Міністерство освіти і науки України

Чернівецький національний університет

імені Юрія Федьковича

Інститут фізико-технічних та комп'ютерних наук

(повна назва інституту/факультету)

Програмного забезпечення комп'ютерних систем

(повна назва кафедри)

Дослідження ефективності використання генетичного алгоритму для розв'язання математичних рівнянь

Дипломна робота

освітнього рівня «Магістр»

Виконав: студент (ка) 6 курсу, групи 643 спеціальності

121 – Інженерія програмного забезпечення

(шифр і назва спеціальності)

Мораараш А.В

(прізвище та ініціали)

Керівник Шумиляк Л.М

(прізвище та ініціали)

Рецензент

(прізвище та ініціали)

До захисту допущено:

Протокол засідання кафедри №

від „_____" _____ 20___ р.

зав. кафедри _____ проф. Остапов С.Е.

Чернівці – 2022

Чернівецький національний університет імені Юрія Федьковича
(назва вузу)

Інститут *фізико-технічних та комп'ютерних наук*
Кафедра *програмного забезпечення комп'ютерних систем*
Спеціальність *Інженерія програмного забезпечення*

ЗАТВЕРДЖУЮ
зав. кафедрою _____
"___"_____2022 р.

ЗАВДАННЯ НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Мораараш Артем Володимирович
(прізвище, ім'я, по батькові)

1. Тема роботи: Дослідження ефективності застосування генетичного алгоритму для розв'язання математичних рівнянь

2. Затверджена наказом по університету від «___»_____202___р. № ___

Термін подачі студентом закінченого проекту (роботи) *" " 20 р.*___

3. Вихідні дані до проекту (роботи) програмне забезпечення, дані та ілюстрації досліджень

4. Зміст документації до кваліфікаційної роботи

(перелік питань, що їх належить розробити) документація повинна містити аналіз вимог до програмного забезпечення; архітектура розробленого програмного забезпечення; обґрунтування вибору засобів розробки; опис програми; методики досліджень; результати досліджень висновки та додатки, які містять код програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) блок-схема(и) роботи програми; діаграма модулів; діаграма класів; графіки результатів досліджень

6. Консультанти по проекту (роботі), із зазначенням розділів проекту, щостосуються їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівники _____

(підпис)

Завдання прийняв до виконання _____

(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Одержання технічного завдання	10.09.2022
2	Аналіз наявних рішень	20.09.2022
3	Затвердження вимог до ПЗ керівником	25.09.2022
4	Вибір оптимальної архітектури ПЗ	01.10.2022
5	Розробка програмного забезпечення	25.10.2022
6	Проведення дослідження та аналіз результатів	15.11.2022
7	Оформлення програмної документації	25.11.2022
8	Представлення готової роботи	13.11.2022
9	Захист роботи	згідно розкладу

Студент _____

(підпис)

Керівник проекту_____

АНОТАЦІЯ

Метою дипломної роботи є дослідження ефективності використання генетичного алгоритму для розв’язання математичних рівнянь. Також додатковаю задачею є створення програмного продукту, який можна буде використати для цієї задачі.

В даній роботі розроблено програмний продукт у вигляді локального веб-застосунку, яким можна розв’язувати лінійні та нелінійні математичні рівняння з одним розв’язком, проводити дослідження в рамках різних налаштувань алгоритму. Проведено дослідження ефективності використання такого підходу при різних налаштуваннях генетичного алгоритму, проведено порівняння з аналогом на тригонометричних, лінійних, логарифмічних та показникових рівняннях. В якості аналога була вибрана sympy – бібліотекою з інструментами для роботи з комп’ютерною алгеброю, в яку також входить функціонал для розв’язування рівнянь. Детальні результати наведено далі.

Проведені дослідження показують, що генетичний алгоритм можна застосовувати для такого типу задач, коли не потрібна висока точність (в межах декількох сотих або десятих), але швидкодія сильно залежить від початкового налаштування параметрів.

Ключові слова: КОМП’ЮТЕРНЕ ПРОГРАМУВАННЯ, PYTHON, МАТЕМАТИЧНІ РІВНЯНЯ, ГЕНЕТИЧНИЙ АЛГОРИТМ, КОМП’ЮТЕРНА АЛГЕБРА.

