Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

Магістерська робота

освітній ступінь – магістр

на тему: «**МОДЕЛЮВАННЯ ДЕТЕРМІНОВАНОГО ХАОСУ ЗАСОБАМИ ОБЧИСЛЮВАЛЬНОГО ЕКСПЕРИМЕНТУ**»

Виконала: студентка 2-го року навчання,

освітньо-наукової програми «Прикладна математика», 113

Герасимчук Дарія Павлівна

Керівник Авраменко О.В._____

доктор фіз.-мат. наук, професор

Рецензент____Макарчук О.П.____

(прізвище та ініціали)

Магістерська робота захищена з оцінкою _____

Секретар ЕК _____

«____» _____ 20____ р.

Київ – 20___

ЗМІСТ

ВСТУП

Сучасні дослідження об’єктів та процесів різних явищ пов’язані з їхньої нелінійною динамікою та явищем детермінованого хаосу, існування якого обмежує можливість моделювання складних процесів. Системами, для яких використовуються моделі нерегулярної поведінки, наприклад, є:

- різні сегменти фінансово-економічних процесів цінової динаміки мікро- та макроекономічних показників;
- вивчення режимів динамічної взаємодії існуючих реальних динамічних систем із врахуванням всіх ефектів;
- моделювання метеорологічних явищ і процесів тощо.

Складність дослідження нелінійних математичних моделей динамічних систем, які, як правило, не мають точних аналітичних розв’язків, зумовлюється тим, що надзвичайно мала похибка задання початкового стану повністю детермінованої системи призводить до непередбачуваних наслідків її стану на великих проміжках часу. Результатом такої поведінки системи є те, що вона на перший погляд, здається, що характеризується нерегулярним, хаотичним розвитком динамічних змінних в часі, хоча сама динаміка хаотичного режиму системи є детермінованою, і в ній можна встановити ряд закономірностей та властивостей, що відрізняють її від класичних випадкових процесів. Така хаотична система має фрактальну розмірність та проявляє чутливу залежність від початкових умов. Наприклад, довготермінове прогнозування погоди стає неможливим не тому, що математичні моделі, які при цьому використовуються, обмежені, а через неймовірну чутливість цих систем до найменших похибок, які згодом приводять до неправильних прогнозів.

В основу теорії хаосу лягли, серед інших, роботи Анрі Пуанкаре, Едварда Лоренца, Бенуа Мандельброта, Бориса Чирікова, Якова Синая та Мітчела Фейгенбаума. Коротко розглянувши капсулу історії динамічних систем, та

динаміки в цілому, можна виділити основних науковців та їхній вклад в розвиток. Ньютон (1666) запропонував поняття математичного аналізу, пояснив планетарний рух, в цей період відбувся розквіт математичного аналізу та класичної механіки. Пізніше, в 1800 тих роках, з’явились аналітичні дослідження планетарного руху. Пуанкаре (1890-ті) описав геометричний підхід та кошмари хаосу. Протягом 1920-1950 років з’явились поняття “нелінійні осцилятори” в фізиці та інженерії, також винахід радіо, радару, та лазера. Протягом 1920-1960 років Біркгоф, Колмогоров, Арнольд та Мозер працювали над явищем складної поведінки в гамільтоновській механіці. Лоренц (1963) дослідив дивний атрактор в простій моделі конвекції. Вже в 1970 тих роках з’явились такі дослідження як: турбуленція і хаос за теорією Рюеля та Такенса; хаос в логістичному відображенні за теорією Мея; універсальність Фейгенбаума та ренормалізація, зв'язок між хаосом і фазовими переходами; експериментальні дослідження хаосу; нелінійні осцилятори у біології за теорією Вінфрі; фрактали Мандельброта. Починаючи з 1980 тих років з’явився широкий інтерес до хаосу, фракталів, осциляторів та їх застосувань на практиці [5].

Актуальність теми: останні роки відбувається стрімке зростання інтересу до пошуків нових моделей нерегулярної поведінки в складних системах різної природи, які могли б давати більш точні та довготривалі прогнози для явищ, які описуються динамічними системами з явищем детермінованого хаосу. На практиці такі явища можуть виникати в різних сферах, як було вже приведено вище, в сегментах фінансово-економічних процесів цінової динаміки мікро- та макроекономічних показників; при спостереженні за експериментальними часовими рядами (присутня хаотична поведінка) для задач в різних галузях природознавства, техніки та світової економіки. Застосування комп’ютерних потужностей для реконструкції, моделювання та дослідження (тобто проведення обчислювального експерименту) різних динамічних систем вже у другій половині XX ст. стало основним інструментом для розв'язання даної задачі.

Таким чином, удосконалення математичного моделювання та розроблення нових математичних моделей для дослідження регулярних і хаотичних режимів поведінки нелінійних систем в залежності від біфуркаційних параметрів (розв’язання прямих задач) є актуальним.

Мета дослідження: експериментально перевірити властивості динамічних систем з детермінованим хаосом комп’ютерними засобами.

Завдання дослідження:

- на основі аналізу динамічних систем з детермінованим хаосом та відомостей про обчислювальні експерименти виділити серію експериментів для подальшого дослідження;
- виконати аналіз алгоритмів для обчислювальних експериментів та обрати засоби та програмне середовище для їх реалізації;
- розробити програмний продукт для реалізації експериментів та візуалізації їх результатів та провести експерименти для виявлення властивостей динамічних систем;
- виконати аналіз результатів експериментів, підтвердити чи спростувати властивості динамічних систем, що розглядаються.

Об’єкт дослідження: динаміка систем з детермінованим хаосом.

Методи дослідження: обчислювальний експеримент, мова програмування Python у програмному середовищі Pycharm.

Наукова новизна одержаних результатів: Автоматизовано процес пошуку вікон на орбітній діаграмі, експериментально підтверджено принцип дихотомії на основі взаємозв’язків між множинами Жюліа та Мандельброта та деякі інші властивості динамічних систем з детермінованим хаосом.

Практичне значення одержаних результатів: Розроблені програмні продукти можна використовувати для аналізу динамічних систем та візуалізації процесів.

Апробація результатів магістерської: матеріали квафілікаційної роботи доповідались на науковій конференції XII Всеукраїнська наукова конференція молодих математиків 9 травня 2024 року [1].

Структура роботи.

Робота складається зі вступу, двох розділів, висновків, списку використаної літератури та додатків.

По першому розділі представлено основні відомості теорії динамічних систем, огляд на різні засоби проведення обчислювального експерименту та на алгоритми їх реалізації.

По другому розділі сформульована постановка задачі та представлені результати експериментів.

РОЗДІЛ 1. Основні відомості про обчислювальні експерименти у динамічних системах

В першому розділі даної роботи розглядаються основні відомості з теорії хаоса та динамічних систем, які будуть застосовуватись для реалізації алгоритмів. Оглянуто різні засоби для проведення експерименту, включаючи огляд сайтів, різних програм чи середовищ як Wolfram Mathematica та Maple та можливості пакетів. Описано різні алгоритми для реалізації обчислювальних експериментів. У цьому розділі використовуються такі літературні джерела [2], [3], [4], [6], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17].

1.1. Основні відомості теорії динамічних систем

В першу чергу, перед визначенням поняття “детермінований хаос” та ознайомленням з теорією хаосу, варто вказати, що мається на увазі під динамічними системами, які будуть розглядатися в даній роботі. Прості математичні операції, як от піднесення до квадрату, взяття кореня квадратного чи кубічного, будуть представляти собою динамічні системи, якщо розглядати їх як повторювані ітераційні операції, таким чином результат попередньої операції - вхідні дані для наступної ітерації. Тепер вже розглядаючи список отриманих значень, можна деяким чином проаналізувати поведінку системи: буває так, що вона повністю передбачувана, в іншому ж випадку - хаотична. Далі буде розглянуто детальніше кожен такий випадок.

Точного визначення поняття хаосу ще немає, оскільки його можна тлумачити по різному. Одне з запропонованих означень формулюється наступним чином:

Хаос - це аперіодична довготривала (тобто навіть при $t \to \infty$ періодичні траєкторії не існує) поведінка в детерміністичний (система не має ніяких стохастичних характеристик, нерегулярна поведінка відбувається за рахунок

нелінійності системи) системі, яка має властивість чуттєвої залежності від початкових умов (близькі траєкторії віддаляються експоненційно швидко) [2].

Іншим можливим означенням хаосу може бути наступне:

Динамічна система F хаотична, якщо:

1) Періодичні точки для F є всюди щільними,
2) F є транзитивною,
3) F чутливо залежить від початкових умов.

Отримане означення хаосу, яке було сформульоване в [3], буде основним в даній роботі, тож варто одразу ввести означення орбіти, періодичної точки, щільності множини, транзитивності F та чутливості від початкових умов.

Визначення орбіти. Орбіта - це послідовність точок, яка утворюється при застосуванні функції F до вихідної точки x_0 та її наступних ітерацій. Іншими словами, орбіта - це множина всіх точок, які отримуються, здійснюючи послідовні ітерації функції F від початкової точки x0.

Визначення періодичної точки. Періодичною точкою періоду n будемо називати таку точку x_0, для якої виконуватиметься рівність $F^n(x_0) = x_0$, де $n > 0$. В $F^n(x_0)$ n позначається не степінь функції, а кількість ітерацій її виконання на точку. Тоді, фіксованим називатиметься таке x_0, для якого $F(x_0) = x_0$, або, інакше кажучи, $F^n(x_0) = x_0$ для будь якого $n > 0$.

Визначення атрактора та репелера. Нехай x_0 - фіксована точка для F. Тоді x_0 є атрактором, якщо $|F'(x_0)|<1$. Точка x_0 - репелер, якщо $|F'(x_0)|>1$. Нарешті, якщо $|F'(x_0)|=1$, фіксована точка називається нейтральною [3]. З цього можна сформулювати, що періодична точка періоду n є атрактором (чи репейлером), якщо атрактором (чи репелером) є фіксована точка для F^n. Перевірку для періодичної точки можна здійснювати по-різному, але одним з найзручніших методів є знаходження похідної F та добуток її значень для $x_0, x_1, \ldots, x_{n-1}$.

Щодо особливих точок динамічних систем на площині, то можна виділити такі типи: вузол стійкий та нестійкий, сідло, центр, а також стійкий та нестійкий

фокус. Стійкість та нестійкість характеризують поведінку в точці, подібно до атрактору та репейтеру відповідно.

Варто одразу зазначити, що однієї з важливих відмінностей між фіксованими точками типу атрактору та репейлеру є те, що перші “видимі” при моделюванні на комп’ютері, тоді як інші - зазвичай ні, оскільки їх важко відстежувати. Тому в даній роботі шукатимуться та досліджуватимуться саме атрактуючі фіксовані та періодичні точки.

Визначення суперстійкої точки. Припустимо, що x_0 є критичною точкою для F, тобто $F'(x_0) = 0$. Якщо x_0 також є періодичною точкою F з періодом n, тоді орбіта x_0 називається супер стійкою. Причина полягає в тому, що $(F^n)'(x_0) = 0$ [3].

Визначення щільності. Нехай X — множина, а Y — підмножина X. Y є усюди щільною в X, якщо для будь-якої точки $x \in X$ існує точка y з підмножини Y, яка є як завгодно близькою до x. Наприклад, підмножина раціональних чисел є всюди щільною в множині дійсних чисел.

Визначення транзитивності F. Динамічна система F є транзитивною, якщо для будь-яких двох непорожніх відкритих множин U і V в просторі, існує таке натуральне число n, що $F^n(U) \cap V \neq$ порожня множина. Іншими словами, це означає, що через деяку кількість ітерацій образ однієї довільної відкритої множини під дією F перетинається з іншою довільною відкритою множиною. Транзитивність свідчить про те, що система має високу ступінь змішування орбіт, тобто що для будь-яких двох точок можна знайти орбіту, яка наближається як завгодно близько до обох.

Визначення чутливості від початкових умов. Динамічна система F чутливо залежить від початкових умов, якщо існує $\beta>0$ таке, що для будь-якої точки x і будь-якого $\epsilon>0$ знайдеться точка y яка знаходиться на відстані не більше ϵ від x, і таке число k, що відстань між $F^k(x)$ і $F^k(y)$ буде принаймні β [3].

Існують два основні типи динамічних систем: диференціальні рівняння(differential equations) та ітеровані відображення(iterated maps - також відомі як рівняння різниць). Диференціальні рівняння описують еволюцію систем в неперервному часі, тоді як ітеровані відображення виникають у задачах, де час дискретний. Диференціальні рівняння використовуються набагато ширше в науці та інженерії. Їх неможливо розв'язати явно, тому використовуються числові методи, які, частіше за все, є ітеративними процесами, до прикладу метод Рунге-Кутта. Найпростішою неперервною системою, в якій спостерігається детермінований хаос, є дивний атрактор Лоренца.

З іншої сторони, використовуючи ітеровані відображення, можна описати динаміку, тобто поведінку системи з плином часу, провести якісний аналіз, при цьому не знаючи розв'язків диференціальних рівнянь у конкретний момент.

Для систем нелінійних диференціальних рівнянь обов'язковою умовою існування детермінованого хаосу є вимога того, щоб ця система описувалась принаймні трьома динамічними змінними. У двовимірному випадку неможливо побудувати фазовий портрет системи, в якому фазові траєкторії не перетиналися б (вимога детермінізму) й існував хаос. У системах із дискретним часом такої вимоги не існує, тож хаос, наприклад, може виникнути і в одновимірній задачі (до прикладу задача про відображення повернення).

Окрім як вже розглянутих вище термінів, варто зазначити і решту понять, ідея яких буде реалізована в експериментах.

