span is a tuple (t0, tf) representing the initial and final times
    # y0 is a dictionary representing the initial conditions
    # h is the step size
    # Returns a tuple (t, y) representing the time points and the solution

    # Define the number of time steps and initialize the solution
    t0, tf = t_span
    n = int((tf - t0) / h) + 1
    t = np.linspace(t0, tf, n)
    y = np.zeros((n, len(y0)))
    y[0, :] = [y0[key][0] for key in y0]

    # Define the function to evaluate the system of differential equations
```

```
    def eval_f(f_str, y_val, t_val):
        var_dict = {f"y{i + 1}": y_val[i] for i in range(len(y_val))}
        var_dict["x"] = t_val
        return eval(f_str, var_dict)

    # Perform the Runge-Kutta method
    for i in range(n - 1):
        k1 = [h * eval_f(f[j], y[i, :], t[i]) for j in range(len(f))]
        k2 = [h * eval_f(f[j], y[i, :] + 0.5 * np.array(k1), t[i] + 0.5 * h) for j in range(len(f))]
        k3 = [h * eval_f(f[j], y[i, :] + 0.5 * np.array(k2), t[i] + 0.5 * h) for j in range(len(f))]
        k4 = [h * eval_f(f[j], y[i, :] + np.array(k3), t[i] + h) for j in range(len(f))]
        y[i + 1, :] = y[i, :] + (1 / 6) * (np.array(k1) + 2 * np.array(k2) + 2 * np.array(k3) + np.array(k4))

    print(t.shape, y.shape)
    image = plot_integral_curves_RK(t, y)
    return image


def picard_general(func_strings, x0, init_cond, taylor_order=5, num_iters=3, interval = None): # TODO : Gauss-Zeydel + порівняння точності
    # func_strings - list of strings, which contain right hand sides of equations
    # x0 - float - point around which to find solutions
    # init_cond - dict of initial conditions like {"y1" : [1], "y2" : [2]}
    # current_y_estim - dict like init_cond - current estimation of solution
    k = taylor_order
    m = len(init_cond.keys())

    if interval is None:
        interval = [x0-2, x0+2]

    current_y_estim = init_cond
    for iteration in range(num_iters):
        equation_inx = 0
        prev_y_estim_buffer = current_y_estim
        for y_i in current_y_estim.keys():
            # taylor_poly - list, which represents the polynomial approximation of the current solution
            taylor_poly = to_taylor_series(func_strings[equation_inx], current_y_estim, x0, taylor_order)
            # integrate the polynomial to get next approximation
            y_0_i=init_cond[y_i][0]
            y_i_next = integrate_linear(taylor_poly, y_0_i=y_0_i)
            equation_inx += 1
            #current_y_estim[y_i] = y_i_next # Gauss-Zeydel
            prev_y_estim_buffer[y_i] = y_i_next
        current_y_estim = prev_y_estim_buffer

    y_approx = list(current_y_estim.values())
    encoded_img = plot_integral_curves(current_y_estim, interval)
    a, interval = find_alpha(func_strings, x0, k, y_approx, m, 0.9)
    return current_y_estim, encoded_img, a,interval  # return dictionary - last approximation of solutions - format like init_cond
```

Модуль “server.py”

```
from flask import Flask, render_template, request
from main import picard_general, list_to_formula, add_superscript, runge_kutta_system
import random

GLOBAL_PRECISION = 4

app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def index():
    return render_template('input.html')

@app.route('/solve', methods=['GET', 'POST'])
def solve_equations():
    if request.method == 'POST':

        num_eqns = int(request.form['num_eqns'])
        x0 = float(request.form['x0'])
        func_strings = [request.form[f'eqn{i+1}'] for i in range(num_eqns)]
        init_conds = {}
        for i in range(num_eqns):
            eqn_name = f'y{i+1}'
            eqn_val = float(request.form[f'{eqn_name}_0'])
            init_conds[eqn_name] = [eqn_val]
        # int(request.form['taylor_order'])
        taylor_order = 5
        num_iters = int(request.form['iters'])
        method = request.form['method']
        if method == "Picard":
            solutions, encoded_img, a,interval = picard_general(func_strings, x0, init_conds, taylor_order, num_iters)
            error = round(random.uniform(1, 5),4)
            for key in solutions.keys():
                solutions[key] = add_superscript(list_to_formula(solutions[key], precision=GLOBAL_PRECISION)).replace("*","·")
            return render_template('output.html', solution=solutions, num_iters=num_iters, image_data=encoded_img,
                                   alpha = round(a,4), start = interval[0], end = interval[1], error = error, method = method, image_data_picard = encoded_img)
        if method == "RK4":
            encoded_img = runge_kutta_system(func_strings, (x0, x0 + 2), init_conds, 0.05)
            solutions, encoded_img2, a,interval = picard_general(func_strings, x0, init_conds, taylor_order, num_iters,interval=[x0, x0 + 2])
            error = round(random.uniform(1, 5),4)
            for key in solutions.keys():
                solutions[key] = add_superscript(list_to_formula(solutions[key], precision=GLOBAL_PRECISION)).replace("*","·")

            return render_template('output.html', solution=solutions, num_iters=num_iters, image_data=encoded_img,
                                   alpha=round(a, 4), start=x0, end=x0+2, error=error, method = method, image_data_picard = encoded_img2)

    else:
        return render_template('input.html')
```

```
if __name__ == '__main__':
    app.run(debug=True)
```

Сторінка “output.html”