ABSTRACT

The purpose of the thesis is to study the effectiveness of using genetic algorithm for solving mathematical equations. Also an additional task is to create a software product that can be used to solve this problem.

In this work, a software product has been developed in the form of a local web application that can solve linear and nonlinear mathematical equations with one solution, conduct research within different algorithm settings. The study of the effectiveness of using such an approach at different settings of the genetic algorithm was carried out, a comparison with the analogue on trigonometric, linear, logarithmic and exponential equations was carried out. As an analogue was chosen sympy - a library with tools for working with computer algebra which also includes functionality for solving equations. Detailed results are given below.

The results show that the genetic algorithm can be used for this type of problems, when high accuracy is not required (within a few hundredths or tenths), but the performance strongly depends on the initial parameter setting.

Keywords: COMPUTER PROGRAMMING, PYTHON, MATHEMATICAL EQUATIONS, GENETIC ALGORITHM, COMPUTER ALGEBRA.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ГА – генетичний алгоритм

МНК – метод найменших квадратів

ПЗ – програмне забезпечення

ВСТУП

Актуальність теми. В наш час наука та повсякденне життя не обходиться без комп'ютерів, які виконують найрізноманітнішу роботу, починаючи з елементарних обчислень на калькуляторі і закінчуючи суперкомп'ютерами, які шукають рішення наукових проблем, всіх їх об'єднує спільна задача над якою вони працюють – це пошук рішення та оптимізація набору параметрів, тобто в загальному розв'язання рівнянь різного вигляду та складності, наприклад розрахунок положення, траєкторії об'єктів в графічному середовищі або розв'язання системи складних рівнянь для симуляції якогось процесу, і чим ефективніше це буде зроблено, тим менші будуть витрати, а вони, в міру поширення такого типу обчислень, в наш час дуже великі, це як економічні (придбання великої кількості серверів), так і екологічні (витрата електроенергії). Можна сказати, що розв'язання рівнянь є важливою частиною математики та часто використовується в різних сферах (науковій, економічній, комп'ютерній).

Мета роботи – дослідити ефективність використання генетичного алгоритму (далі ГА) для розв'язання лінійних та нелінійних математичних рівнянь різного ступеня складності.

Встановлена мета обумовлює наступні завдання:

– проведення аналізу аналогів;

– визначення архітектури ПЗ;

– обґрунтування та вибір засобів реалізації експерименту;

– проведення дослідження;

– аналіз отриманих результатів;

– реалізація програмного продукту.

Об'єктом дослідження є лінійні та нелінійні математичні рівняння.

Предметом дослідження є застосування генетичного алгоритму як способу розв'язання математичних рівнянь.

Наукова новизна – хоча використання генетичного алгоритму досить широке, навіть в математиці та програмуванні, але ще є теми, в яких він не застосовувався або даних по їх застосуванню мало та вони не повні. Аналогічний робіт цій, в яких би ГА застосовували для розв’язання математичних рівнянь та досліджувались якісні показники (час пошуку розв’язку, величину похибки, відсоток невдач), не знайдено.

Метод дослідження – емпіричний, який включає в себе спостереження за поведінкою системи, проведення наукового експерименту, що в даній роботі є процесом виконання коду програми, вимірювання таких показників як час пошуку розв’язку, величину похибки, відсоток невдач, залежність одних параметрів ГА від інших.

В роботі було використано методи об’єктно-орієнтованого проектування, веб-програмування, функціонального програмування, математичний апарат, а саме ГА та метод найменших квадратів (далі МНК) для інтерполяції функцій, якими ілюструвався тренд.

Розроблене програмне забезпечення (далі ПЗ) для вирішення поставлених задач та при необхідності повтору експерименту.

Пояснювальна записка до дипломної роботи містить 67 сторінок, 31 ілюстрацію та 14 таблиць, список літературних джерел містить 10 найменувань.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ

1.1 Інструменти дослідження

Інструменти дослідження грають важливо роль, тому від їх вибору залежить ефективність роботи та взагалі точність результатів роботи.

Згідно поставленого технічного завдання необхідно провести аналіз доступних технологій та обрати необхідні.

Для роботи потрібні наступні інструменти:

1. Середовище програмування;
2. основна мова програмування;
3. фрейморк для створення основного ПЗ;
4. бібліотека для роботи з ГА;
5. бібліотека для візуалізації результатів;
6. інші допоміжні інструменти розробки.