Розглянемо поняття біфуркації, її типи та умови виникнення. Біфуркація в широкому сенсі може означати якісну зміну системи, тоді точка біфуркації - це точка, в якій система переходить з одного стану в інший. У точці біфуркації можливі три сценарії розвитку системи: система може перейти в хаотичний стан і зруйнуватись; вона може перейти до стану рівноваги; або починає формувати нову впорядкованість [4]. Існують різні типи біфуркації в залежності від умов

виникнення та поведінки системи. Серед них для нелінійних систем можна виділити наступні :

- Сідлово-вузлова біфуркація (Saddle-Node Bifurcation). Розглядаючи рівняння першого порядку $f_a(x) = x^2 + a$, сідлово-вузлова біфуркація може утворюватись при зміні параметру a. Так при $a > 0$ рівняння не має точок рівноваги, при $a = 0$ з'являється одна точка рівноваги $x = 0$, а вже при $a < 0$ утворюються дві точки рівноваги, при чому одна з них вузол (стабільна), інша - сідло (нестабільна).
- Вилкоподібна біфуркація (Pitchfork Bifurcation). Названа була через форму біфуркаційної діаграми, яка нагадує форму вилки. Схожа по принципу до сідлово-вузлової біфуркації, але в даному випадку може виникати одразу три точки рівноваги. Так і виходить, що при деякому параметру система має три рівноважні точки: одну нестабільну та дві стабільні, при іншому - єдина рівноважна точка, яка змінює стабільність, в третьому ж випадку - єдина рівноважна точка, яка стабільність не змінює.
- Біфуркація Хопфа (Hopf Bifurcation). Система при такій біфуркації не утворює нових точок рівноваги, замість цього виникає новий періодичний розв'язок(тобто орбіта) при проходженні параметра через деяке критичне значення. Так для системи в полярних координатах $r' = ar - r^3, \theta' = 1$ при $a < 0$ всі рішення прямують до початку координат, який є стоком; при $a = 0$ початок координат є рівноважною точкою, але особливих змін у поведінці системи немає; при $a > 0$ початок координат стає джерелом, і з'являється новий стабільний періодичний розв'язок (орбіта радіусом $\sqrt{a}$). Всі розв'язки прямують до цієї орбіти з часом [6].
- Біфуркація подвоєння періоду (Period-Doubling Bifurcation).

Визначення. Сім'я кривих $F_\lambda = \lambda x(1-x)$ зазнає біфуркації подвоєння періоду при значенні параметру $\lambda = \lambda_0$ якщо існує відкритий інтервал I та $\epsilon > 0$, для яких виконується:

1) Для кожного λ в інтервалі $[\lambda_0 - \epsilon, \lambda_0 + \epsilon]$, існує унікальна фіксована точка p_λ для F_λ в I.
2) Для $\lambda_0 - \epsilon < \lambda \leq \lambda_0$, F_λ не має циклів періоду 2 в I та p_λ є атрактором(відповідно репелером).
3) Для $\lambda_0 < \lambda < \lambda_0 + \epsilon$, з'являється єдиний 2-періодичний цикл $q^1{}_\lambda, q^2{}_\lambda$ в I з $F_\lambda(q^1{}_\lambda) = q^2{}_\lambda$. Цей 2-цикл є атрактором (відповідно репелером), тоді як фіксована точка q_λ стає відштовхуючою (відповідно притюгуючою).
4) Коли $\lambda \to \lambda_0$, то $q^i{}_\lambda \to p_{\lambda_0}$ [3].

- Комплексні біфуркації (Complex Bifurcations). Для дійсних функцій зазвичай утворюються сідлово-вузлова біфуркація та біфуркація подвоєння періоду, тоді як для функцій на комплексній площині можуть виникати біфуркації періоду q (period-q bifurcation) [3].

Розглядатимуться також такі поняття як біфуркаційна та орбітна діаграми. Біфуркаційна діаграма — це графічне зображення поведінки динамічної системи в залежності від зміни параметра системи. Вона показує, як змінюються рівноважні точки, цикли та інші динамічні режими системи при варіації параметра. На біфуркаційний діаграмі зображені як атракторна, так і репелерна фіксовані точки, тоді як на орбітній - лише фіксована точка - атрактор.

Визначення. Вікно періоду n - це проміжок значень параметра системи, при яких спостерігається динаміка з періодом n. Для системи з параметром c, яка може

мати динаміку з різними періодами, вікно періоду n - це діапазон значень c, де виявляється стабільна динаміка з цим періодом.

Варто згадати також порядок Шарковського, який представляє собою деякий порядок натуральних чисел і пов'язаний з дослідженням періодичних точок динамічних систем на відрізку. Сама ж теорема Шарковського сформульована наступним чином:

Якщо неперервне відображення одновимірного інтервалу в себе має цикл періоду m, то воно має також і цикли всіх періодів m', які передують числу m у переліку всіх цілих чисел, записаних в порядку Шарковського:

3, 5, 7, 9, . . .

$2 \cdot 3, 2 \cdot 5, 2 \cdot 7, \ldots$

$2^2 \cdot 3, 2^2 \cdot 5, 2^2 \cdot 7, \ldots$

$2^3 \cdot 3, 2^3 \cdot 5, 2^3 \cdot 7, \cdots$

. . .

. . . , 2^n, . . . , 2^3, 2^2, 2^1, 1 [4].

Наступним, що буде використовуватись для проведення експериментів це поняття ітераційної системи функцій та фрактальна розмірність.

Означення. Набір деяких функцій $\{A_1, \ldots, A_n\}$ називається ітерованою системою функцій, якщо для фіксованих точок $p_1, \ldots, p_n$, кута та параметру $0 < \beta < 1$, функції мають вигляд

$A\begin{pmatrix} x \\ y \end{pmatrix} = \beta \cdot \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix} \cdot \begin{pmatrix} x-x_0 \\ y-y_0 \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$, де $\beta < 1$ і фіксована точка в $p = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$ [3].

Фрактал будується, якщо вибирати довільну початкову точку на площині та обчислювати її орбіту для випадкової ітерації A_i достатню кількість разів. Таким чином, для множини фіксованих точок будується орбіта, де p - це точка з орбіти, а p_i - це точка з множини фіксованих точок, яка на кожній ітерації розрахунку похідної вибирається випадковим чином.

Фрактальна розмірність (fractal dimension) зазвичай обчислюється як логарифм кількості копій фрактала, які потрібні для покриття фрактала, поділений на логарифм зменшення масштабу цих копій. Вона вимірює ступінь складності геометричних фігур, які не є цілими числами.

У комплексній динаміці, коли розглядаються ітерації раціональних функцій, важливими поняттями є множини Фату, Жуліа та Мандельброта. Ось детальне пояснення цих множин:

Визначення множини Фату. Множина Фату $F(R) \in \overline{\mathbb{C}}$ для раціонального відображення R: $\overline{\mathbb{C}} \to \overline{\mathbb{C}}$ — це множина точок $z \in \overline{\mathbb{C}}$, таких що сімейство ітерацій $\left\{R^n\right\}_{n\in N}$ є нормальним у z.

Нормальність сімейства функцій означає, що будь-яка послідовність функцій із цього сімейства має підпослідовність, яка рівномірно збігається на компактах. Точки з множини Фату поводяться регулярно, їхні орбіти не демонструють хаотичної поведінки.

Визначення множини Жюліа. Множина Жюліа $J(R)$ — це доповнення до множини Фату, тобто $J(R) = \overline{\mathbb{C}} \backslash F(R)$. Множина Жюліа містить точки, для яких ітерації раціональної функції демонструють хаотичну поведінку. Вона є місцем найбільшої складності та нестабільності динамічної системи. Обидві множини $F(R)$ та $J(R)$ є повністю інваріантними відносно R.

Визначення множини Мандельброта. Розглянемо сім'ю функцій $Q_c(z) = z^2 + c, c \in \mathbb{C}$ - вони відповідають класам кон'югації квадратичних відображень. Це означає, що кожне значення параметра c визначає унікальну динамічну поведінку для відповідної функції. За одним з означень множина Мандельброта - це множина значень c, для яких орбіта точки 0 (розглядається саме вона, оскільки ця точка є єдиною критичною точкою для даної сім'ї функцій, а отже може описати всі можливі орбіти для різних параметрів c) під дією Q_c є обмеженою.

За теоремою Дюаді - Хаббарда (DH82), множина Мандельброта $M = \left\{c \in \mathbb{C}: \left|Q_c^n(0) \leq 2 \text{ для всіх } n \in N\right|\right\}$, M є замкнутою і простозв'язною. Іншими словами, для кожного c, який належить M, ітерації точки 0 не перевищують модуль 2 [7]. Множина Мандельброта є "словником" всіх множин Жюліа.

Принцип фундаментальної дихотомії.

Нехай $Q_c(z) = z^2 + c$. Тоді або

1. Орбіта критичної точки 0 прямує в нескінченність, в цьому випадку K_c (заповнена множина Жюліа) складається з безлічі компонент, що не перетинаються (disjoint components), або
2. Орбіта критичної точки 0 залишається обмеженою, в такому випадку K_c - зв'язна множина.

Введемо ще одне поняття - p/q бульба або $B_{p/q}$, де q - період бульби, p - деякий нумератор. На множині Мандельброта можна спостерігати явище q-періодичної біфуркації для різних значень параметру c. Так для значень c всередині основної кардіоди, множини Жюліа матимуть одну фіксовану точку-атрактор, для значень c всередині найбільшої бульби (має вигляд кола) матиме місце притягуючий цикл періоду 2. Відомо розташування та порядок усіх бульб [8], і зокрема, що розташування бульб $B_{p/q}$ на границі основної кардіоїди у визначеному порядку, відповідає раціональним числам між 0 і 1.

В даній роботі розглядатимуться дискретні відображення по типу сім'ї квадратичних функцій та логістичних функцій вигляду $F_\lambda(z) = \lambda \cdot z \cdot (1 - z)$, які також називають квадратичним та логістичним відображенням відповідно.

1.2. Огляд засобів проведення обчислювального експерименту

Експерименти, описані в [3], були реалізовані студентом автора Себастьяном Маротта в Wolfram Mathematica та представлені на [9]. Wolfram Mathematica - це єдина інтегрована система комп'ютерної алгебри, розроблена компанією Wolfram Research. Вона охоплює широту та глибину технічних обчислень, і легко доступна в хмарі через будь-який веб-браузер, а також у всіх сучасних настільних системах. Mathematica має понад 6 000 вбудованих функцій, що охоплюють усі області технічних обчислень, спираючись на три десятиліття розвитку, Mathematica досягає успіху у всіх областях технічних обчислень, включаючи нейронні мережі, машинне навчання, обробку зображень, геометрію, науку про дані, візуалізацію та багато іншого. Використовується Wolfram Notebook Interface, який дозволяє організовувати все у розширених документах, які включають текст, виконуваний код, динамічну графіку, інтерфейси користувача тощо [10]. Однак, варто зазначити, що це комерційне програмне забезпечення, яке може бути досить дорогим, особливо для індивідуальних користувачів або невеликих організацій. Також, хоча Mathematica має дуже потужний набір вбудованих функцій для аналізу та візуалізації даних, вона може бути менш гнучкою в порівнянні з обраною для проведення експериментів, мовою програмування Python при необхідності інтеграції з іншими системами або спеціальними бібліотеками; у зв'язку зі своїм унікальним синтаксисом, є досить специфічною на відміну від простого та зрозумілого синтаксису Python.

Наступним пакетом, що розглядається, буде Maple, розроблений Waterloo Maple. Він є популярним серед широкого кола дослідників з різних дисциплін. Це символьний, числовий та графічний пакет, який робить його зручним для вивчення нелінійних динамічних систем [11]. Хоча Maple може бути корисним для символьних обчислень і інтерактивної роботи, а також здатен інтегруватися з іншими програмами та мовами програмування, такими як MATLAB та R, Python надає більше гнучкості, доступності ресурсів і можливостей для інтеграції з іншими сучасними технологіями.

Серед розглянутих в даній роботі експериментів, варто зазначити експеримент побудови множин Мандельброта та Жюліа. Була використана ідея інтерфейсу, схожа до Javascript Julia Set Generator [12]. Запропонована реалізація дозволяє співставляти множину Жюліа до області на множині Мандельброта, в якій був обраний параметр c, при чому координати точки зберігаються. Присутня можливість побудови орбіти на множині Жюліа, а також перемикання між звичайною множиною та заповненою. Власна реалізація була дещо модифікована, дозволяючи збільшувати будь яку область на множинах, також є можливість задання довільної функції, побудова кольорового зображення та ідентифікація бульб.

Python був обраний у якості основної мови програмування для проведення експериментів, оскільки має багатий набір бібліотек для числових обчислень (NumPy, SciPy), обробки даних (pandas), та візуалізації (Matplotlib, Seaborn). Також є спеціалізовані бібліотеки для аналізу динамічних систем і хаосу, такі як Chaospy та PyDSTool, хоча вони і не були використані, оскільки були застосовані загальні алгоритми для побудови орбіт, множин тощо. Python також підтримує створення графічних інтерфейсів користувача за допомогою бібліотек, таких як Tkinter, PyQt, Dash та PySimpleGUI, що дозволяє інтерактивно маніпулювати параметрами моделей та візуалізувати результати в реальному часі. Останній використовувався в даній роботі. Необмежена можливість різних модифікацій, надбудов а також комбінацій з іншими середовищами стала фактором для вибору цієї мови програмування.

NumPy (Numerical Python) — це відкрита бібліотека для Python, яка використовується майже в кожній галузі науки і техніки. Вона є універсальним стандартом для роботи з числовими даними в Python і є основою наукової екосистеми Python і PyData. Бібліотека NumPy містить багатовимірні масиви та структури даних матриць. Вона надає ndarray, однорідний n-вимірний об'єкт

масиву, з методами для ефективної роботи з ним [13]. В даній роботі використовуватиметься за вищезазначеним признанням.

SciPy — це колекція математичних алгоритмів і зручних функцій, побудована на основі розширення NumPy для Python. Використання SciPy у поєднанні з NumPy надає додаткові можливості для моделювання динамічних систем і детермінованого хаосу. В даній роботі знаходження розв'язків рівнянь, знаходження фіксованих точок буде здійснюватись за рахунок математичних алгоритмів, запропонованих в цій колекції.

Matplotlib та PySimpleGUI є важливими інструментами в екосистемі Python, які були використані для візуалізації усіх результатів та створення графічних інтерфейсів користувача відповідно. Matplotlib є однією з найпопулярніших бібліотек для створення статичних, анімаційних та інтерактивних візуалізацій, тому була обрана як основний інструмент для візуалізації. PySimpleGUI був обраний для даної роботи, оскільки має простий та інтуїтивно зрозумілий API для створення графічних інтерфейсів користувача, дозволяє легко інтегрувати графіки Matplotlib безпосередньо в інтерфейси користувача незважаючи на те, що має деякі обмеження в продуктивності і функціональності порівняно з більш спеціалізованими GUI-бібліотеками [14].

1.3. Алгоритми реалізації обчислювальних експериментів

В цьому пункті представлені основні алгоритми реалізації обчислювальних експериментів.