```
            <!DOCTYPE html>
<html>
<head>
  <title>Пощастило! Розв'язок знайдено!</title>
  <style>
    body {
      background-color: #f7f7f7;
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
      color: #333333;
      font-size: 16px;
      line-height: 1.5;
    }
    h1 {
      font-size: 36px;
      margin: 20px 0;
      color: #333333;
      text-align: center;
      text-shadow: 1px 1px #ffffff;
    }
    .container {
      justify-content: center;
      align-items: center;
      margin: 0 auto;
      max-width: 800px;
      padding: 20px;
      box-shadow: 0 0 10px rgba(0,0,0,0.2);
      background-color: #ffffff;
      border-radius: 10px;
      box-sizing: border-box;
      margin-top: 30px;
      margin-bottom: 30px;
    }
    .container p {
      margin: 10px;
      padding: 5px;
      font-size: 24px;
      font-weight: bold;
      text-align: center;
      color: #333333;
      border: 1px solid #cccccc;
      border-radius: 5px;
      background-color: #f7f7f7;
```

```
      box-shadow: 0 0 5px rgba(0,0,0,0.1);
      word-wrap: break-word;
    }
    .container p:nth-child(2n) {
      background-color: #e8e8e8;
    }


            input[type="submit"] {
      background-color:  #0000FF;
      color: white;
      font-size: 16px;
      padding: 10px;
      border: none;
      border-radius: 5px;
      cursor: pointer;
      box-shadow: 0 5px 0 #00008B;
      transition: all 0.1s ease-in-out;
        display: block;
  margin: 0 auto;
      text-align: center;
    }
    input[type="submit"]:hover {
      background-color: #00008B;
      box-shadow: 0 3px 0 #00008B;
      transform: translateY(2px);
        display: block;
  margin: 0 auto;
      text-align: center;
    }

  </style>

  </style>
</head>
<body>
  {% if method == "Picard" %}
  <h1>Наближені розв'язки після {{num_iters}} ітерацій</h1>
  {% endif %}

  {% if method != "Picard" %}
  <h1>Наближені розв'язки системи за методом Рунге-Кутта</h1>
  {% endif %}

  {% if method == "Picard" %}
  <div class="container">
    {% if solution is defined %}
      {% for y in solution %}
    <p>y<sub>{{loop.index}}</sub> = {{solution[y] | safe}}</p>
        <br>
      {% endfor %}
    {% else %}
```

```
      <p>Розв'язків не знайдено</p>
    {% endif %}
  </div>

  <div class="container">
    <p>Коефіцієнт стистку = {{alpha}}</p>
    <p>Розв'язок збіжний в [{{start}}, {{end}}]</p>
    <p>При чому похибка не перевищує {{error}}%</p>
  </div>

    <div class="container">
    <img src="{{ image_data }}" alt="Matplotlib Plot">
  </div>
  {% endif %}

  {% if method != "Picard" %}
  <div class="container">
    <img src="{{ image_data }}" alt="Matplotlib Plot">
  </div>

  <h1>Наближені розв'язки системи за методом Пікара після {{num_iters}} ітерацій</h1>

  <div class="container">
    {% if solution is defined %}
      {% for y in solution %}
    <p>y<sub>{{loop.index}}</sub> = {{solution[y] | safe}}</p>
        <br>
      {% endfor %}
    {% else %}
      <p>Розв'язків не знайдено</p>
    {% endif %}
  </div>

  <div class="container">
    <p>Коефіцієнт стистку = {{alpha}}</p>
    <p>Розв'язок збіжний в [{{start}}, {{end}}]</p>
    <p>При чому похибка не перевищує {{error}}%</p>
  </div>

    <div class="container">
    <img src="{{ image_data_picard }}" alt="Matplotlib Plot">
      </div>
{% endif %}

<div class="container">
    <form method="GET" action="{{ url_for('solve_equations') }}">
    <input type="submit" value="Повернутися на головну" >
      </form>
      </div>

</body>
</html>
```

Сторінка “input.html”

```
<!DOCTYPE html>
<html>
<head>
  <title>Дикий в плані дизайну сайт</title>
  <style>

    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
      background-color: #f7f7f7;
    }
    h1 {
      font-size: 36px;
      margin: 20px 0;
      color: #444444;
      text-align: center;
      text-shadow: 1px 1px #ffffff;
    }
    form {
      margin: 20px 0;
      padding: 20px;
      background-color: #ffffff;
      border-radius: 10px;
      box-shadow: 0 0 10px rgba(0,0,0,0.2);
      max-width: 600px;
      margin: 0 auto;
    }
    label {
      display: inline-block;
      margin-bottom: 10px;
      margin-right: 10px;
      font-size: 18px;
      font-weight: bold;
      color: #444444;
      width: 160px;
      text-align: right;
    }
    input[type="text"], input[type="number"] {
      padding: 10px;
      font-size: 16px;
      border-radius: 5px;
      border: 1px solid #ccc;
      width: 300px;
      box-sizing: border-box;
      margin-bottom: 20px;
    }
```