Проаналізувавши існуючі варіанти для кожного пункту були вибрані наступні.

HTML (HyperText Markup Language, мова розмітки гіпертексту) − це стандарт верстки веб-сторінок, який визначає, які елементи і як повинні розташовуватися в документі. ***CSS (Cascading Style Sheets, каскадні таблиці стилів)*** − це набір параметрів форматування, який застосовується до елементів документа аби змінити їх зовнішній вигляд.

Python – інтерпретована об'єктно-рієнтована мова програмування високого рівня зі строгою динамічною типізацією. Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно орієнтована, процедурна, функціональна та аспектно-орієнтована.

JetBrains PyCharm – комерційне кроссплатформенне інтегроване середовище розробки для Python. Розробляється компанією JetBrains на основі платформи IntelliJ IDEA. PyCharm є інтелектуальним редактором для Python з можливостями аналізу коду на льоту, запобігання помилок в коді і автоматизованими засобами рефакторинга Python. Автодоповнення коду в PyCharm підтримує специфікацію Python 2,3 (сучасні та традиційні проекти), включаючи генератори, спів програми, простори імен, замикання, типажі і синтаксис коротких масивів, повноцінний SQL – редактор з можливістю редагування отриманих результатів запитів. Також існує науковий режим.

Bootstrap – це безкоштовна колекція CSS та JavaScript/jQuery з відкритим вихідним кодом, яка використовується для створення динамічного макету веб-сайтів та веб-додатків. Bootstrap — один із найпопулярніших інтерфейсних фреймворків із дійсно гарним набором зумовлених кодів CSS. Bootstrap використовує різні типи класів для створення адаптивних веб-сайтів. Bootstrap 5 був офіційно випущений 16 червня 2020 після декількох місяців перевизначення його функцій. Bootstrap – це фреймворк, який підходить для мобільної веб-розробки. це означає, що код і шаблон, доступні в початковому завантаженні, застосовуються до різних розмірів екрана. Він адаптується до будь-якого розміру екрана. Платформа безкоштовна і може використовуватися двома способами: або завантаживши zip-файли та ввімкнувши бібліотеки/модулі початкового завантаження в проект, або безпосередньо включивши URL-адресу початкового завантаження та використовуючи онлайн-версію.

Flask – це веб-фреймворк. Це означає, що flask надає вам інструменти, бібліотеки та технології, що дозволяють створювати веб-програми. Ця веб-програма може бути веб-сторінкою, блогом, вікі або може бути настільки великим, як веб-додаток календаря або комерційний веб-сайт. Flask є частиною категорій мікрофреймворку. Мікрофреймворк зазвичай є фреймворк, що практично не залежить від зовнішніх бібліотек. У цьому є плюси та мінуси. Плюси полягають у тому, що фреймворк легкий, мало залежностей для

оновлення та відстеження помилок безпеки, мінуси у тому, що деякий час вам доведеться виконувати більше роботи самостійно або збільшувати список залежностей, додаючи плагіни. У випадку з Flask його залежності такі:

- Werkzeug службова бібліотека WSGI
- jinja2, який є його шаблонізатором

Matplotlib – один із найпопулярніших пакетів Python, які використовуються для візуалізації даних. Це кросплатформова бібліотека для створення 2D-графіків із даних у масивах. Matplotlib написано на Python і використовує NumPy, розширення Python для числової математики. Він надає об'єктно-орієнтований API, який допомагає вбудовувати графіки до програм за допомогою наборів інструментів Python GUI, таких як PyQt, WxPythonotTkinter. Його також можна використовувати в оболонках Python та IPython, ноутбуках Jupyter та серверах веб-додатків. Matplotlib має процедурний інтерфейс під назвою Pylab, який нагадує MATLAB, пропрієтарну мову програмування, розроблену MathWorks. Matplotlib разом із NumPy можна як еквівалент MATLAB з відкритим вихідним кодом. Matplotlib був спочатку написаний Джоном Д. Хантером у 2003 році. Поточна стабільна версія – 2.2.0, випущена у січні 2018 року.