1.3.1. Розрахунок орбіти, пошук фіксованої точки, критичної точки, суперстійкої орбіти та метод Ньютона-Рафсона

Розглянемо детальний алгоритм знаходження орбіти точки для деякої заданої функції $f(r,\ x)$.

Спочатку визначаються параметри для проведення обчислень. Обираються початкові значення параметра r та початкова точка x, а також встановлюється кількість ітерацій для обчислення орбіти. Потім відбувається ітераційний процес. Початкове значення x встановлюється на початкову точку, після чого за допомогою функції відображення $x_{n+1} = f(r, x_n)$ обчислюється наступне значення x. Отримані значення x зберігаються в масиві для подальшого аналізу.

Після завершення ітерацій проводиться аналіз результатів. Визначається тип орбіти на основі отриманих значень x. Якщо значення x наступної ітерації співпадає з попередньою, це означає, що точка фіксована. Якщо значення x повторюється, це вказує на наявність циклу. Якщо орбіта росте до нескінченності або має складний характер, тип орбіти може бути не визначений.

Пошук фіксованих точок для довільної функції $F(x)$ відбувається за рахунок розв'язання рівняння $F(x) = x$, а характер цієї точки, тобто вона атрактор чи репелер можна визначити аналітично та графічно. Для аналітичного методу потрібно знайти похідну функції і отримати модуль значення у фіксованій точці: якщо значення = 1, тоді точка нейтральна, якщо <1, тоді атрактор, якщо >1 - репелер. Графічно визначити чи є точка атрактором можна, якщо побудувати орбіту довільної точки з околу і вона буде прямувати до фіксованої. Критична точка шукається шляхом розв'язку рівняння $F'(x) = 0$. Супер стійкою орбітою точки x_0 називається орбіта, якщо x_0 - критична точка періоду n, тобто $F_n'(x) = 0$.

Для знаходження похідної n порядку для орбіти не обов'язково шукати похідні порядку i, для i = 1, …, n, оскільки якщо відомі значення орбіти до n-го елементу, похідна n-го порядку дорівнюватиме добутку похідних в кожній точці орбіти.

Метод Ньютона застосовується для знаходження розв'язку рівняння вигляду $F(x) = 0$, використовуючи ітераційну систему функцій Ньютона $N(x)$, що представлена формулою:

$$N(x) = x - \frac{F(x)}{F'(x)}.$$

Фіксовані точки для функції $N(x)$, які є атракторами, і будуть розв'язками рівняння $F(x) = 0$. Тому достатньо розв'язати $N(x) = x$ та дослідити характер отриманих фіксованих точок.

1.3.2. Алгоритм побудови орбітної діаграми

Визначається функція відображення, яка відображає точку *x* на наступну точку у відповідності з певним параметром *c*. Розглянемо побудову для квадратичної функції відображення $x^2 + c$.

Ітераційний процес починається з того, що обирається діапазон значень параметра c, для якого буде будуватися орбітна діаграма. Для кожного значення параметра вказаного діапазону: вибирається початкове значення *x*, яка є критичною точкою (0 для функції, що розглядається); проводяться ітерації за функцією відображення для обчислення орбіти точки *x*; отримані значення (*c*, *x*) зберігаються в масиві.

При побудові діаграми отримані значення (*c*, *x*) використовуються для побудови точок на площині (*c*, *x*). Кожна точка на діаграмі відображається як чорна точка. Зазвичай на орбітній діаграмі відображаються тільки певна кількість значень, щоб уникнути перевантаження графіка.

1.3.3. Метод пошуку вікон на орбітній діаграмі

Для знаходження вікон періодів n, в першу чергу потрібно побудувати орбітну діаграму для функцій:

$$Q_c(x) = x^2 + c, \text{ при } -2 \le c \le 0.25, -2 \le x \le 2$$

та

$$F_\lambda(x) = \lambda x(1 - x), \text{ при } 0 \le \lambda \le 4,\ 0 \le x \le 1$$

Кожне вікно періоду n (period-n window) характеризується тим, що в ньому з’являється притягуючий цикл періоду n (attracting n-cycle), після якого відбувається явище подвоєння періоду подальших циклів (period-doubling bifurcation), з чого випливає, що послідовність періодів вікон можна отримати ітеративно.

Один з способів реалізації наступний: потрібно спочатку проаналізувати основну орбітну діаграму, по ній знайти декілька найбільших вікон, потім збільшити область між цими вікнами і таким чином повторювати процес.

Інший спосіб був реалізований за допомогою комп’ютерних обчислень. Алгоритм визначає вікна, потім збільшує ділянку між ними і продовжує процес задану кількість ітерацій. Варто зазначити, що присутнє також графічне представлення, за допомогою якого користувач сам може перевіряти роботу програми.

В алгоритмі, для кожної розглянутої точки на діаграмі визначається її тип після побудови для неї орбіти. До типів належать: фіксована точка, зрештою фіксована, цикл та невизначена. Для всіх, крім типу невизначена, формується список всіх c на проміжку, де ця точка з'являється. Далі відбувається процес пошуку подвоєних періодів, і в список записуються лише цикли найменших значень. Наприклад, з послідовності [1, 2, 4] залишиться 1 що і є вікном періоду 1. Таким чином утворюється список з періодів для вікон.

Варто зазначити, що в алгоритмі розглядається значення c для циклу періоду $n \cdot 2^k$ n-періодичного вікна, де k - це довільне натуральне число. Це дозволяє відшукати невеликі вікна, які в іншому випадку можуть бути пропущені. Знаходження нових вікон залежить також від кількості ітерацій при обчисленні орбіти, оскільки потрібно ідентифікувати не лише вікно періоду n, а також наступну точку періоду $2n$. Тобто, при кожній наступній ітерації розглядаються ділянки між сусідніми вікнами попарно.

1.3.4. Алгоритм знаходження константи Фейгенбаума (Feigenbaum's Constant)

Алгоритм знаходження константи Фейгенбаума полягає в наступному: спочатку потрібно визначити всі значення параметрів c, для яких функція має суперстійкі точки періоду 1, 2, 4, 8, 16, 32 та 64 - всі потрібні знаходяться в 1-періодичному вікні. Точність для c дорівнює 10^{-6}, тобто до 7 значень після коми. Після цього для кожного значення c порахувати 2^n точок орбіти критичної точки(для функції $Q_c(x)$ це 0). Тоді повторно змінювати параметр, намагаючись наблизитися до значення, для якого критична точка є періодичною з правильним періодом. Після знаходження усіх потрібних значень c порахувати відношення, для знаходження константи Фейгенбаума, до якого мають сходитись розв'язки цих відношень. Отже, після знаходження параметрів c_n для 2^n при n = 0, …, 6 та розрахунку наступних відношень:

$$f_0 = \frac{c_0 - c_1}{c_1 - c_2},\ f_1 = \frac{c_1 - c_2}{c_2 - c_3},\ \dots, f_4 = \frac{c_4 - c_5}{c_5 - c_6}$$

Отримуємо послідовність значень $f_0, f_1, \dots, f_4$, які мають збігатись до теоретичного значення Фейгенбаума $\delta = 4.669201609\dots$.

1.3.5. Ітераційна система функцій та гра "Хаос"

Ітераційна система функцій будується відповідно до означення. Обирається кількість фіксованих точок, які будуть використовуватись в трансформаціях, а також параметри β та θ. Для кожної ітерації випадковим чином вибирається одна з фіксованих точок. Застосовується трансформація, яка включає скорочення відстані до вибраної фіксованої точки з урахуванням обертання (якщо воно використовується). Отримана нова точка додається до послідовності точок (орбіти). Процес трансформацій повторюється задану кількість разів, формуючи орбіту. Всі точки орбіти та фіксовані точки відображаються на графіку.

Використовуючи отриману орбіту і параметр β, розраховується фрактальна розмірність за формулою:

$$D = \frac{log(k)}{log(M)} = \frac{log(number\ of\ pieces)}{log(magnification\ factor)} = \frac{log(\text{кількість фіксованих точок})}{log(1/\beta)}$$ [3].

Розглянемо інший метод побудови фракталів, ідея якого використовується для побудови трикутника Серпінського в грі хаосу (The Chaos Game).

Обирається початкова точка, довільне стартове положення на площині. Починаючи з цієї точки, алгоритм виглядає так: кидається, до прикладу, кольоровий кубик, потім, в залежності від того, який колір випав, точка переміщується на половину відстані до відповідної кольорової вершини. Процес повторюється, використовуючи кінцеву точку попереднього переміщення як початкову для наступного. Зазвичай перші 15 (або близько того) точок, згенерованих цим алгоритмом не відображаються, але потім починайте записується кожна точка [15].

Важливим аспектом гри Хаосу є випадковий вибір вершини. Папороть Барнслі - це ще один приклад ітерації випадкових точок, яка створює фрактальний візерунок. По правилам, обирається будь-яка точка (x, y) і повертаються нові значення x і y за наступними правилами:

- 1% часу:

  $x \to 0$

  $y \to 0.16 \cdot y$
- 85% часу:

  $x \to 0.85 \cdot x + 0.04 \cdot y$

  $y \to -0.04 \cdot x + 0.85 \cdot y + 1.6$
- 7% часу:

  $x \to 0.2 \cdot x - 0.26 \cdot y$

  $y \to 0.23 \cdot x + 0.22 \cdot y + 1.6$

- 7% часу:

  $x \to -0.15 \cdot x + 0.28 \cdot y$

  $y \to 0.26 \cdot x + 0.24 \cdot y + 0.44$

Розглянемо останній приклад - алгоритм побудови множини Мандельброта. Це множина точок (c, d), для яких збігається наступний ітераційний процес:

- $x \to x^2 - y^2 + c$
- $y \to 2xy + d$

Описані алгоритми та зображення фракталів представлені в [16].

1.3.6. Методи побудови множин Жюліа K_c та J_c

Заповнена множина Жюліа K_c та звичайна множина Жюліа J_c будуються за різними принципами.

Розглянемо алгоритм для побудови K_c. Вхідними параметрами для комп'ютерної реалізації є: діапазон значень для дійсної та уявної частин x_{min}, x_{max}, y_{min}, y_{max}; параметр c; деяка функція $q(z, c)$ (наприклад, $Q_c(x)$); порогове значення threshold (дорівнює 2 для цієї функції); максимальна кількість ітерацій; кількість точок для кожної осі. Створюється рівномірно розподілена сітка значень x та y заданому діапазоні. Кожна точка проходить перевірку на приналежність до множини. Для кожної точки z на сітці виконується ітеративний процес: $z \to q(z, c)$. Якщо $|z|$ перевищує порогове значення до досягнення максимальної кількості ітерацій, то вважається, що точка не належить множині, інакше - належить. У випадку чорно-білого зображення, білими позначаються точки, що не належать множині, а чорними - ті, що належать. Чим більша кількість ітерацій проводиться, тим деталізованіше стає зображення. Такий метод побудови краще використовувати для зв'язних множин Жюліа. Також зображення можна створити кольоровим. Чорні точки залишаються чорними, а ті, які були білими, зафарбовуються в залежності від швидкості прямування до нескінченності. Таким

чином, маємо, що червоні точки мають орбіти, які “втікають” найшвидше, а фіолетові - найповільніше.

Розглянемо алгоритм для побудови J_c. Цей алгоритм спирається на той факт, що $Q_c(x)$ гіперчутлива на J_c. Для знаходження точки цієї множини, обирається будь-яке $z \in \mathbb{C}$ і обчислюється його “зворотна орбіта”. Таким чином, кожна точка w за винятком с має дві попередні точки під дією $Q_c(x)$, а саме $\pm\sqrt{w-c}$. Щоб обчислити зворотну орбіту w, випадковим чином обирається одна з двох попередніх точок на кожному етапі. Цей алгоритм дуже точно будує множину J_c, яку коректріше використовувати, що множина Жюліа представляє собою множину Кантора. Програмно, можливість будувати множину Жюліа за цим алгоритмом реалізовано лише для функції $Q_c(x)$, оскільки вона потребує знаходження її розв’язків.

1.3.7. Методи побудови множини Мандельброта

Алгоритм для побудови множини Мандельброта подібний до першого алгоритму для множини Жюліа. Обирається деяке значення максимальної кількості ітерацій N. Для кожної точки с на сітці розраховуються перші N точок на орбіті критичної точки функції q (наприклад, $Q_c(x)$). Потім проводиться перевірка приналежності точки множині з пороговим значенням threshold (дорівнює 2 для цієї функції). Кольорове зображення будується по тому ж принципу, що і для множини Жюліа.

1.3.8. Метод знаходження бульб на основній кардіоїді

Для знаходження бульб $B_{p/q}$ на основній кардіоїді C_1 для функції $Q_c(x)$ та визначення параметрів p та q, перш за все, потрібно описати аналітично функцію для C_1. З [3] отримали:

$$c = c(\theta) = \frac{1}{2}e^{2\pi i\theta} - \frac{1}{4}e^{4\pi i\theta}.$$

Функція для бульби періоду 2 ($B_{1/2}$) має вигляд: $|c + 1| = \frac{1}{4}$

Відповідно, для знаходження положення довільної бульби, потрібно в рівняння функції для C_1 підставити значення $\frac{p}{q}$.

Значення q визначає період бульби, тоді як p є деяким нумератором, який можна визначити, проаналізувавши орбіту точки. Якщо всі точки орбіти, які утворюють певний цикл, розташовані в деякому порядку(порядку зростання чи спадання) і по черзі обходять кожен “листок” на множині Жюліа, тоді p = 1, якщо ж, наприклад, при обході листків, проходиться кожен другий, тоді p = 2. Більш детальне роз’яснення доступне за посиланням [17].

РОЗДІЛ 2. Експериментальне дослідження динамічних систем

У другому розділі даної роботи розглядаються основні реалізовані обчислювальні експерименти, наводяться результати роботи програм. Описано алгоритми, процес програмування, користувацькі можливості програмного продукту, а також проведено обчислювальні експерименти.

2.1. Постановка задачі

1. Розробити програмні продукти для реалізації обчислювальних експериментів [3]:
    - Experiment: The Computer May Lie
    - Experiment: Rates of Convergence
    - Graphical Analysis and Newton’s Method
    - Experiment: The Transition to Chaos
    - Experiment: Windows in the Orbit Diagram
    - Experiment: Feigenbaum’s Constant
    - Experiment: Find the Iterated Function Systems
    - Experiment: Filled Julia Sets and Critical Orbits
    - Experiment: Find the Julia Set
    - Experiment: Similarity of the Mandelbrot Set and Julia Sets
    - Experiment: Periods of the Bulbs on the main cardioid
    - Experiment: Periods of the other Bulbs(coloured)
2. Провести експерименти за допомогою відповідних програмних продуктів.
3. Виконати дослідження властивостей динамічних систем та сформувати висновки.