```
    select {
      padding: 10px;
      font-size: 16px;
      border-radius: 5px;
      border: 1px solid #ccc;
      width: 300px;
      box-sizing: border-box;
      margin-bottom: 20px;
    }
    input[type="submit"] {
      background-color: #df4b4b;
      color: white;
      font-size: 16px;
      padding: 10px;
      border: none;
      border-radius: 5px;
      cursor: pointer;
      box-shadow: 0 5px 0 #c03434;
      transition: all 0.1s ease-in-out;
       display: block;
  margin: 0 auto;
      text-align: center;
    }
    input[type="submit"]:hover {
      background-color: #c03434;
      box-shadow: 0 3px 0 #c03434;
      transform: translateY(2px);
       display: block;
  margin: 0 auto;
      text-align: center;
    }
  </style>
</head>
<body>
  <h1>PicardSolver2</h1>
  <form method="POST" action="{{ url_for('solve_equations') }}">
    <label for="numEquations">Кількість рівнянь:</label>
    <input type="number" id="numEquations" name="num_eqns" min="1" max="10" required>
    <div id="equations"></div>
    <input type="submit" value="Розв'язати систему" >
  </form>

  <script>
    // function to generate equation input fields based on number of equations
    function generateEquationInputs(numEquations) {
      let equationInputs = "";
      for (let i = 1; i <= numEquations; i++) {
        equationInputs += `
          <label for="equation${i}">y${i}' =</label>
          <input type="text" id="equation${i}" name="eqn${i}" placeholder="Наприклад : sin(x)+y${i}" required>
          <label for="initial${i}">y${i}(x0)=</label>
          <input type="number" placeholder="Наприклад : ${i}" id="initial${i}" name="y${i}_0" required>
```

```
            <br>
          `;
        }
        equationInputs += '<label htmlFor="x0">x0=</label><input placeholder="Наприклад : 0" type="number" id="x0" name="x0" required><br><br>';

        equationInputs += '<label htmlFor="iters">Кількість ітерацій=</label><input placeholder="Наприклад : 5" type="number" id="iters" name="iters" required><br><br>';

        equationInputs += '<label htmlFor="method">Метод :</label><select id="method" name="method" required><option value="Picard">Пікара</option><option value="RK4">Рунге-Кутта та Пікара</option></select><br><br>';

        document.getElementById("equations").innerHTML = equationInputs;
      }

      // listen for changes to the number of equations field
          document.getElementById("numEquations").addEventListener("change", function() {
        let numEquations = parseInt(this.value);
        if (numEquations >= 1 && numEquations <= 10) {
          generateEquationInputs(numEquations);
        }
      });