SciPy – вимовляється як Sigh Pi, є науковим Python з відкритим вихідним кодом, що розповсюджується під ліцензійною бібліотекою BSD для виконання математичних, наукових та інженерних обчислень. Бібліотека SciPy залежить від NumPy, який забезпечує зручну та швидку роботу з N-мірними масивами. Бібліотека SciPy призначена для роботи з масивами NumPy та надає безліч зручних та ефективних чисельних методів, таких як процедури для чисельного інтегрування та оптимізації. Разом вони працюють на всіх популярних операційних системах, швидко встановлюються та безкоштовні. NumPy та SciPy прості у використанні, але досить потужні, щоб на них могли покластися деякі провідні вчені та інженери світу. Використовувалась для побудови графіків та в інших дрібних обчисленнях.

PyGAD – проста у використанні бібліотека Python з відкритим вихідним

кодом для створення генетичного алгоритму. PyGAD підтримує широкий спектр параметрів, щоб надати користувачеві контроль над всім у своєму життєвому циклі. Це включає популяції, діапазон значень гена, тип даних гена, вибір батьків, кросовер і мутація. PyGAD розроблено як бібліотеку оптимізації загального призначення, яка дозволяє користувачеві налаштовувати функція фітнесу. Його використання складається з 3 основних кроків: збірка, функція fitness, створити екземпляр класу pygad.GA, і виклик методу pygad.GA.run(). Бібліотека підтримує навчання моделей глибокого навчання, створених або за допомогою самого PyGAD або за допомогою таких фреймворків, як Keras і PyTorch. PyGAD також знаходиться в активній розробці.

SymPy **–** це бібліотека Python для виконання символьних обчислень. Це система комп'ютерної алгебри (CAS), яку можна використовувати або як окрему програму, або як бібліотеку для інших програм. Оскільки це чиста бібліотека Python, її можна використовувати як в інтерактивному режимі, так і як програмну програму. SymPy стала популярною символічною бібліотекою для наукової екосистеми Python. SymPy має широкий спектр функцій, що застосовуються в області базової символьної арифметики, обчислення, алгебри, дискретної математики, квантової фізики і т.д. SymPy здатний форматувати результати в різних форматах, включаючи LaTeX, MathML і т.д.. Команда розробників на чолі з Ондржеєм Чертіком та Аароном Мерером опублікувала першу версію SymPy у 2007 році. Поточна версія - 1.5.1.

Ця бібліотека була вибрана як аналог, та з неї порівнювався ГА та його ефективність в швидкості розв’язання рівнянь.

1.2 Теоретичні відомості про ГА

Генетичний алгоритм — це евристика пошуку, яка базується на теорії природної еволюції Чарльза Дарвіна. Цей алгоритм відображає процес природного відбору, коли для розмноження відбираються найбільш

пристосовані особини з метою отримання нащадків наступного покоління які будуть краще за попередні.

При описі генетичних алгоритмів використовуються поняття, запозичені з генетики. Наприклад, мова йде про популяцію особин, а в якості базових понять застосовуються ген, хромосома, генотип. Також використовуються відповідні цим термінам визначення з технічного лексикону, зокрема, ланцюг, двійкова послідовність, структура.

Популяція – це кінцева множина особин. В данному випадку це набір розв'язків з певною похибкою.

Особини, що входять в популяцію, у генетичних алгоритмах представляються хромосомами з закодованими в них множинами параметрів задачі, тобто рішень, які інакше називаються точками в просторі пошуку (search points). Розв'язок рівняння, тобто невідоме x.

Хромосоми – це впорядковані послідовності генів. Представляють собою масив чисел (генів), сума яких є розв'язком рівняння.

Ген – це атомарний елемент генотипу (число), зокрема, хромосоми.

Дуже важливим поняттям у генетичних алгоритмах є **функція пристосованості** (fitness function), або функцієя оцінки. В данному випадку це 1/f(x), f(x) – рівняння яке дорівнює нулю, тобто чим ближче буде розв'язок, тим ближче значення буде до нуля, відповідно більше значення функції пристосованості. Вона являє міру пристосованості даної особини в популяції. Ця функція відіграє найважливішу роль, оскільки дозволяє оцінити ступінь пристосованості конкретних особин у популяції і вибрати з них найбільш пристосовані (тобто мають найбільші значення функції пристосованості) відповідно з еволюційним принципом виживання «найсильніших» (які найкраще пристосувалися).