2.2. Реалізація експериментів

Для кожного експерименту буде сформульоване завдання, описана програмна реалізація, наведені результати, надані висновки та зазначені користувацькі можливості.

2.2.1. Experiment: The Computer May Lie

Завдання. Порахувати орбіти (200 ітерацій) для 10 різних точок та визначити їх тип (фіксована, періодична, зрештою періодична чи не визначена) для функцій: $F(x) = x^2 - 2;\ G(x) = x^2 + c, c <- 2; D(x) = \begin{cases} 2x, 0 \le x \le 1/2 \\ 2x-1, 1/2 \le x < 1 \end{cases}$

Для функції $D(x)$ значення x мають бути в інтервалі (0, 1), для $F(x)$ - в інтервалі (-2, 2).

Програмна реалізація: Дана програма розроблена для вивчення орбіт різних, заданих умовою експерименту, функцій з різними початковими значеннями, щоб дослідити їхню поведінку: чи є вони фіксованими точками, періодичними, чи, можливо, мають складніші патерни. Для кожної функції використовується набір початкових значень (seeds), щоб дослідити різні орбіти. Для кожного початкового значення програма ітерує функцію до 200 разів, щоб дослідити поведінку орбіт. Процес ітерації включає наступні кроки: ініціалізація початкового значення (seed); ітеративне застосування функції до поточного значення; зберігання результатів кожної ітерації в орбіті; перевірка наявності певних патернів (фіксована точка, періодична орбіта, кінцево-періодична орбіта).

Результати експерименту: Для функції $F(x)$ були обрані значення [-1.9, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 1.9]; для функції $G(x)$ $c =- 3$, а значення обрані як для першої функції, проте, варто зазначити, що кількість ітерацій була значно зменшена до 10, оскільки виникає помилка переповнення сітки значень, тобто значення отриманої функції вже на 12 ітерації стає буде великим; для функції $D(x)$ були обрані значення [3/10, 7/10, 1/8, 1/16, 1/7, 1/14, 1/11, 3/22]. Результати роботи програми представлені в Додатку А.

Висновки: Порівнюючи функції $F(x)$ та $G(x)$ можна сказати, що обидві мають фіксовані точки при $x = \pm 1$, функція $F(x)$ має зрешту періодичну (в даному випадку фіксовану) в $x = 0$. Хоч різниця параметрів c для першої та другої функцій всього лиш 1, для $F(x)$ з вигляду(та значень точок) орбіт можна зробити висновок, що вона прямує до нескінченності у вигляді павутинки і швидкість прямування мала, що дозволяє порахувати всі 200 ітерацій; для $G(x)$ швидкість віддалення від точки дуже велика, орбіта має вигляд сходинки, тому і кількість ітерацій обмежена до 10.

Для функції $D(x)$ орбіту для точок $\frac{1}{5}$та $\frac{1}{9}$ можна порахувати явно:

- для $x = \frac{1}{5}$ 4-періодичний цикл вигляду $\frac{1}{5}, \frac{2}{5}, \frac{4}{5}, \frac{3}{5}, \frac{1}{5}, \ldots$
- для $x = \frac{1}{9}$ 6-періодичний цикл вигляду $\frac{1}{9}, \frac{2}{9}, \frac{4}{9}, \frac{8}{9}, \frac{7}{9}, \frac{5}{9}, \frac{1}{9}, \ldots$

Тим не менше, використовуючи комп'ютерні обчислення, маємо вреші решт періодичну точку, що не зовсім правильно. Це лише один з прикладів, який демонструє випадки некоректного розрахунку (і як результат, класифікацію) на комп'ютері (що може пояснюватись внутрішнім представленням числа), тому експеримент і отримав таку назву.

Користувацькі можливості. Додаткових користувацьких можливостей у вигляді інтерактивного інтерфейсу дана програмна реалізація експерименту не має, тим не менше, є можливість зміни значень параметрів безпосередньо в коді програми.

2.2.2. Experiment: Rates of Convergence

Завдання. Оцінка швидкості збіжності орбіти до фіксованої точки-атрактора та порівняння цієї швидкості до значення похідної функції в фіксованій точці. Для кожної функції, яку потрібно розглянути, будується орбіта, починаючи з точки 0.2, та оцінюється (скажімо, орбіта досягає фіксовану точку з точністю 10^{-5}). Для кожної функції потрібно знайти:

a. Фіксовану точку p. Для випадку, якщо таких декілька, то потрібно визначити ті, до яких буде притягуватись орбіта.
b. $|F'(p)|$.
c. Чи є точка p атрактором чи репелером.
d. Кількість ітерацій, які необхідні орбіті x_0=0.2 досягти p.
e. Вивід останніх трьох елементів орбіти.

В другій частині програми розглянути усі функції, для яких відомо, що вони мають притягуючий цикл періодом 2. Потрібно визначити експериментально цей цикл та оцінити $\left|F^{(2)}(p)\right|$.

Розглядаються функції: $F(x) = x^2 - 1$, $F(x) = x^2 - 1.1$, $F(x) = x^2 - 1.25$

Програмна реалізація. Дана програма розроблена для аналізу орбіт функцій та їх збіжності до фіксованих точок, а також для оцінки швидкості цієї збіжності.

Програма аналізує 12 різних функцій, включаючи квадратичні функції, логістичні відображення та тригонометричні функції. Кожна з цих функцій має свої унікальні властивості. Для визначення збіжності орбіт до фіксованих точок програма обчислює похідні функцій в цих точках. Програма знаходить фіксовані точки функцій, вирішуючи рівняння $F(x) = x$ за допомогою бібліотеки sympy. Для кожної функції програма визначає, чи є фіксована точка притягальною або нейтральною, базуючись на абсолютному значенні похідної в цій точці.

Процес ітерації включає наступні кроки: ініціалізація початкового значення - для кожної функції використовується початкове значення x_0=0.2; ітеративне застосування функції - програма ітерує функцію до досягнення фіксованої точки або досягнення максимального числа ітерацій (1000); Зупинка ітерацій: Ітерації зупиняються, якщо різниця між поточним значенням орбіти і попереднім значенням стає менше порогового значення (threshold = 1×10^{-5}) або досягається максимальне число ітерацій.

Результати експерименту. Результат роботи програми представлений у Додатку Б. Швидкість збіжності до фіксованої точки пов’язана зі значенням $|F'(p)|$, так нейтральна точка, для якої $|F'(p)| = 1$, збігається найдовше до p за решту розглянутих прикладів.

Результат роботи програми для другої частини завдання представлений у Додатку В.

Висновки. Видно, що швидкість збіжності в цьому випадку залежить від значення $\left|F^{(2)}(p)\right|$, так при найменшому значенні для розглянутих функцій, швидкість найбільша.

Отже, швидкість збіжності орбіти до точки періоду n залежить від значення $\left|F^{(n)}(p)\right|$.

Користувацькі можливості. Додаткових користувацьких можливостей у вигляді інтерактивного інтерфейсу дана програмна реалізація експерименту не має, тим не менше, є можливість зміни значень параметрів безпосередньо в коді програми.

2.2.3. Graphical Analysis and Newton’s Method

Завдання. Демонстрація орбіти довільної точки для заданої функції, знаходження розв’язків рівняння $F(x) = 0$.

Програмна реалізація. Для програмної реалізації експерименту створений графічний інтерфейс, тому програма розділена на дві частини (два файли з розширенням .py), при чому в одному представлені основні розрахунки, інший для побудови інтерфейсу. Розглянемо детально перший, який є основним.

Програмне забезпечення складається з кількох основних функцій та класів, які дозволяють обчислювати і візуалізувати фіксовані точки та орбіти для даних математичних функцій. Основний акцент робиться на використанні символьних та числових методів для розв'язання рівнянь, а також на інтерактивній візуалізації процесу ітерацій. До основних функцій належать :

- find_fixed_point(expr). Ця функція використовується для знаходження фіксованих точок заданої функції (у вигляді виразу expr). Спочатку використовується символьне розв'язання за допомогою sympy.solve. Якщо рішення не знаходиться, застосовуються чисельні методи, такі як fsolve з різними початковими здогадками. Їхнє використання дозволяє обійти обмеження символічного розв'язання, забезпечуючи більшу гнучкість та надійність.
- input_func(x_val, expr). Перетворює символьний вираз (функцію, яка була введена) в числову функцію за допомогою sympy.lambdify. Використовується для знаходження функції в заданих точках x_val.
- arrow_orbit(x0, ax, func). Візуалізує орбіту функції, починаючи з початкового значення x0. Додає стрілки, що вказують напрямок ітерацій та контролює їх масштабування в залежності від величини руху, що забезпечує кращу візуалізацію. Орбіта розраховується для перших 100 ітерацій, причому є обмеження на значення функції під час кожної (значення не має перевищувати 100.000), що в противному випадку дозволяє вийти з циклу раніше.
- newton_iter_func(expr). Обчислює ітераційну функцію Ньютона для заданого виразу. Використовує похідну функції для формування ітераційної формули. Повертає як символьний вираз ітераційної функції Ньютона, так і числову функцію.

Окрім функцій, розглянемо також основні класи:

- Plot. Базовий клас для створення графіків. Встановлює основні параметри графіку (назви осей, межі, сітку). Налаштовує обробку подій, таких як натискання кнопок миші.
- FuncOrbit. Наслідується від класу Plot. Додає функціональність для обробки орбіт функцій. Дозволяє користувачеві взаємодіяти з графіком, обираючи початкові точки для ітерацій. Знаходить всіх фіксовані точки та відображає

лише ті, що на дійсній площині. Визначає колір орбіти залежно від її поведінки. Таким чином, рожевим позначаються ті орбіти, значення яких дуже великі, зазвичай це орбіти, які віддаляються від точки-репейлера; зеленим позначаються ті, які наближаються до точки-атрактора і визначаються вони як ті, в яких різниця модулів від останнього і передостаннього значень в орбіті менше 0, або модуль їх різниці менше за дуже мале число; червоні орбіти подібні до рожевих, але для них функція не набуває дуже великих значень, а умови такі, що модуль різниці двох останніх точок більший за модуль різниці двох передостанніх, або ж різниця двох останніх точок більша за 0. Орбіти з невизначеною поведінкою позначаються чорним кольором.

- FuncNewtonMethod. Наслідується від класу Plot. Реалізує ітераційний метод Ньютона для даного виразу. Дозволяє користувачеві обирати початкові точки для ітерацій Ньютона. Відображає як початкову функцію, так і ітераційну функцію Ньютона на графіку. Будує орбіти та знаходить фіксовані точки подібно до FuncOrbit.

Новизна в методах: комбіноване використання символьних та числових методів (за допомогою fsolve) для знаходження фіксованих точок, що забезпечує гнучкість та надійність; інтерактивна візуалізація, яка дозволяє в реальному часі обирати початкові точки та бачити результати ітерацій; універсальність та модульність, оскільки класи та функції легко адаптуються для роботи з різними типами функцій і методів; можливість додавання нових методів розв'язання та візуалізації без значних змін у структурі програми.

Результати експерименту. Приклад роботи програми продемонстрований на Рис. 1.

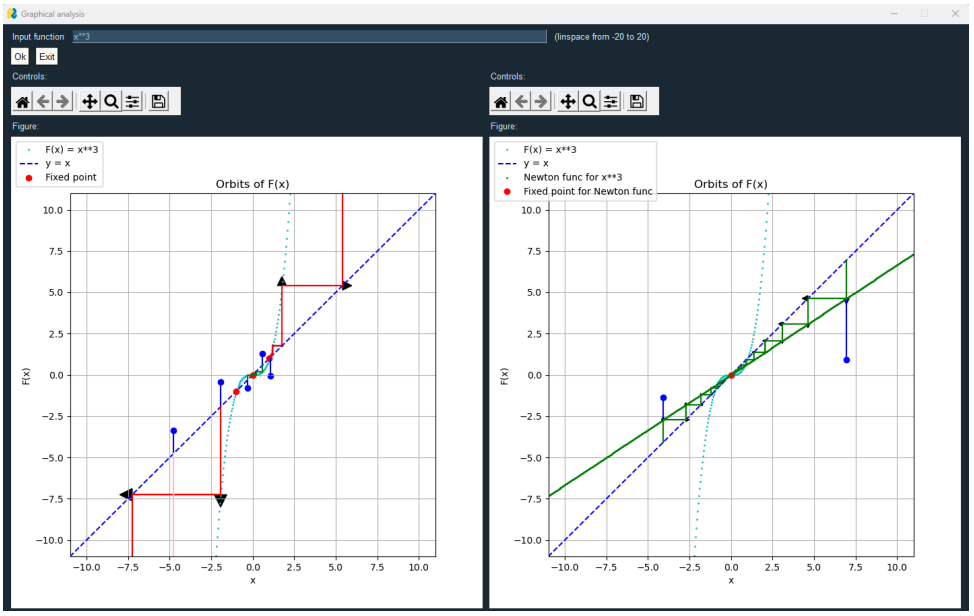

Рис.1 Графічний аналіз та метод Ньютона

Висновки. За допомогою інтерактивного інтерфейсу можна досліджувати поведінку орбіт біля фіксованих точок, а також класифікувати ці точки.

Користувацькі можливості. Програма має назву “Graphical analysis” і представляє собою вікно з можливістю введення довільної функції $F(x)$ та отримання двох графіків: знаходження фіксованих точок цієї функції ($F(x) = x$) та розв’язання рівняння $F(x) = 0$ методом Ньютона, алгоритм якого вже був детально описаний. Важливо зазначити, що введення функції підпорядковується синтаксичним правилам: обов’язково має бути присутня змінна x; додавання, віднімання, множення, ділення, піднесення в степінь мають позначатися символами “+”, “-”, “*”, “/”, “**” відповідно. Також є можливість застосування функцій sqrt, log, exp, sin та cos. Натискаючи кнопку “ОК”, отримуємо: зліва - графіки $F(x)$, $y = x$ та фіксовані точки, що отримуються при перетині функцій; справа - графіки $F(x)$, функції, побудованої методом Ньютона, $y = x$ та розв’язки для $F(x) = 0$, які отримуються шляхом перетину останніх двох. Інтерактивно можна вибрати точку правою кнопкою миші, з якої будуватиметься орбіта до чи від фіксованої точки, таким чином демонструючи її характер

(атрактор/репелер). Також зображення можна збільшувати, переміщати та зберігати.