    </script>
</body>
</html>
```

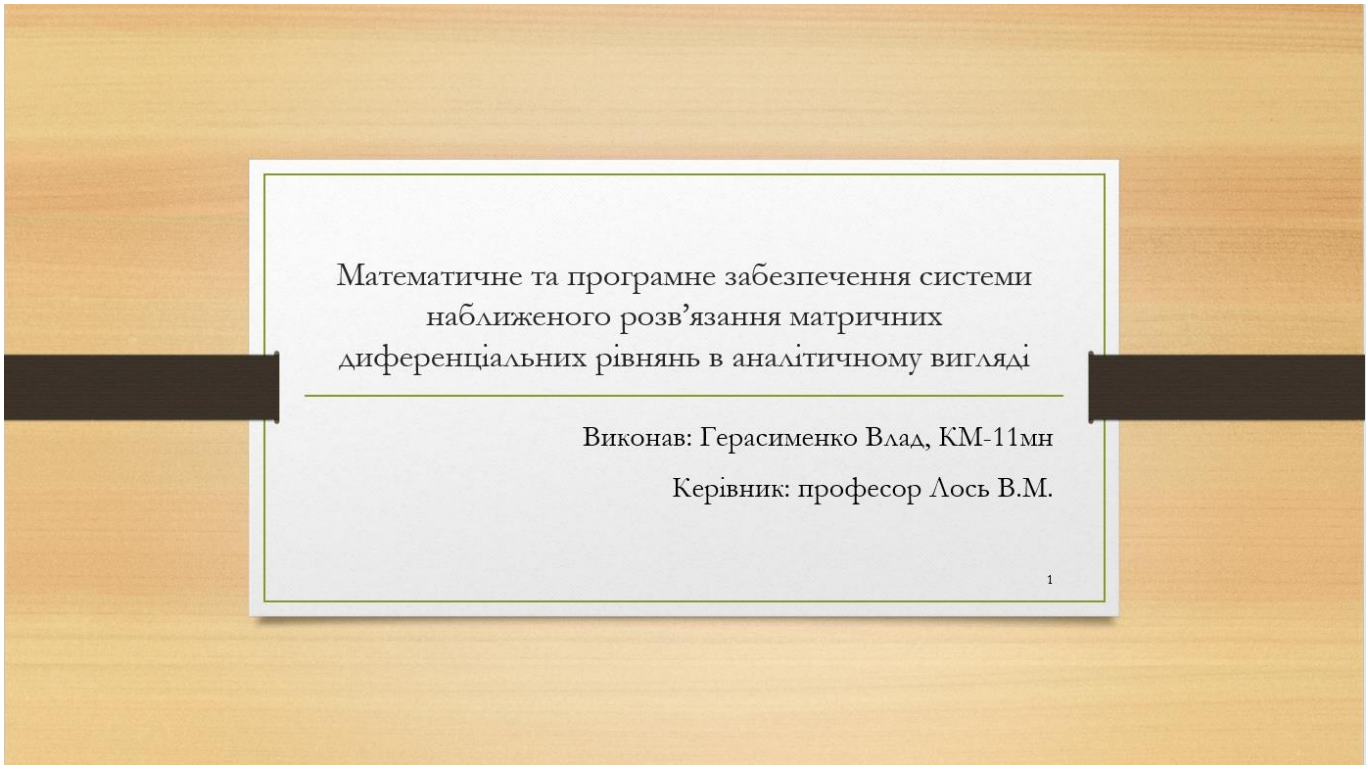
Сторінка “Error.html”

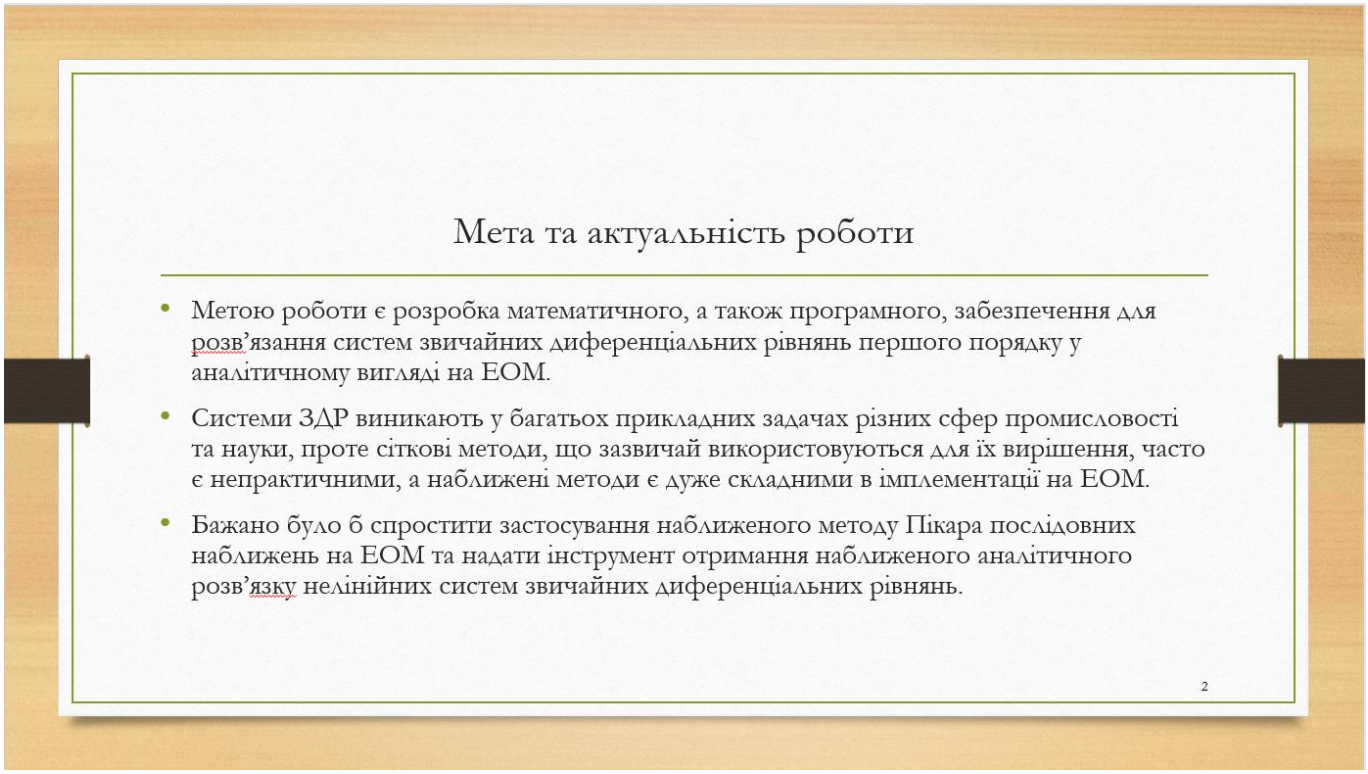
```
            <!DOCTYPE html>
<html>
<head>
  <title>Хай йому грець! Помилка!</title>
  <style>
    body {
      background-color: #f7f7f7;
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
      color: #333333;
      font-size: 16px;
      line-height: 1.5;
    }
    h1 {
      font-size: 36px;
      margin: 20px 0;
      color: #333333;
      text-align: center;
      text-shadow: 1px 1px #ffffff;
    }
    .container {
```