На кожній ітерації генетичного алгоритму пристосованість кожної особини даної популяції оцінюється за допомогою функції пристосованості, і на цій основі створюється наступна популяція особин, що складають безліч потенційних рішень проблеми, наприклад, задачі оптимізації. Чергова

популяція в генетичному алгоритмі називається поколінням, а до новостворюваної популяції особин застосовується термін «нове покоління» або «покоління нащадків».

Основний (класичний) генетичний алгоритм (який також називається елементарним чи простим генетичним алгоритмом) складається з наступних кроків:

1. Ініціалізація початкової популяції;
2. оцінка пристосованості хромосом в популяції;
3. перевірка умови зупинки алгоритму;
4. селекція хромосом;
5. застосування генетичних операторів;
6. формування нової популяції;
7. вибір «найкращої» особи.

Блок - схема основного генетичного алгоритму зображена на рис. 1.1.

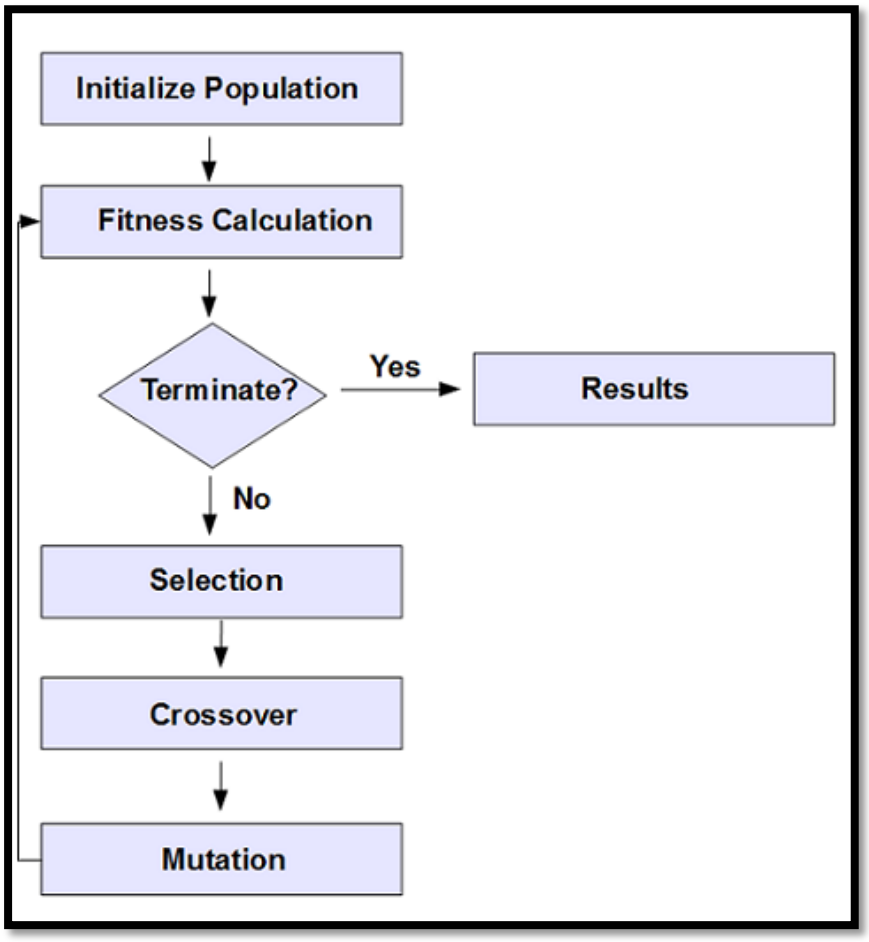

Рис. 1.1 – схема роботи генетичного алгоритму

Ініціалізація, або іншими словами формування вихідної популяції, полягає у випадковому виборі заданої кількості хромосом (особин).

Оцінювання пристосованості хромосом в популяції полягає в розрахунку функції пристосованості для кожної хромосоми цієї популяції. Чим більше значення цієї функції, тим вище «якість» хромосоми. Форма функції пристосованості залежить від характеру розв'язуваної задачі. Передбачається, що функція пристосованості завжди приймає невід'ємні значення і, крім того, що для вирішення оптимізаційної задачі потрібно максимізувати цю функцію.