2.2.4. Experiment: The Transition to Chaos

Завдання. Побудувати орбітну діаграми для функції:

$Q_c(x) = x^2 + c$, при $-2 \leq c \leq 0.25$, $-2 \leq x \leq 2$, та позначити періодичні точки.

Програмна реалізація. Для більш точного та повного зображення пропонується обирати різну кількість точок для різних під інтервалів:

- П'ять значень c у інтервалі -0.75 < c < 0.25.
- П'ять значень c у інтервалі -1.25 < c < -0.75.
- Щонайменше двадцять рівномірно розподілених значень c у інтервалі -1.4 < c < -1.25.
- Щонайменше п'ять рівномірно розподілених значень c у інтервалі -1.75 < c < -1.4.
- Щонайменше десять рівномірно розподілених значень c у інтервалі -1.78 < c < -1.76.
- Щонайменше п'ять рівномірно розподілених значень c у інтервалі -2 < c < -1.78.

Реалізація даного завдання представляє собою частину програми для наступного експерименту Experiment: Windows in the Orbit Diagram.

Результати експерименту. Результати експерименту наведені для експерименту, на реалізацію якого посилається даний.

Висновки. При побудові орбітної діаграми важливо правильно визначити кількість модельованих точок для різних під інтервалів значень c, оскільки при однаковому розкиді на всьому інтервалі, важливі деталі діаграми можуть не відображатись.

Користувацькі можливості. Користувацькі можливості детально описані для експерименту, на реалізацію якого посилається даний.

2.2.5. Experiment: Windows in the Orbit Diagram

Завдання. Оглянути які визначені структури та шаблони можуть виникати в орбітній діаграмі, та порівняння подібності та відмінності між функціями $Q_c(x)$ та $F_\lambda(x)$, для яких будуватиметься орбітна діаграма. Знайти порядок n-періодичних вікон для кожної з функцій та порівняти результати.

Програмна реалізація. Кожне вікно періоду n (period-n window) характеризується тим, що в ньому з'являється притягуючий цикл періоду n (attracting n-cycle), після якого відбувається явище подвоєння періоду подальших циклів (period-doubling bifurcation), з чого випливає, що послідовність періодів вікон можна отримати ітеративно. Для програмної реалізації експерименту створений графічний інтерфейс, тому програма розділена на три частини (три файли з розширенням .py), при чому в двох основних представлені розрахунки для функцій $Q_c(x)$ та $F_\lambda(x)$, інший для побудови інтерфейсу. Розглянемо детально реалізацію для $Q_c(x)$, оскільки для іншої функції реалізація аналогічна.

Початковий вибір значень c-параметрів у відрізку [-2, 0.25] важливий для побудови орбітної діаграми функції, тому використовується numpy для створення послідовностей c-значень у різних діапазонах, що допомагає спостерігати поведінку різних періодичних вікон та зони біфуркацій.

Для обчислення орбіти використовується функція compute_behavior, яка для кожного значення c обчислює її з початкової точки x0. Перші 1000 ітерацій використовуються для уникнення перехідних ефектів, після чого обчислюються 100 ітерацій для спостереження стабільної поведінки. За допомогою функції sequence_type орбіти класифікуються на фіксовані точки, цикли та невизначені або блукаючі точки. Враховується обмеження на максимальний період (200) для

виявлення циклів. Самі значення в орбіті округлюються до 7 знаків після коми для спрощення розрахунків.

Алгоритм наступний:

На проміжку значень c, що розглядається, описується характер кожної точки (точка фіксована, цикл, чи не визначена). Використовується функція period_mean_c, яка групує c-значення за їх періодом, визначеним за останніми точками орбіти, з чого отримується значення періодів для точок з координатами c.

Функція remove_near_duplicates_or_doubles використовується для автоматичного видалення близьких або дубльованих точок періоду, що підвищує точність аналізу.

Функція check_doubling_list_is_REVERSED перевіряє наявність подвоєння періодів в знайдених даних, відкидаючи близькі або дублюючі значення.

На кожній ітерації відбувається уточнення періодів за допомогою нових наборів c-значень, які обчислюються на основі попередніх результатів. Використовується функція generation, яка генерує нові значення c в межах попередньо знайдених періодів та обчислює нові орбіти. Цей процес повторюється задану кількість разів.

Для кожного періоду генерується близько 200 значень c, що дозволяє детально дослідити динамічну поведінку системи в кожному вікні. Кількість точок обирається таким чином, щоб забезпечити точне виявлення біфуркацій при збереженні прийнятного часу обчислень.

Новизна в методах: впровадження ітеративного підходу для уточнення знайдених періодів, що дозволяє точно виявляти точки біфуркації; використання функції plot_diagram для побудови графіків на кожній ітерації, що надає можливість візуального аналізу процесу сходження до періодичних вікон (кольором зображені періодичні точки періоду n, синього - Unknown/Wandering - значення періоду яких або > 200, або точка не характеризується як фіксована чи цикл.); врахування різних типів орбіт (фіксовані точки, цикли, невизначені) з

автоматичним визначенням періоду, що покращує якість аналізу динамічних систем; програмна реалізація збільшення кожної області між сусідніми вікнами, щоб, з одного боку, користувач міг самостійно визначити нові вікна, з іншого - алгоритм коректно відпрацьовував.

Програмна реалізація дозволяє автоматично визначати та класифікувати періодичні вікна для квадратичного та логістичного відображень.

Результати експерименту. Приклад роботи програми для $Q_c(x)$ продемонстрований на Рис. 2.

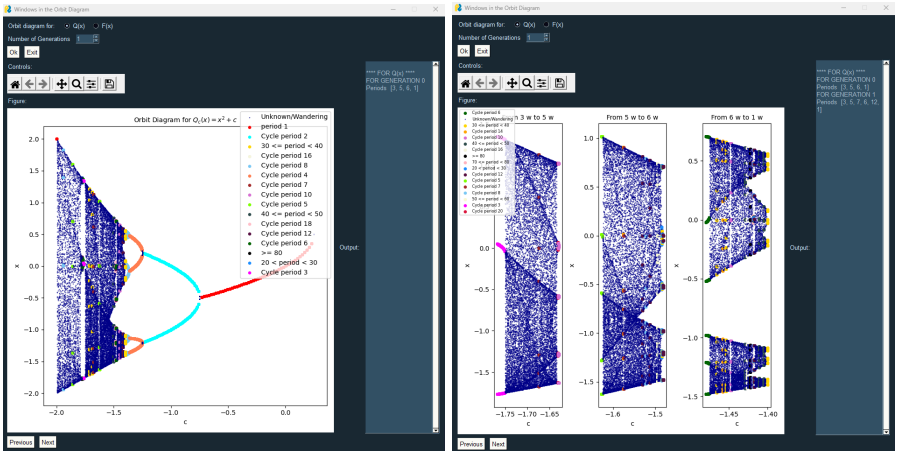

Рис. 2. Орбітна діаграма(зліва) та результат першої ітерації пошуку вікон для $Q_c(x)$

Приклад роботи програми для $F_\lambda(x)$ продемонстрований на Рис. 3.

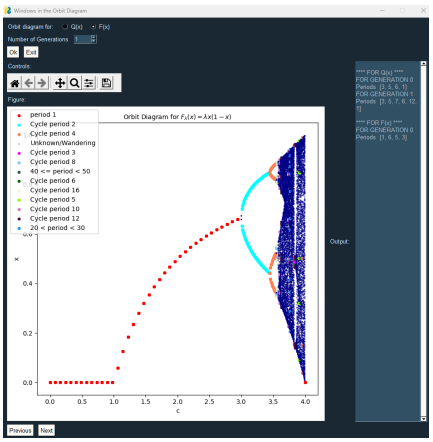

Рис. 3. Орбітна діаграма для $F_\lambda(x)$

Кольором зображені періодичні точки періоду n, синього - Unknown/Wandering - значення періоду яких або > 200, або точка не характеризується як фіксована чи цикл.

Висновки. Проводячи даний експеримент, можна зробити висновок, що функції $Q_c(x)$ та $F_\lambda(x)$ мають "дзеркальні" результати, тобто і орбітна діаграма, і послідовність вікон однакові. Після проведення достатньої кількості ітерацій можна виявити деякий шаблон: для кожного покоління значення періодів у послідовності є половиною значень найправіших періодів у послідовності наступного покоління, за винятком числа 1. Тобто якщо маємо вікна періодів 3, 5, 6 то вікна періодів 6, 10 та 12 мають бути присутні правіше цих вікон, (хоча комп'ютер не завжди може знайти всі потрібні цикли).

Теоретично, порядок період має відповідати порядку Шарковського, але на практиці, реалізуючи алгоритм комп'ютерно, отримуємо лише приблизний результат, який уточнюється зі збільшенням ітерацій. Результат роботи програми для $F_\lambda(x)$ при кількості ітерацій > 3: PERIODS [1, 48, 40, 24, 28, 20, 12, 14, 10, 6, 8, 6, 7, 5, 3]

Користувацькі можливості. Програма має назву "Windows in the Orbit Diagram" і зображує орбітну діаграму графічно для дійсних значень. Користувачу пропонується обрати одну з функцій ($Q_c(x)$ чи $F_\lambda(x)$), для якої буде проводитись подальше дослідження орбітної діаграми та знаходження n-періодичних вікон для різних значень n. Наступним кроком буде визначення бажаної кількості генерацій (ітерацій пошуку) - користувач має змогу обрати значення від 1 до 5 (кількість генерацій можна змінювати в коді програми). Натискаючи кнопку "ОК", зліва з'являється орбітна діаграма обраної функції, а справа в полі "Output" текстом зазначена обрана функція і послідовність знайдених вікон для кожної генерації. Оскільки кожна генерація виводить графічне представлення ділянок орбіти, для їх перегляду можна використовувати кнопки "Previous" та "Next". При обраній мінімальній кількості обраних генерацій (тобто 1), вивід складатиметься з двох

сторінок: генерація 0 (загальна орбітна діаграма для функції) та генерація 1 (графічне представлення ділянок орбіти для першої генерації). Отримані зображення можна збільшувати, переміщати та зберігати. Чим більше генерацій обрано, тим більше часу програмі потрібно на її виконання.

2.2.6. Experiment: Feigenbaum's Constant

Використовуючи діаграму орбіт, попередній експеримент продемонстрував, що квадратична $Q_c(x)$ та логістична $F_\lambda(x)$ функції зазнають послідовності біфуркацій подвоєння періоду, коли параметр (c або λ, відповідно) наближається до хаотичного режиму. Збільшуючи ділянку орбітної діаграми можна було також спостерігати явище самоподібності.

Завдання. Перевірити, чи біфуркації подвоєння періоду завжди відбуваються з певною однаковою швидкістю. Розглядатиметься функція $Q_c(x)$.

Програмна реалізація. Алгоритм знаходження константи Фейгенбаума був описаний вище. Теоретичне значення Фейгенбаума $\delta = 4.669201609\ldots$.

Алгоритмічно, для кожної періодичної точки потрібного, за умовою, періоду зберігаються усі значення c. Потім зі списку видаляються ті c, які не задовольняють умові суперстійкої точки (похідна функції в точці для заданого періоду має дорівнювати 0). Після цього, серед тих c, які залишились, шукається єдине значення c, для якого орбіта сходиться якнайшвидше до фіксованої точки. Останній крок - це розрахунок відношень.

Програма обчислює поведінку орбіти початкової точки x_0 для кожного значення c. Це здійснюється за допомогою функції compute_behavior, яка генерує послідовність значень x та аналізує тип орбіти (фіксована точка, цикл або невизначена). Для кожного значення c, програма визначає, чи є орбіта супер стійкою, тобто чи дорівнює похідна композитної функції нулю. Це досягається за допомогою функції find_superstable_cycles. Програма обчислює середнє значення параметра c для кожного періоду за допомогою функції period_mean_c. Ця функція

групує значення *c* по періодах та розраховує середнє значення. Функція find_fastest обчислює значення *c*, для яких орбіта найшвидше досягає стабільності або циклу. Це важливо для знаходження значень *c*, що відповідають найшвидшим періодам подвоєння. Програма також обчислює коефіцієнти Фейгенбаума для визначених значень *c*, використовуючи функцію compute_ratios, яка розраховує відношення між послідовними різницями значень *c*.

Результати експерименту. Результат, отриманий за допомогою реалізованого експерименту:

c_0 = 0.005102040816326481 для періоду 2^0

c_1 = -0.9948979591836735 для періоду 2^1

c_2 = -1.3108216432865731 для періоду 2^2

c_3 = -1.3813627254509018 для періоду 2^3

c_4 = -1.396969696969697 для періоду 2^4

c_5 = -1.4 для періоду 2^5

c_6 = -1.4006012024048096 для періоду 2^6

Відповідно, значення відношень:

f_0 = 3.165321406147852

f_1 = 4.478577226345084

f_2 = 4.519844357976643

f_3 = 5.150300601202571

f_4 = 5.040404040403604

Висновки. Як видно, значення збігаються, хоча потрібно значно більше ітерацій для уточнення результату. Для логістичної функції константа Фейгенбаума має дорівнювати тому ж значенню, що і для вже розглянутої.

Користувацькі можливості. Додаткових користувацьких можливостей у вигляді інтерактивного інтерфейсу дана програмна реалізація експерименту не має.

2.2.7. Experiment: Find the Iterated Function Systems

Завдання. Визначити ітераційні системи функцій, які породили певні фрактали та їх атрактори. В завданні представлено 6 зображень, для яких потрібно визначити формулу та порахувати фрактальну розмірність.

Програмна реалізація. Була розроблена програмна реалізація, яка дозволяє вводити кількість точок, їх координати та решту параметрів, необхідних для визначення ітераційної системи функцій, а також розрахунок фрактальної розмірності. Програмна реалізація описана в одному файлі з розширенням .py і включає в себе основну частину та інтерфейс.