```
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
    align-items: center;
    margin: 0 auto;
    max-width: 800px;
    padding: 20px;
    box-shadow: 0 0 10px rgba(0,0,0,0.2);
    background-color: #ffffff;
    border-radius: 10px;
    box-sizing: border-box;
    margin-top: 30px;
    margin-bottom: 30px;
  }
  .container p {
    margin: 10px;
    padding: 5px;
    font-size: 24px;
    font-weight: bold;
    text-align: center;
    color: #333333;
    border: 1px solid #cccccc;
    border-radius: 5px;
    background-color: #f7f7f7;
    box-shadow: 0 0 5px rgba(0,0,0,0.1);
    word-wrap: break-word;
  }
  .container p:nth-child(2n) {
    background-color: #e8e8e8;
  }
  .container p:first-child {
    margin-top: 0;
  }
  .container p:last-child {
    margin-bottom: 0;
  }

      input[type="submit"] {
    background-color:  #0000FF;
    color: white;
    font-size: 16px;
    padding: 10px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    box-shadow: 0 5px 0 #00008B;
    transition: all 0.1s ease-in-out;
     display: block;
margin: 0 auto;
    text-align: center;
  }
  input[type="submit"]:hover {
```

```
        background-color: #00008B;
        box-shadow: 0 3px 0 #00008B;
        transform: translateY(2px);
          display: block;
  margin: 0 auto;
        text-align: center;
      }

  </style>
</head>
<body>
  <h1>Дідько! Сталася якась помилка!</h1>
  <div class="container">
      <p>Перевірте правильність вводу рівнянь : </p>
      <p>Введені рівняння можуть містити лише змінні x, y1, y2,...</p>
      <p>На даному етапі підтримуються тільки такі елементарні функції : e, sin, cos та log</p>
      <p>Дозволені лише базові арифметичні операції : +, -, *, /, ^ </p>
      <p>Якщо помилка повторюється навіть при правильному введені рівнянь, то перевірте визначеність, неперервність та ліпшицевість правих частин рівнянь </p>
      <p>Прийміть до уваги, що система все ще знаходиться на початковому етапі розробки, тому помилки в роботі сервера також можливі </p>

      <form method="GET" action="{{ url_for('solve_equations') }}">
    <input type="submit" value="На головну" >
  </form>
   </div>