Перевірка умови зупинки алгоритму. Визначення умови зупинки генетичного алгоритму залежить від його конкретного застосування. У оптимізаційних задачах, якщо відомо максимальне (або мінімальне) значення функції пристосованості, то зупинка алгоритму може відбутися після

досягнення очікуваного оптимального значення, можливо — з заданою точністю.

Алгоритм може бути зупинений після закінчення певного часу виконання або після виконання максимальної кількості ітерацій.

Селекція хромосом полягає у виборі (по розрахованим на другому етапі значеннями функції пристосованості) тих хромосом, які братимуть участь у створенні нащадків для наступної популяції, тобто для чергового покоління. Такий вибір здійснюється згідно з принципом природного добору, за яким найбільші шанси на участь у створенні нових особин мають хромосоми з найбільшими значеннями функції пристосованості.

Існують різні методи селекції. Найбільш популярним вважається так званий метод рулетки, який свою назву отримав за аналогією з відомою азартною грою. Кожній хромосомі може бути зіставлений сектор колеса рулетки, величина якого встановлюється пропорційною значенню функції пристосованості даної хромосоми. Тому чим більше значення функції пристосованості, тим більше сектор на колесі рулетки. Очевидно, що чим більше сектор, тим більше вірогідність «перемоги» відповідної хромосоми. Тому ймовірність вибору даної хромосоми виявляється пропорційною значенню її функції пристосованості.

В результаті процесу селекції створюється батьківська популяція, також звана батьківським пулом з чисельністю N, що дорівнює чисельності поточної популяції.

У класичному генетичному алгоритмі застосовуються два основних генетичних оператора: оператор схрещування (crossover) та оператор мутації (mutation). Однак слід зазначити, що оператор мутації грає явно другорядну роль в порівнянні з оператором схрещування. Це означає, що схрещування в класичному генетичному алгоритмі здійснюється практично завжди, тоді як мутація — досить рідко. Вірогідність схрещування, як правило, досить велика (звичайно $0,5 < Pc < 1$), тоді як імовірність мутації встановлюється дуже малою (найчастіше $0 < Pm < 0,1$). Це випливає з аналогії зі реальним світом живих

організмів, де мутації відбуваються надзвичайно рідко, але саме вони призводять до появи нових видів та біологічних механізмів.

Оператор схрещування. На першому етапі схрещування відбираються пари хромосом з батьківської популяції. Це тимчасова популяція, що складається з відібраних в результаті відбору хромосом і призначена для подальшого перетворення операторами схрещування і мутації для формування нової популяції нащадків. На цьому етапі хромосоми з батьківської популяції об'єднуються в пари.

Формування нової популяції. Хромосоми, отримані в результаті застосування генетичних операторів до хромосом тимчасової батьківської популяції, включаються в нову популяцію. Вона стає так званою поточною популяцією для даної ітерації генетичного алгоритму.

На кожній наступній ітерації обчислюються значення функції пристосованості для всіх хромосом цієї популяції, після чого перевіряється умова зупинки алгоритму, і або записується результат у вигляді хромосоми з найбільшим значенням функції пристосованості, або виконується наступний крок генетичного алгоритму, тобто селекція. У класичному генетичному алгоритмі вся попередня популяція хромосом замінюється новою популяцією нащадків з такою ж кількістю.

Відбір "найкращої" хромосоми. Якщо умова зупинки алгоритму виконана, необхідно вивести результат роботи, тобто представити шуканий розв'язок задачі. Найкращим рішенням є хромосома з найбільшим значенням фітнес-функції.

На закінчення слід визнати, що генетичні алгоритми успадкували властивості природного еволюційного процесу, який полягає в генетичних змінах популяцій організмів з плином часу.

Основним фактором еволюції є природний добір (тобто природний відбір), який призводить до того, що серед генетично різних особин однієї і тієї ж популяції виживають і залишають потомство тільки найбільш пристосовані до навколишнього середовища.У генетичних алгоритмах також виділяється

етап селекції, на якому з поточної популяції вибираються і включаються до батьківської популяції особини, що мають найбільші значення функції пристосованості. На наступному етапі, який іноді називається еволюцією, застосовуються генетичні оператори схрещування і мутації, що виконують рекомбінацію генів в хромосомах.