Використання PySimpleGUI для створення GUI значно спрощує процес розробки інтерфейсу. PySimpleGUI дозволяє легко взаємодіяти з Matplotlib, що уможливлює візуалізацію фрактальних структур. Важливим аспектом є функція draw_figure_w_toolbar, яка відповідає за відображення графіку та панелі інструментів Matplotlib у Tkinter canvas, що створює інтерактивний досвід для користувача.

Функція iterated_function_system є основною функцією, яка представляє собою утворення ітераційної системи функцій. Вона реалізує випадковий вибір однієї з заданих точок та обчислення нової точки з використанням кута обертання θ і коефіцієнта β. Цей процес забезпечує генерацію нових точок для фрактала. Функція compute_orbit обчислює послідовність точок орбіти, використовуючи метод випадкового вибору та трансформації з iterated_function_system .

Кількість ітерацій визначається слайдером iter_slider. Це дозволяє користувачу встановлювати кількість точок, які будуть згенеровані для побудови фрактала. У прикладі використовується діапазон від 1000 до 50000 ітерацій, $0 < \beta \leq 1$, $-\pi \leq \theta \leq \pi$. Кожна ітерація обирає випадкову точку з choose_points, обчислює нову точку за допомогою функції iterated_function_system, та додає її до орбіти. Точність побудови фрактала залежить від кількості ітерацій та точності математичних обчислень (наприклад, обчислення косинуса та синуса для обертання). Кількість фіксованих точок, навколо яких відбувається трансформація,

визначається користувачем через введення значення в поле Number of points. За замовчуванням фіксовані точки ініціалізуються координатами [1/2, 1/2], але можуть бути змінені користувачем через слайдери.

Новизна в методах: інтерактивне налаштування параметрів, яке дозволяє використовувати слайдери для динамічної зміни параметрів β, θ та координат фіксованих точок без необхідності перезапуску програми; при необхідності, є можливість легко додавати нові типи трансформацій, змінюючи функцію iterated_function_system.

Результати експерименту. Таким чином, на Рис. 4 представлено зліва - перше зображення по завданню для експерименту [3], справа - вирішене завдання в програмі.

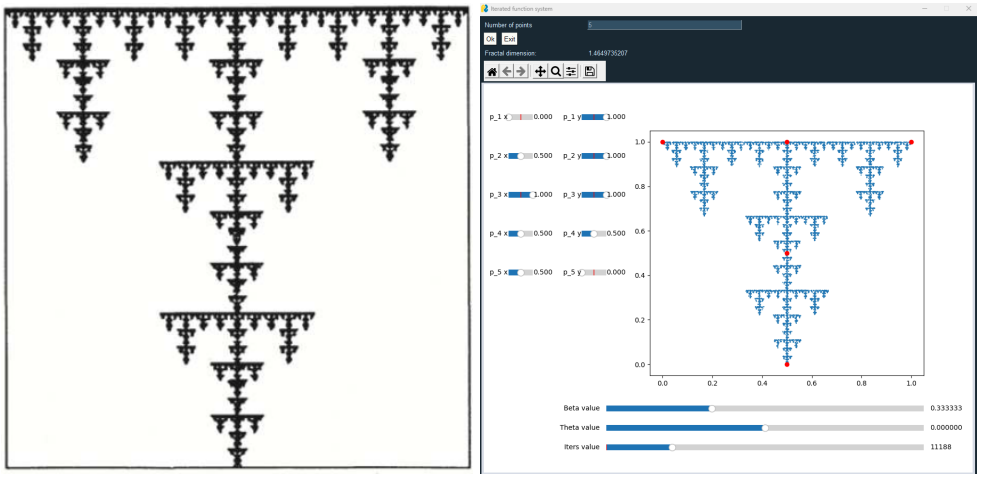

Рис. 4. Побудова фрактала методом ітераційної системи функцій.

Кількість точок була обрана таким чином, тому що на зображенні 5 основних самоподібних фігур, а значення параметру β, тому що в стільки фігура самоподібно зменшується.

Висновки. Програмою побудовано фрактал, зображення якого представлене в умові експерименту, таким чином, можна сказати, що програма відпрацьовує коректно. Спостереження: чим більше значення параметру β, тим сильніше розсіювання і менш видна фрактальна структура.

Користувацькі можливості. Програма має назву “Iterated function system” і демонструє породження певних фракталів за допомогою ітераційної системи

функцій на дійсній площині. Перш за все, користувачу пропонується обрати кількість точок, на основі яких будуватиметься фрактал. Натискаючи кнопку “ОК”, з’являється можливість фіксування координат (x, y) цих точок, значення β, $\boldsymbol{\theta}$ та кількість ітерацій за допомогою повзунків, при цьому зображення динамічно змінюється в залежності від цих значень. Таким чином, маючи, наприклад, зображення деякого фракталу, раніше вже побудованого ітераційною системою функцій, можна досить швидко знайти кількість точок, їх координати та параметри β і $\boldsymbol{\theta}$. При кожній зміні параметрів, окрім побудови самого фракталу, розраховується його фрактальна розмірність. Зображення можна збільшувати, переміщати та зберігати.

2.2.8. Experiment: Filled Julia Sets and Critical Orbits

Завдання. Дослідити взаємозв’язок між виглядом множини Жюліа(чи є вона замкненою, чи являє собою ізольовані області) та орбітою деякого обраного значення x на множині для функції $Q_c(z)$.

Програмна реалізація. Цей експеримент був програмно реалізований при проведенні експерименту Experiment: Find the Julia Set .

Результати експерименту. На Рис. 5 зображено зліва - орбіта точки для замкненої повної множини Жюліа, справа - орбіта точки для множини Жюліа у вигляді ізольованих областей.

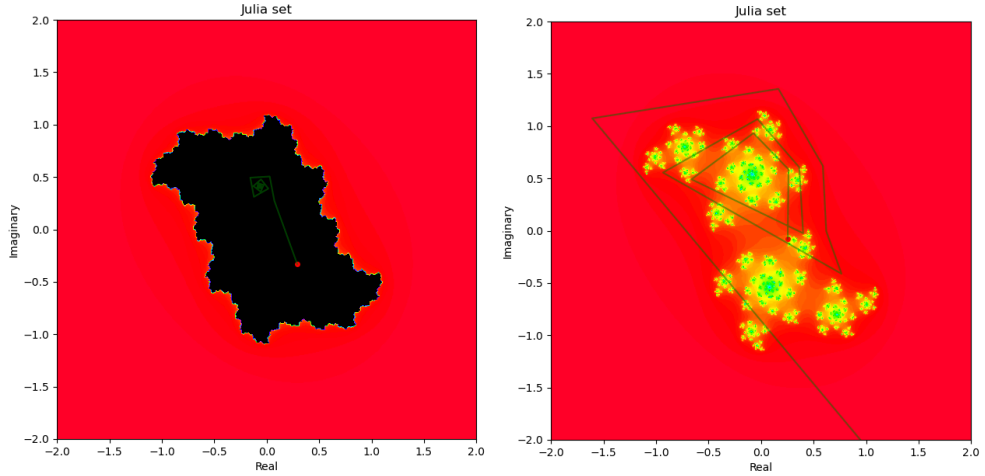

Рис. 5. Зображення орбіти точки на множині Жюліа.

Висновки. Одразу можна зазначити, що у випадку, якщо множина Жюліа замкнена, при виборі довільної точки z_0 всередині цієї області, її орбіта буде збігатись до деякої фіксованої точки, або циклу. У випадку множини у вигляді ізольованих областей, точка буде намагатись "втекти" з цієї області.

Користувацькі можливості. Користувацькі можливості детально описані для експерименту, на реалізацію якого посилається даний.

2.2.9. Experiment: Find the Julia Set

Завдання. Показати як вигляд множини Жюліа залежить від значення параметру с множини Мандельброта.

Програмна реалізація. Для цього була розроблена інтерактивна програма, в якій можна взаємодіяти з множиною Мандельброта для вибору точки с та збільшення області, та з множиною Жюліа, для вибору точки і побудови орбіти для неї, та збільшення бажаної області. Є можливість введення довільної функції, для якої будуватимуться відповідні множини.

Оберемо функцію вигляду $S_c(z) = c \cdot sin(z)$, для якої буде побудована множина Мандельброта. Потім оберемо довільне значення c (позначено білою точкою на графіку) для побудови множини Жюліа та позначимо довільну точку z_0 на ній, для відображення орбіти. Для програмної реалізації експерименту створений графічний інтерфейс, тому програма розділена на дві частини (два файли з розширенням .py), при чому в одному представлені основні розрахунки, інший для побудови інтерфейсу. Розглянемо детально перший, який є основним.

Програмна реалізація передбачає створення класів для генерації та візуалізації множин Жюліа та Мандельброта.

Клас Set є базовим класом, від якого успадковуються класи Julia та Mandelbrot. Він містить основні методи для роботи з графіками та взаємодії з користувачем через події (наприклад, натискання кнопок миші). Клас Julia

розширює базовий клас, додаючи специфічні методи для генерації та візуалізації множини Жюліа. Клас Mandelbrot також розширює базовий клас, додаючи методи для генерації та візуалізації множини Мандельброта, а також можливість інтерактивної зміни множини Жюліа.

Використовується бібліотека matplotlib для відображення множин. Функція search_critical_point знаходить критичні точки заданої функції, для якої будуються множини. Застосовуються різні колірні карти (colormaps), з можливістю налаштування, щоб підкреслити особливості множин. Наприклад, використання кастомної колірної карти з додаванням чорного кольору в кінці для виділення точок, що не сходяться.

Події кліків миші використовуються для вибору точок на графіку, які можуть бути відображені або змінені в інших множинах. Події масштабування (zoom) дозволяють збільшувати або зменшувати області на графіку для більш детального перегляду.

Метод point_orbit генерує орбіти для даних точок в множині Жюліа. Це допомагає візуалізувати траєкторії точок при ітераціях функції. Методи in_julia_set та in_mandelbrot_set перевіряють, чи належить точка множині Жюліа та Мандельброта відповідно, і скільки ітерацій потрібно для цього. Серед іншого на вхід подається важливий параметр threshold, який визначає чи буде належати точка досліджуваній множині. Методи generate_filled_julia_set та generate_mandelbrot_set генерують множини Жюліа та Мандельброта відповідно, заповнюючи їх значеннями залежно від кількості ітерацій, необхідних для визначення приналежності точки до множини. Оскільки для побудови звичайної множини Жюліа розраховується орбіта зворотних ітерацій, для якої потрібна обернена функція до початкової, то побудова цієї множини реалізована лише для $Q_c(x)$.

Точність обчислень забезпечується використанням числових методів (наприклад, символьне обчислення з sympy та числове обчислення з numpy).

Кількість ітерацій може бути налаштована для досягнення необхідної точності. Типово використовується 100-500 ітерацій. Кількість точок для побудови графіків задається параметром num_points, який зазвичай дорівнює 500 для деталізації, але може бути збільшений або зменшений, залежно від умови задачі. Знаходження бульб на множині Мандельброта відбувається за рахунок позначення різним кольором точок (параметрів c) відповідно до періодів їхніх орбіт.

Новизна в методах: впровадження методів для налаштування колірних карт, включаючи видалення небажаних кольорів і додавання чорного кольору для кращого відображення множини; можливість інтерактивного вибору точок на графіку для зміни множини Жюліа або збільшення області для детального перегляду; можливість побудови множин для довільної заданої функції; можливість знаходження бульб різних періодів на множині Мандельброта.

Результати експерименту. На Рис. 6 зображено результат роботи програми для функції $S_c(z)$.

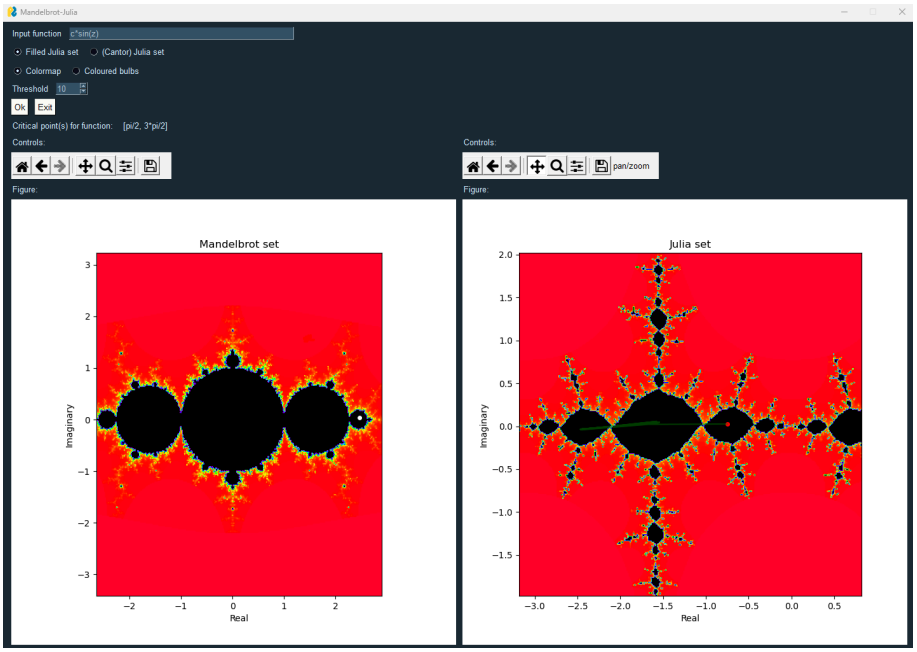

Рис. 6. Множини Мандельброта та Жюліа для функції $S_c(z)$.

Висновки. Програмна реалізація наглядно демонструє, як вигляд множини Жюліа залежить від значення параметру c множини Мандельброта.

Користувацькі можливості. Програма називається “Mandelbrot-Julia” і зображує множини Мандельброта та Жюліа таким чином, що їх можна співставити одна до одної, обираючи точку, що відповідає параметру c на першій, та отримуючи відповідну множину другої. Користувачу надається можливість вводити довільну функцію, враховуючи всі синтаксичні обмеження, описані вище для програми “Graphical analysis” з тією відмінністю, що в цьому випадку замість змінної “x” використовується змінна “z”. Присутня опція вибору яким чином буде побудована множина Жюліа - методом для заповненої множини Жюліа (Filled Julia set), чи для звичайної множини Жюліа (Cantor Julia set). Побудова останньої реалізована лише для функції $Q_c(x)$, оскільки вимагає знаходження оберненої функції до даної.