</body>
</html>
```

Додаток Б. Ілюстративний матеріал





Опис проблематики

- Нехай маємо систему звичайних диференціальних рівнянь першого порядку наступного вигляду :

$$\begin{cases} y_1' = f_1(x, y_1(x) \dots y_m(x)) \\ \dots \\ y_m' = f_m(x, y_1(x) \dots y_m(x)) \end{cases} \quad (1)$$

- З початковими умовами $y_1(x_0) = y_{0,1} \dots y_m(x_0) = y_{0,m}$ **(2)**
- Функції $f_1 \dots f_m$ припускаються визначеними, неперервними та ліпшицевими у області інтегрування I (для існування та єдиності розв'язку).
- Задача полягає розробці модифікації методу Пікара для систем **(1)-(2)** для спрощення його застосування на ЕОМ

3

Існуючі методи

- Існує велика кількість методів чисельного розв'язання задачі Коші, проте здебільшого, у практичних задачах користуються методами Рунге-Кутта або Адамса.
- Існують також і менш популярні методи, як от метод Ейлера і модифікований метод Ейлера, метод Сімпсона і подібні.

4

Існуючі методи

- Беззаперечним плюсом методів Рунге-Кутта та Адамса є швидкість роботи і простота реалізації, але вони поступаються методам Пікара та Чаплигіна у простоті підвищення точності та своїм табличним представленням розв'язку (навідміну від аналітичного у методі Пікара)

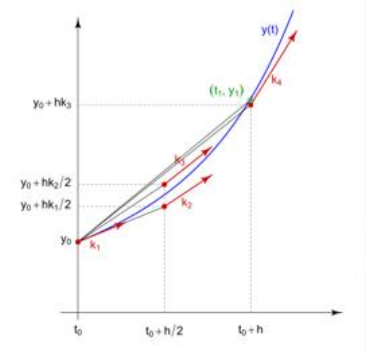


Рисунок 1 – Робота методу Рунге-Кутта 4-го порядку

5

Існуючі програмні засоби

- Ймовірно, найбільш популярним засобом розв'язання диференціальних рівнянь є веб-додаток «Wolfram Alpha», який позиціонується як «науковий онлайн-калькулятор» і здатний розв'язувати дуже велику кількість різних задач, зокрема і знаходити інтегральні криві диференціальних рівнянь у аналітичному вигляді, а також імплементує різні чисельні методи, наприклад, метод Рунге-Кутта.

6

Існуючі програмні засоби

- Тим не менш, легко знайти неймовірно багато інших програмних засобів для чисельного знаходження розв'язку звичайних диференціальних рівнянь.
- Серед таких можна виділити, наприклад, системи обробки природньої мови, як «ChatGPT».

7

Опис класичного алгоритму методу Пікара

- Тоді класичний метод Пікара базується на наступній ітеративній процедурі, яка полягає у інтегруванні кожного з рівнянь системи:

$$\varphi_i^n(x) = y_{0,i} + \int_{x_0}^{x} f_i\left(\tau, \varphi_1^{n-1}(\tau), \dots, \varphi_m^{n-1}(\tau)\right) d\tau, \quad \textbf{(3)}$$

- Де $\varphi_i^n(x)$ – це *n*-те наближення *i*-ї компоненти розв'язку $\hat{y} = (\hat{y}_1 \dots \hat{y}_m)$. $f_i(x, y_1 \dots y_m)$ – права частина диференціального рівняння, а $y_i(x_0) = y_{0,i}; i = \overline{1,m}$ – початкові умови.

8

Опис запропонованої модифікації методу Пікара

- З наведеної формули видно, що найважчою частиною цієї процедури – є інтегрування правої частини рівняння.
- Дійсно, інтегрування є дуже складною, алгоритмічно, операцією і може привести до квадратур, що можуть навіть і не виражатися у елементарних функціях.
- Оскільки інтегрувати поліноми – досить просто, то модифікуємо метод таким чином, щоб тільки інтегрувати поліноми.

9

Опис запропонованої модифікації методу Пікара

- Для цього замінимо підінтегральні у функції у правій частині системи їх рядами Тейлора, відкинемо залишковий член і використовуватимемо отримане наближення функції у ітеративній процедурі **(3).**

$$f_i\left(\tau, \varphi_1(\tau), \dots, \varphi_m(\tau)\right) \approx \sum_{n=0}^{k} \frac{f_i^{(n)}(a, \varphi_1(a), \dots, \varphi_m(a))}{n!} (\tau - a)^n \quad \textbf{(4)}$$

- Не вдаватимемося тут у деталі доведення, яке є доволі громіздким, але відмітимо, що у роботі доведено, що така модифікація методу Пікара є збіжною до справжнього розв'язку.

10

Оцінка точності

- Класичний метод Пікара має наступну похибку :