Наступною парою опцій для вибору є: colormap - тобто, “розфарбовані” множини, алгоритм знаходження яких був описаний вище, чи coloured bulbs - “розфарбовані” множини по бульбам відповідно до їх періодів. Присутня можливість вибору деякого граничного значення threshold, від якого залежить чи увійде точка в множину Мандельброта. Значення threshold = [1, 2, 5, 10, 20], по замовчуванню threshold = 2 (саме такий поріг для сім’ї функцій $Q_c(x)$). Після натискання кнопки “ОК”, першим, що виводиться, це список критичних точок для заданої функції. Якщо точок немає, користувачу це повідомляється і пропонується видозмінити функцію, у випадку кількості точок ≥ 1, обирається перша запропонована точка і далі вона використовується у дослідженні. Зліва зображена множина Мандельброта, на якій, за допомогою правої кнопки миші, можна вибрати точку (параметр c), після чого справа будується множина Жюліа для відповідного значення. Вже на множині Жюліа правою кнопкою миші можна вибрати довільну точку, для якої буде будуватися її орбіта. На одному графіку точок може бути декілька. Отримані зображення можна збільшувати, переміщати та зберігати. Після збільшення зображення, на нього потрібно натиснути лівою кнопкою миші, щоб воно стало більш детальним. Час виконання програми залежить від складності побудови функції, кількості ітерацій, а також від того, чи

була обрана опція пошуку бульб, оскільки для кожної точки, яка перевіряється, будується її орбіта, визначається період і зафарбовується відповідним кольором, що може значно уповільнити роботу програми.

2.2.10. Experiment: Similarity of the Mandelbrot Set and Julia Sets

Завдання. Показати значний взаємозв'язок між структурою множини Мандельброта для деяких параметрів c та відповідною множиною Жюліа. Розглядатимемо функцію $Q_c(z)$.

Програмна реалізація. Реалізація даного завдання представляє собою частину програми для експерименту Experiment: Find the Julia Set.

Результати експерименту. На Рис. 7 зліва зображена збільшена область множини Мандельброта та позначено значення параметру c білою точкою, справа - множина Жюліа для значення c.

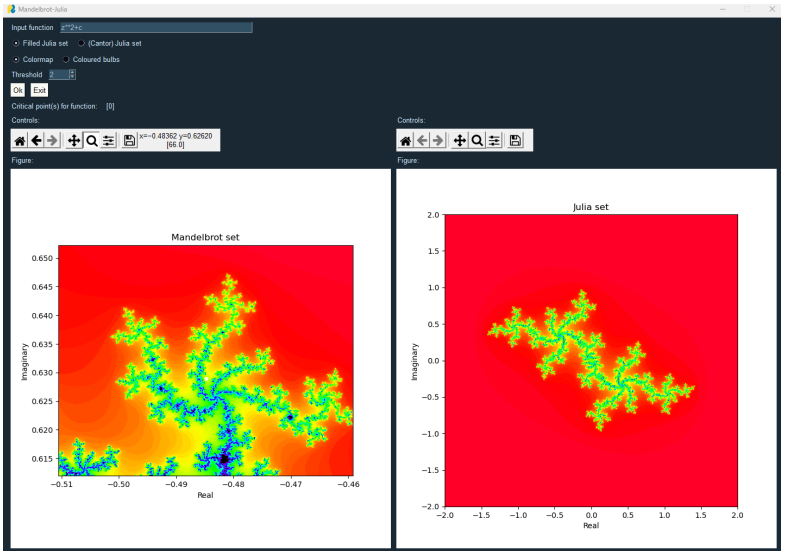

Рис. 7. Зображення множини Мандельброта та множини Жюліа для визначення взаємозв'язку між ними.

Висновки. Подібність між збільшеними ділянками множини Мандельброта та відповідними заповненими множинами Жюліа виявляється лише біля певних

значень c, таких як центральні з'єднання антени. Це значення c, для яких 0 в кінці кінцево-періодичне, і називається точками Місюревича.

Користувацькі можливості. Користувацькі можливості детально описані для експерименту, на реалізацію якого посилається даний.

2.2.11. Experiment: Periods of the Bulbs on the main cardioid

Завдання. Знаходження бульб $B_{p/q}$ на головній кардіоїді для функції $Q_c(z)$.

Програмна реалізація. Алгоритм знаходження $\frac{p}{q}$ був описаний вище в розділі 1.3.8. . Функція point_orbit обчислює орбіту точки z під дією функції q_c з параметром c на задану кількість ітерацій. Функція sequence_type визначає тип послідовності: чи є вона фіксованою точкою, циклом або такою, що сходиться до фіксованої точки. Якщо послідовність не є жодним з цих типів, вона вважається "Unknown/Wandering". Функція plot_diagram будує діаграму, яка візуалізує результати аналізу орбіт. Вона малює точки різного кольору в залежності від типу орбіти та її періоду. Функція in_mandelbrot_set перевіряє, чи належить точка c до множини Мандельброта, використовуючи задану кількість ітерацій та поріг. Функція generate_mandelbrot_set генерує множину Мандельброта для заданих меж, використовуючи зазначену кількість точок. Вона також може генерувати кольорове зображення, що відображає час втечі для кожної точки. Функції для додаткових меж cardioida та cycle_2 використовуються для обчислення меж основної кардіоїди та кола для бульби другого періоду. Таким чином, генерується та візуалізується множина Мандульброта, генеруються межі для кардіоїди та кола та зображується графік. Додатково додаються точки $\frac{p}{q}$ вигляду “*”, значення яких можна зазначати в умові програми.

Результати експерименту. На Рис. 8 представлена множина Мандельброта та позначено бульби в першій чверті для значень $\frac{p}{q} = [\frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}, \frac{1}{8}, \frac{1}{9}, \frac{1}{12}]$

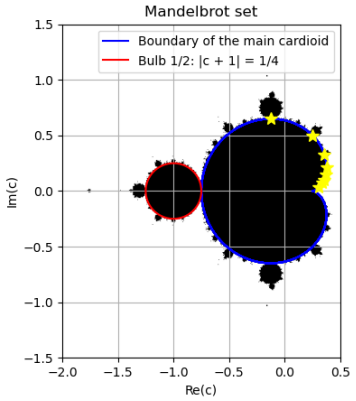

Рис. 9. Зображення множини Мандельброта та визначення бульб.

Висновки. Експериментально отримали правильний результат, що зображує кардіоїду, коло для бульби періоду 2, та позначає розташування обраних бульб при різних $\frac{p}{q}$.

Користувацькі можливості. Додаткових користувацьких можливостей у вигляді інтерактивного інтерфейсу дана програмна реалізація експерименту не має, тим не менше, є можливість зміни значень параметрів безпосередньо в коді програми.

2.2.12. Experiment: Periods of the other Bulbs(coloured)

Завдання. Визначити період для різних бульб на множині Мандельброта та знайти спосіб їх ідентифікувати.

Програмна реалізація. Було прийнято рішення провести цей експеримент графічно, замальовуючи бульби на множині в різні кольори, відповідно до значень періодів.

Реалізація даного завдання представляє собою частину програми для експерименту Experiment: Find the Julia Set.

Результати експерименту. На Рис. 9 зображена множина Мандельброта для функції $Q_c(z)$ та множина Жюліа для параметра c, який був обраний в бульбі 3-тього періоду (позначено фіолетовим кольором).

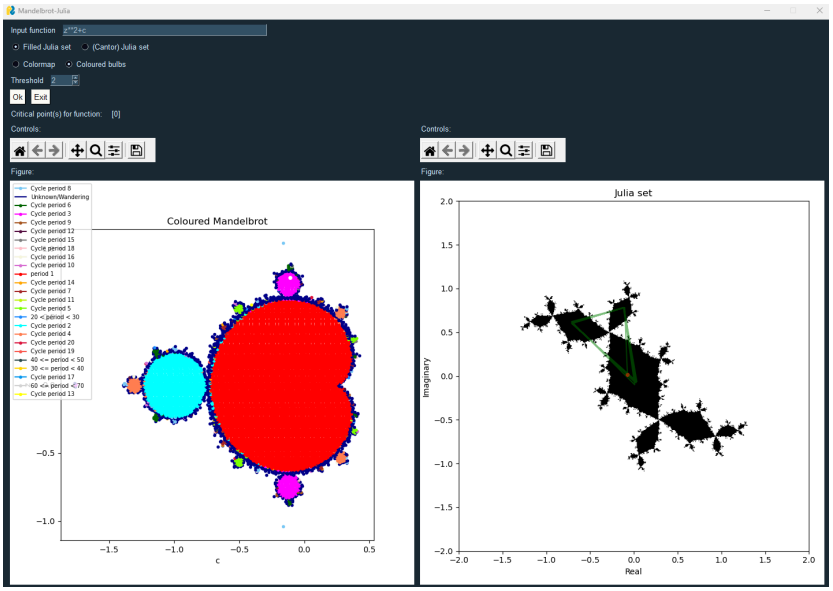

Рис. 9. Зображення множини Мандельброта та множини Жюліа для визначення бульб.

Висновки. Як видно, на множині Жюліа орбіта має період 3, що відповідає бульби, на якій було обрано точку параметра *c*.

Користувацькі можливості. Користувацькі можливості детально описані для експерименту, на реалізацію якого посилається даний.

ВИСНОВКИ

Дослідження динамічних систем з детермінованим хаосом є актуальним у зв'язку зі стрімким зростанням інтересу до пошуків нових моделей нерегулярної поведінки в складних системах різної природи. Це стосується не лише наукової сфери, але й практичного застосування в різних галузях, таких як фінанси, економіка, природознавство та техніка.

Мета дослідження полягає в експериментальній перевірці властивостей динамічних систем з детермінованим хаосом за допомогою комп’ютерних засобів. Для досягнення цієї мети було виконано ряд завдань:

- На основі аналізу динамічних систем з детермінованим хаосом та відомостей про обчислювальні експерименти було виділено серію експериментів для подальшого дослідження: Experiment: The Computer May Lie, Experiment: Rates of Convergence, Graphical Analysis and Newton’s Method, Experiment: The Transition to Chaos, Experiment: Windows in the Orbit Diagram, Experiment: Feigenbaum’s Constant, Experiment: Find the Iterated Function Systems, Experiment: Filled Julia Sets and Critical Orbits, Experiment: Find the Julia Set, Experiment: Similarity of the Mandelbrot Set and Julia Sets, Experiment: Periods of the Bulbs on the main cardioid, Experiment: Periods of the other Bulbs(coloured).
- Проведено аналіз алгоритмів для розглянутих обчислювальних експериментів та реалізовано програмні продукти. Для наступних експериментів присутній інтерактивний інтерфейс для введення параметрів і дослідження поведінки ДС: Graphical Analysis and Newton’s Method, Experiment: Windows in the Orbit Diagram, Experiment: Find the Iterated Function Systems, Experiment: Find the Julia Set.

- Виконано аналіз результатів та підтверджено властивості динамічних систем для наступних експериментів:
    - Experiment: The Computer May Lie: продемонстровані випадки некоректного розрахунку (і як результат, класифікацію) комп'ютером значення орбіт для різних значень в малому околі, що доводить чутливість функції від початкових умов.
    - Experiment: Rates of Convergence: продемонстровано, що швидкість збіжності орбіти до точки періоду n залежить від значення $\left|F^{(n)}(p)\right|$.
    - Graphical Analysis and Newton's Method: за допомогою інтерактивного інтерфейсу проведено дослідження поведінки орбіт біля фіксованих точок, а також класифікація цих точок.
    - Experiment: Windows in the Orbit Diagram: продемонстровані результати послідовностей періодів для функцій $Q_c(x)$ та $F_\lambda(x)$, і показано їх взаємозв'язок.
    - Experiment: Feigenbaum's Constant: отримані експериментальні результати підтверджують теоретичні дані.
    - Experiment: Find the Iterated Function Systems: продемонстрована коректна побудова фракталу методом ітераційної системи функцій.
    - Experiment: Filled Julia Sets and Critical Orbits: експериментально досліджено та доведено взаємозв'язок між виглядом множини Жюліа(чи є вона замкненою, чи являє собою ізольовані області) та орбітою деякого обраного значення x на множині для функції $Q_c(z)$.
    - Experiment: Find the Julia Set: програмна реалізація наглядно демонструє, як вигляд множини Жюліа залежить від значення

параметру c множини Мандельброта та підтверджує принцип фундаментальної дихотомії.

- Experiment: Similarity of the Mandelbrot Set and Julia Sets: продемонстровано, що подібність між збільшеними ділянками множини Мандельброта та відповідними заповненими множинами Жюліа виявляється біля певних значень c, таких як центральні з'єднання антени.
- Experiment: Periods of the Bulbs on the main cardioid: експериментально продемонстрували розташування бульб на основній кардіоїді для різних значень $\frac{p}{q}$.
- Experiment: Periods of the other Bulbs(coloured): продемонстровано, що період орбіти точки з множини Жюліа буде дорівнювати періоду бульби на множині Мандельброта, з якої було обрано параметр *c*.

Можливі доопрацювання включають оптимізацію коду, інтеграція з різними математичними пакетами по типу Wolfram, Maple, не обмежуючись також використанням інших методів дослідження різних типів динамічних систем.

ЛІТЕРАТУРА

1) Д. П. Герасимчук “Моделювання детермінованого хаосу засобами обчислювального експерименту”
2) Steven H. Strogatz. NONLINEAR DYNAMICS AND CHAOS with Applications to Physics, Biology, Chemistry and Engineering-Second Edition.
3) Robert L. Devaney *A First Course in Chaotic Dynamical Systems Theory and Experiment Second Edition*. London: CRC Press, Taylor and Francis Group, 2020. 329 p.
4) В. В. Пічкур, О. В. Капустян, В. В. Собчук. ТЕОРІЯ ДИНАМІЧНИХ СИСТЕМ, Навчальний посібник
5) “A history of chaos theory”, Christian Oestreicher [Електронний ресурс] - Режим доступу: A history of chaos theory - PMC (nih.gov)
6) Morris W. Hirsch, Stephen Smale, Robert L. Devaney. DIFFERENTIAL EQUATIONS, DYNAMICAL SYSTEMS, AND AN INTRODUCTION TO CHAOS
7) Michael Brin, Garrett Stuck. Introduction to Dynamical Systems
8) Douady, A. and Hubbard, J. On the Dynamics of Polynomial-like Maps.Ann. Scient. Ec. Norm. Sup. 18 (1985), 287.
9) Sebastian M. Marotta, Mathematica Tools for Use in MA471/671, ресурс: http://math.bu.edu/people/bob/MA471/mathematica.html
10) Wolfram Research. Wolfram Mathematica, ресурс: https://www.wolfram.com/mathematica/
11) Stephen Lynch. Dynamical Systems with Applications using MapleTM Second Edition
12) Javascript Julia Set Generator , ресурс: https://www.marksmath.org/visualization/julia_sets/
13) NumPy documentation. NumPy: the absolute basics for beginners. ресурс: https://numpy.org/doc/stable/user/absolute_beginners.html

14) PySimpleGUI, PySimpleGUI Documentation, pecypc: https://docs.pysimplegui.com/en/latest/documentation/module/elements/

15) The Dynamical Systems and Technology Project at Boston University. Chaos, Fractals, and Arcadia, pecypc:http://math.bu.edu/DYSYS/arcadia/sect2.html

16) The Mathenæum: Mathematical explorations, games, and learning, Fractals Part III, pecypc:http://thewessens.net/ClassroomApps/Main/chaosgame.html

17) The Dynamical Systems and Technology Project at Boston University. Exploration #7, pecypc: http://math.bu.edu/DYSYS/explorer/tour7.html

ДОДАТКИ

Додаток А

Скріншоти роботи програми Experiment_TheComputerMayLie

```
Results for Function a (F(x) = x^2 - 2):
(-1.9, 'No visible pattern:', [1.6099999999999999, 0.5920999999999994, -1.6494175900000008, 0.7205783862014106, -1.4807667893393708, 0.19267028441042866, -1.9628781
(-1.5, 'No visible pattern:', [0.25, -1.9375, 1.75390625, 1.0761871337890625, -0.8418212530668825, -1.2913369778849038, -0.33244880954708367, -1.8894777890307268, 1
(-1, 'Fixed point:', -1)
(-0.5, 'No visible pattern:', [-1.75, 1.0625, -0.87109375, -1.2411956787109375, -0.4594332871492952, -1.7889210546591932, 1.2002385398029598, -0.559427447571659, -1
(0, 'Eventually periodic:', [-2, 2, 2, 2])
(0.5, 'No visible pattern:', [-1.75, 1.0625, -0.87109375, -1.2411956787109375, -0.4594332871492952, -1.7889210546591932, 1.2002385398029598, -0.559427447571659, -1.
(1, 'Fixed point:', -1)
(1.5, 'No visible pattern:', [0.25, -1.9375, 1.75390625, 1.0761871337890625, -0.8418212530668825, -1.2913369778849038, -0.33244880954708367, -1.8894777890307268, 1.
(1.9, 'No visible pattern:', [1.6099999999999999, 0.5920999999999994, -1.6494175900000008, 0.7205783862014106, -1.4807667893393708, 0.19267028441042866, -1.96287816

Func b

Results for Function b (G(x) = x^2 - 3):
(-1.9, 'No visible pattern:', [0.6099999999999999, -2.6279000000000003, 3.9058584100000022, 12.255729918967745, 147.20291584668112, 21665.698433765083, 469402485.62
(-1.5, 'No visible pattern:', [-0.75, -2.4375, 2.94140625, 5.6518707275390625, 28.94364272081293, 834.7344539500674, 696778.6086113172, 485500429415.3232, 2.3571066
(-1, 'Fixed point:', -2)
(-0.5, 'No visible pattern:', [-2.75, 4.5625, 17.81640625, 314.42433166503906, 98859.66034300649, 9773232440.13461, 9.55160723288995e+19, 9.12332007313956e+39, 8.32
(0, 'No visible pattern:', [-3, 6, 33, 1086, 1179393, 1390967848446, 193479155541049442461 4913, 37434183628877603174075412715593584918683419975 66, 140131810396052 79
(0.5, 'No visible pattern:', [-2.75, 4.5625, 17.81640625, 314.42433166503906, 98859.66034300649, 9773232440.13461, 9.55160723288995e+19, 9.12332007313956e+39, 8.323
(1, 'Fixed point:', -2)
(1.5, 'No visible pattern:', [-0.75, -2.4375, 2.94140625, 5.6518707275390625, 28.94364272081293, 834.7344539500674, 696778.6086113172, 485500429415.3232, 2.35710666
(1.9, 'No visible pattern:', [0.6099999999999999, -2.6279000000000003, 3.9058584100000022, 12.255729918967745, 147.20291584668112, 21665.698433765083, 469402485.622

Func c

Results for Function c (Doubling function):
(0.2, 'Eventually periodic:', [0.4, 0.8, 0.6000000000000001, 0.20000000000000018, 0.40000000000000036, 0.8000000000000007, 0.6000000000000014, 0.20000000000000284,
(0.1111111111111111, 'Eventually periodic:', [0.2222222222222222, 0.4444444444444444, 0.8888888888888888, 0.7777777777777777, 0.5555555555555554, 0.1111111111111107
(0.3, 'Eventually periodic:', [0.6, 0.19999999999999996, 0.3999999999999999, 0.7999999999999998, 0.5999999999999996, 0.1999999999999993, 0.3999999999999986, 0.79999
(0.7, 'Eventually periodic:', [0.3999999999999999, 0.7999999999999998, 0.5999999999999996, 0.1999999999999993, 0.3999999999999986, 0.7999999999999972, 0.59999999999
(0.125, 'Eventually periodic:', [0.25, 0.5, 0.0, 0.0])
(0.0625, 'Eventually periodic:', [0.125, 0.25, 0.5, 0.0, 0.0])
(0.14285714285714285, 'Eventually periodic:', [0.2857142857142857, 0.5714285714285714, 0.1428571428571428, 0.2857142857142856, 0.5714285714285712, 0.142857142857142
(0.07142857142857142, 'Eventually periodic:', [0.1428571428571428, 0.2857142857142857, 0.5714285714285714, 0.1428571428571428, 0.2857142857142856, 0.57142857142857
(0.09090909090909091, 'Eventually periodic:', [0.18181818181818182, 0.36363636363636365, 0.7272727272727273, 0.4545454545454546, 0.9090909090909092, 0.8181818181818
(0.13636363636363635, 'Eventually periodic:', [0.2727272727272727, 0.5454545454545454, 0.09090909090909083, 0.18181818181818166, 0.3636363636363633, 0.7272727272727
```

Додаток Б

Скріншоти роботи програми Experiment_RatesOfConvergence-part1

```
Function: F(x) = x^2 + 0.25
a. The fixed point(s): [0.500000000000000]
a. The destined fixed point p: 0.500000000000000
b. |F'(fixed_points)|: [1.00000000000000]
b. |F'(p)|: 1.00000000000000
c. Is p attracting or neutral?: Neutral
d. Number of iterations necessary for the orbit of 0.2 to 'reach' p: 310
e. Last 3 points of the orbit of 0.2 to 'reach' p in 1000 iterations: [0.49683624403729953, 0.49684625338909105, 0.49685619950677684]

Function: F(x) = x^2
a. The fixed point(s): [0, 1]
a. The destined fixed point p: 0
b. |F'(fixed_points)|: [0 2]
b. |F'(p)|: 0
c. Is p attracting or neutral?: Attracting
d. Number of iterations necessary for the orbit of 0.2 to 'reach' p: 4
e. Last 3 points of the orbit of 0.2 to 'reach' p in 1000 iterations: [0.0016000000000000007, 2.560000000000002e-06, 6.553600000000011e-12]

Function: F(x) = x^2 - 0.24
a. The fixed point(s): [-0.200000000000000, 1.20000000000000]
a. The destined fixed point p: -0.200000000000000
b. |F'(fixed_points)|: [0.400000000000000 2.40000000000000]
b. |F'(p)|: 0.400000000000000
c. Is p attracting or neutral?: Attracting
d. Number of iterations necessary for the orbit of 0.2 to 'reach' p: 2
e. Last 3 points of the orbit of 0.2 to 'reach' p in 1000 iterations: [0.2, -0.19999999999999998, -0.2]

Function: F(x) = x^2 - 0.75
a. The fixed point(s): [-0.500000000000000, 1.50000000000000]
a. The destined fixed point p: -0.500000000000000
b. |F'(fixed_points)|: [1.00000000000000 3.00000000000000]
b. |F'(p)|: 1.00000000000000
c. Is p attracting or neutral?: Neutral
d. Number of iterations necessary for the orbit of 0.2 to 'reach' p: 1000
e. Last 3 points of the orbit of 0.2 to 'reach' p in 1000 iterations: [-0.4775281512649497, -0.5219668647494793, -0.4775505921035988]
```

```
Function: F(x) = 0.4x(1 - x)
a. The fixed point(s): [-1.50000000000000, 0.0]
a. The destined fixed point p: 0.0
b. |F'(fixed_points)|: [1.60000000000000 0.400000000000000]
b. |F'(p)|: 0.400000000000000
c. Is p attracting or neutral?: Attracting
d. Number of iterations necessary for the orbit of 0.2 to 'reach' p: 11
e. Last 3 points of the orbit of 0.2 to 'reach' p in 1000 iterations: [3.7728192399744874e-05, 1.5090707593297249e-05, 6.03619194553663

Function: F(x) = x(1 - x)
a. The fixed point(s): [0]
a. The destined fixed point p: 0
b. |F'(fixed_points)|: [1]
b. |F'(p)|: 1
c. Is p attracting or neutral?: Neutral
d. Number of iterations necessary for the orbit of 0.2 to 'reach' p: 308
e. Last 3 points of the orbit of 0.2 to 'reach' p in 1000 iterations: [0.003171988695821923, 0.0031619271835355007, 0.00315192940002152

Function: F(x) = 1.6x(1 - x)
a. The fixed point(s): [0.0, 0.375000000000000]
a. The destined fixed point p: 0.375000000000000
b. |F'(fixed_points)|: [1.60000000000000 0.400000000000000]
b. |F'(p)|: 0.400000000000000
c. Is p attracting or neutral?: Attracting
d. Number of iterations necessary for the orbit of 0.2 to 'reach' p: 13
e. Last 3 points of the orbit of 0.2 to 'reach' p in 1000 iterations: [0.37496985219024925, 0.374987939421875, 0.37499517553601797]

Function: F(x) = 2x(1 - x)
a. The fixed point(s): [0, 1/2]
a. The destined fixed point p: 1/2
b. |F'(fixed_points)|: [2 0]
b. |F'(p)|: 0
c. Is p attracting or neutral?: Attracting
d. Number of iterations necessary for the orbit of 0.2 to 'reach' p: 6
e. Last 3 points of the orbit of 0.2 to 'reach' p in 1000 iterations: [0.4998589445046272, 0.49999996020669446, 0.49999999999999684]
```

```
Function: F(x) = 2.4x(1 - x)
a. The fixed point(s): [0.0, 0.583333333333333]
a. The destined fixed point p: 0.583333333333333
b. |F'(fixed_points)|: [2.40000000000000 0.400000000000000]
b. |F'(p)|: 0.400000000000000
c. Is p attracting or neutral?: Attracting
d. Number of iterations necessary for the orbit of 0.2 to 'reach' p: 12
e. Last 3 points of the orbit of 0.2 to 'reach' p in 1000 iterations: [0.5833238336590432, 0.5833371329864642, 0.5833318134374312]

Function: F(x) = 3x(1 - x)
a. The fixed point(s): [0, 2/3]
a. The destined fixed point p: 2/3
b. |F'(fixed_points)|: [3 1]
b. |F'(p)|: 1
c. Is p attracting or neutral?: Neutral
d. Number of iterations necessary for the orbit of 0.2 to 'reach' p: 1000
e. Last 3 points of the orbit of 0.2 to 'reach' p in 1000 iterations: [0.6740041061251537, 0.6591677131547587, 0.6739969172672533]

Function: F(x) = 0.4 sin(x)
a. The fixed point(s): [3.328945176526053e-06]
a. The destined fixed point p: 3.328945176526053e-06
b. |F'(fixed_points)|: [0.4]
b. |F'(p)|: 0.3999999999977837
c. Is p attracting or neutral?: Attracting
d. Number of iterations necessary for the orbit of 0.2 to 'reach' p: 12
e. Last 3 points of the orbit of 0.2 to 'reach' p in 1000 iterations: [2.0805907355029106e-05, 8.322362941411203e-06, 3.328945176526053e-06]

Function: F(x) = sin(x)
a. The fixed point(s): [0.05278775407987688]
a. The destined fixed point p: 0.05278775407987688
b. |F'(fixed_points)|: [0.99860705]
b. |F'(p)|: 0.998607050014721
c. Is p attracting or neutral?: Attracting
d. Number of iterations necessary for the orbit of 0.2 to 'reach' p: 1000
e. Last 3 points of the orbit of 0.2 to 'reach' p in 1000 iterations: [0.052836881805511084, 0.052812300798740386, 0.05278775407987688]
```

Додаток В

Скріншот роботи програми Experiment_RatesOfConvergence-part2

```
Function: F(x) = x^2 - 1
The fixed point(s): [1/2 - sqrt(5)/2, 1/2 + sqrt(5)/2]
The destined fixed point p: 1/2 - sqrt(5)/2
Is the cycle attracting or neutral?: Attracting
Cycle of period 2: [0.2, -0.96, -0.07840000000000003, -0.99385344, -0.012255339800166465, -0.9998498066463825, -0.000300
|F'2(p)|: 2.602362769721282e-13
Number of iterations necessary for the orbit of 0.2 to 'reach' p: 9

Function: F(x) = x^2 - 1.1
The fixed point(s): [-0.661895003862225, 1.66189500386223]
The destined fixed point p: -0.661895003862225
Is the cycle attracting or neutral?: Attracting
Cycle of period 2: [0.2, -1.06, 0.023600000000000065, -1.0994430400000001, 0.10877499820444192, -1.0881679997656237, 0.0
|F'2(p)|: 0.3999783840854565
Number of iterations necessary for the orbit of 0.2 to 'reach' p: 21

Function: F(x) = x^2 - 1.25
The fixed point(s): [-0.724744871391589, 1.72474487139159]
The destined fixed point p: -0.724744871391589
|F'4(p)| for last func: 0.998653818345231
Cycle of period 4: [0.2, -1.21, 0.21409999999999996, -1.20416119, 0.20000417150221605, -1.2099983313817122, 0.2140959619
|F'2(p)|: 0.9718807592081666
Number of iterations necessary for the orbit of 0.2 to 'reach' p: 1000
```