$$\Delta_s = \frac{\alpha^s \cdot \rho(\varphi^0, \varphi^1)}{1-\alpha} \quad (5)$$

Тут α – коефіцієнт стиску оператора A ітераційної процедури, $\rho(\varphi^0, \varphi^1)$– відстань між нульовим та першим наближенням розв'язку, s – кількість ітерацій.

11

Оцінка точності

- Так як ця модифікація методу Пікара є, фактично, іншим методом, то ясно, що похибка модифікації буде відмінною від похибки класичного методу Пікара.
- Дійсно, адже у цій модифікації методу ми змінюємо вигляд ітераційної процедури, при чому розкладаючи підінтегральні функції у ряд Тейлора, ми «додаємо» похибку цього розкладення до загальної похибки методу.

12

Оцінка точності

- У роботі показано, що похибка на s-му кроці методу не перевищуватиме наступний вираз

$$\rho\left(\tilde{A}^s\varphi, A^s\varphi\right) < \frac{\tilde{M}\cdot(\alpha+1)^{m-1}\cdot(b-a)^{k+2}}{(k+2)!}, \textbf{(6)}$$

тут $\tilde{M} = max_{x,i}\left(f_i^{(k+1)}\left(x, \varphi_1(x), \dots, \varphi_m(x)\right)\right)$, α – коефіцієнт стиску оператора A, s – кількість ітерацій, k – порядок до якого розкладається функція у ряд Тейлора.

13

Загальна оцінка похибки

- Загальну похибку модифікації метода отримуємо прямою сумою виразів (5) і (6). Дійсно, згадавши третю аксіому метричних просторів, матимемо, що «відхилення» справжнього розв'язку від розв'язку, отриманого на кроці s модифікації методу, виражається наступним чином:

$$\rho\left(\tilde{A}^s\varphi, \varphi\right) < \frac{\alpha^s\cdot\rho(\varphi^0,\varphi^1)}{1-\alpha} + \frac{\tilde{M}\cdot(\alpha+1)^{m-1}\cdot(b-a)^{k+2}}{(k+2)!} \textbf{(7)}$$

14

Процедура спряження розв'язків на краях області збіжності

- Недоліком такої модифікації методу Пікара є те, що вона збіжна лише у певному околі точки x0 (адже ряд Тейлора дає тільки локальне наближення)
- Ми пропонуємо використовувати спряження розв'язків на краях цього околу збіжності, щоб обійти цю проблему.

15

Процедура спряження розв'язків на краях області збіжності

- Для цього, після знаходження розв'язку у початковій області збіжності, розкладаємо отриманий розв'язок у ряд Тейлора у крайній точці цієї області та застосовуємо до цього розкладення метод Пікара, що дасть нам наближення розв'язку у певному околі цієї крайньої точки. Продовжуємо так стільки, скільки необхідно.

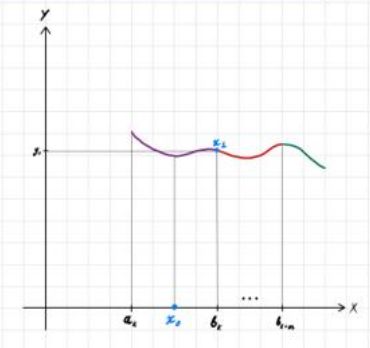


Рисунок 2 – схема роботи алгоритму для неполіноміальних функцій

16

Прийом для подальшого спрощення розрахунків на ЕОМ

- Ми пропонуємо спеціальний прийом для спрощення розрахунків та економії оперативної пам'яті при застосуванні цього алгоритму на ЕОМ.
- Приймаючи до уваги те, як працюють сучасні ЕОМ, при практичних розрахунках, не завжди зручно зберігати у пам'яті усі попередні наближення, які ми знайшли. Дійсно, для знаходження наступного наближення розв'язку системи, при реалізації цього алгоритму на ЕОМ, ітеративна процедура застосовується до правих частин рівнянь не одночасно, а «по порядку», тобто спочатку знаходимо наступне наближення $\varphi_1{}^{(l+1)}(x)$, далі $\varphi_2{}^{(l+1)}(x)$ і так далі аж до $\varphi_m{}^{(l+1)}(x)$.

17

Прийом для подальшого спрощення розрахунків на ЕОМ

- Прийом опирається на ту ж ідею, що й метод Гауса-Зейделя, і полягає у тому, що так як для знаходження наступного наближення використовуються попередні наближення розв'язків, при знаходженні $\varphi_i{}^{(l+1)}(x)$ ми можемо використати уже знайдені наближення $\varphi_1{}^{(l+1)}(x), \dots, \varphi_{i-1}{}^{(l+1)}(x)$, замість $\varphi_1{}^{(l)}(x), \dots, \varphi_{i-1}{}^{(l)}(x)$. Це може трохи пришвидшити збіжність алгоритму

18

Опис розробленої системи

- Розроблена система представлена у вигляді веб-орієнтованого застосунку, розміщеного на безкоштовному хостингу pythonenywhere. Користувач взаємодіє з даним застосунком за допомогою графічного інтерфейсу головної веб-сторінки застосунку, вводячи дані, як кількість рівнянь у системі, самі рівняння, початкові умови тощо у відповідні поля.

19

Опис розробленої системи

- Користувач має змогу розв'язати введену систему диференціальних рівнянь або модифікованим методом Пікара, що був описаний в деталях у розділі 3 магістерської дисертації, або методом Рунге-Кутта четвертого порядку точності (або, звичайно, обома методами для порівняння).
- Спрощена загальна схема функціонування системи наведена на наступному слайді.

20

Рисунок 3 – загальна схема роботи системи

Експериментальні розрахунки

- Наведемо конкретний приклад роботи системи.
- Знайдемо розв'язок простої лінійної системи диференціальних рівнянь :

$$\begin{cases} y_1' = y_2 + y_3 - y_1 \\ y_2' = -y_1 - y_2 \\ y_3' = -y_1 - 3y_3 \end{cases} \quad (5.1)$$

За умов : $y_1(0) = 0, y_2(0) = 1, y_3(0) = -1 \quad (5.2)$

- Аналітичний розв'язок цієї системи легко знайти вручну і її частинний розв'язок, що задовольняє умови (5.2) – це:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = e^{-x} \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix} \quad (5.3)$$

- Доступ до системи можна отримати за наступним посиланням: http://picardsolver.pythonanywhere.com/

22

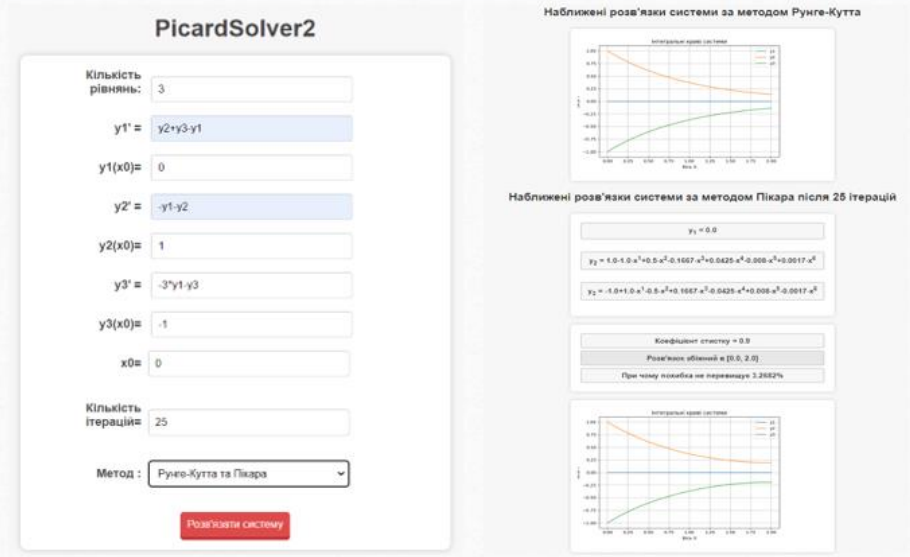


Рисунок 4 – демонстрація роботи системи

23

Експериментальні розрахунки

- Нескладно переконатися у правильності отриманого розв'язку. Дійсно, адже отриманий результат для y_2 та y_3 є нічим іншим, як записом перших декількох членів ряду Тейлора (8) для e^{-x} та $-e^{-x}$, відповідно, що є правильним аналітичним розв'язком системи.

$$e^{\tau} = \sum_{k=1}^{\infty} \frac{\tau^n}{n!} = 1 + \tau + \frac{1}{2}\tau^2 + \frac{1}{6}\tau^3 + \frac{1}{24}\tau^4 + + \frac{1}{120}\tau^5 + \frac{1}{720}\tau^6 + \cdots =$$

$$= 1 + 1 \cdot \tau + 0.5\tau^2 + 0.166(6)\tau^3 + 0.04166(6)\tau^4 + 0.00833(3)\tau^5 + 0.001388(8) \ldots \ (\mathbf{8})$$

- І це дійсно якраз те, що ми і отримали у відповідному полі.

24

Експериментальні розрахунки

- Далі, покажемо, як працюватиме розроблений застосунок при введені в неї нелінійної системи. Візьмемо, наприклад, таку систему :

$$\begin{cases} y_1' = y_2 + y_1 \\ y_2' = x \cdot y_1 \end{cases} \quad (\mathbf{9})$$

За умов : $y_1(0) = 1, y_2(0) = 0 \quad (\mathbf{10})$

25

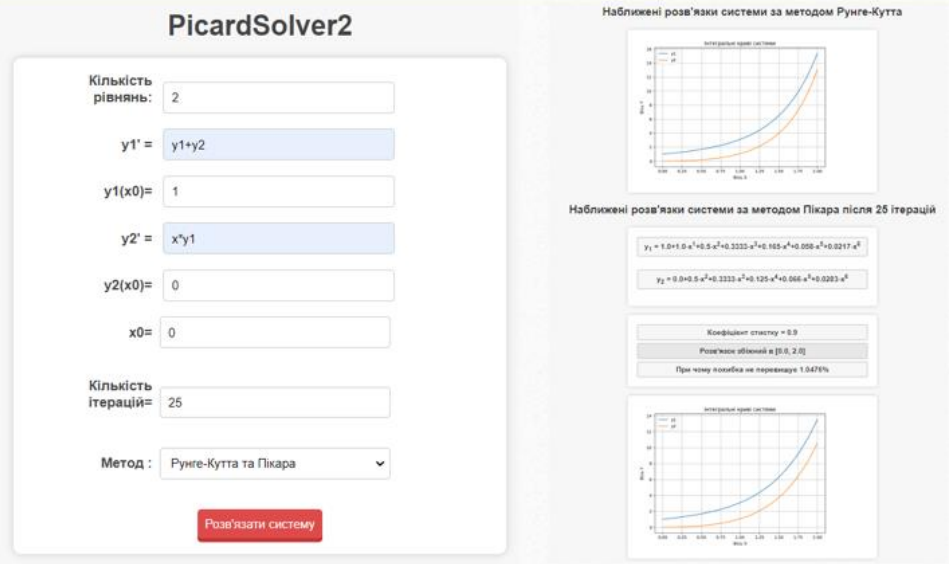


Рисунок 5 – демонстрація роботи системи

26

Експериментальні розрахунки – Система Лотки-Вольтерра

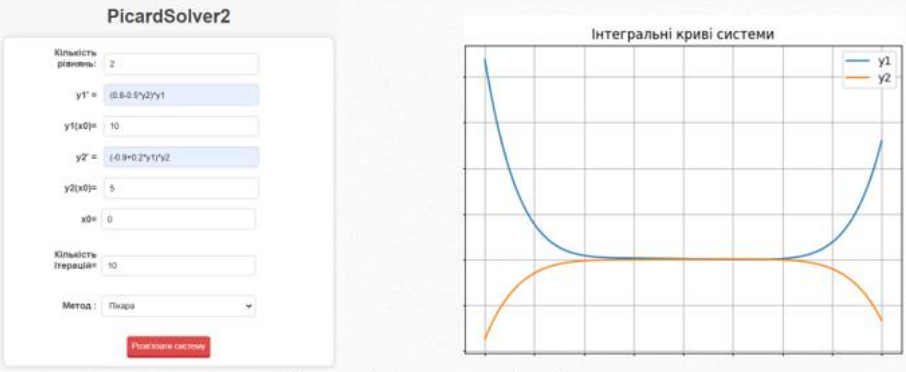


Рисунок 6 – демонстрація роботи системи

27

Висновки

- Виконано порівняльний аналіз існуючих наближених та чисельних методів для розв'язання задачі Коші для систем звичайних диференціальних рівнянь першого порядку.
- Запропоновано математичну модель модифікації методу Пікара послідовних наближень для систем звичайних диференціальних рівнянь першого порядку, для спрощення його застосування на ЕОМ. Ця модифікація полягає в заміні підінтегральних функцій систем їх рядами Тейлора та їх подальшому інтегруванні за стандартною процедурою.

28

Висновки

- Для цієї модифікації було доведено збіжність та оцінено похибку. Було також запропоновано прийом подальшого спрощення розрахунків на ЕОМ, який має схожу ідею на метод Гауса-Зейделя для СЛАР. Насамкінець, запропоновано процедуру спряження розв'язків на кінцях локальних областей збіжності для інтегрування систем диференціальних рівнянь на інтервалах більших за початкову область збіжності.
- Описану модифікацію методу було програмно реалізовано у вигляді веб-орієнтовану застосунку,
- Виконано тестування побудованої системи

29

Дякую за увагу !

30