Операція схрещування – це обмін фрагментами ланцюжків між двома батьківськими хромосомами.

Операція мутації змінює значення генів у хромосомах із заданою вірогідністю способом, представленим при описі відповідного оператора. Це призводить до зміни значень у вибраних генах. Значення ймовірності мутацій, як правило, дуже мале, тому мутації піддається лише невелика кількість генів. Схрещування — це ключовиа операція генетичних алгоритмів, що визначає їх можливості та ефективність. Мутація грає більш обмежену роль. Вона вводить в популяцію деяку різноманітність і попереджає втрати, які могли б відбутися внаслідок виключення якого-небудь значимого гена в результаті схрещування, а також дає новий виток для пошуку рішення та запобігає еволюційного глухого кута.

1.3 Пошук аналогів

Для порівняння аналогів потрібно проаналізувати варіанти які на даний момент доступні для реалізації такої задачі, а саме розв’язування математичних рівнянь. Після пошуку було прийнято рішення обрати бібліотеку sympy.

Де застосовується SymPy:

1. Многочлени
2. Математичний аналіз
3. Дискретна математика
4. Матриці
5. Геометрія
6. Побудова графіків
7. Фізика

8. Статистика
9. Комбінаторика
10. Неповний перелік проектів, які використовують SymPy:
11. Cadabra: тензорна алгебра та (квантова) система теорії поля з використанням SymPy для скалярної алгебри.
12. ChemPy: пакет, корисний для хімії, написаний на Python.
13. EinsteinPy: пакет Python для символьної та чисельної загальної теорії відносності.
14. galgebra: геометрична алгебра (раніше sympy.galgebra).
15. Проект LaTeX Expression: простий набір LaTeX алгебраїчних виразів у символьній формі з автоматичною підстановкою та обчисленням результату).
16. Lcapy: експериментальний пакет Python для навчання аналізу лінійних ланцюгів.
17. OctSymPy: символічний пакет для Octave з використанням SymPy.
18. Optlang: пакет Python для вирішення задач математичної оптимізації.
19. PyDy: багатотільна динаміка в Python.
20. pyodesys: проста чисельна інтеграція систем ODE з Python.
21. QMCPACK : квантовий метод Монте-Карло C++. Sympy використовується для генерації еталонних значень модульних тестів і генерації деякого коду.
22. Квантове програмування на Python : одномірний квантовий простий гармонічний осцилятор і вентиль квантового відображення.
23. SageMath : математична система з відкритим кодом, що включає SymPy.
24. Scikit-fdiff: Дискретизація кінцевих різниць.
25. SfePy : прості кінцеві елементи у Python.
26. Spyder: наукове середовище розробки Python, Python, еквівалентний Rstudio або MATLAB; повна підтримка SymPy може бути включена

у консолях Spyder.

27. Символьне статистичне моделювання: додавання статистичних операцій до складних фізичних моделей.
28. yt : пакет Python для аналізу та візуалізації об'ємних даних (unyt , система одиниць виміру yt, використовує SymPy).

Як видно ця бібліотека є популярною та підримується розробниками, регулярно оновлюється та є досить популярною в науковій сфері. В ній є необхідний функціонал, а саме метод solve() та зручний інтерфейс до нього, який дозволяє роз’язувати лінійні ті нелінійні математичні рівняння різного ступеня складності, а це саме те що потрібно і дасть змогу порівняти швидкодію з ГА.

Висновки до розділу 1

В цьому розділі було проведено опис вибраних інструментів розробки, обгрунтовано їх вибір, наведено інформацію про їх використання, викладено опис принципу роботи ГА, його структуру, основні ключові поняття та їх значення, механізм роботи. Проведений аналіз інструментів для розв’язування математичних рівнянь та вибрано популярну python бібліотеку – SymPy, з якою в подальшому буде проводитись порівняння у швидкодії з ГА.

Як основне середовище програмування було вибрано PyCharm від компанії JetBrains, мову програмування – python, оскільки на ньому є багато готових рішень пов’язаних із цією роботою, а саме бібліотека для візуалізації отриманих результатів – Matplotlib, реалізація самого ГА із зручним інтерфейсов – PyGad, а також веб-фреймворк – Flask який є хорошим вибором при написанні невеликих веб-сервісів, що дозволяє економити час та зосередитись на інших задачах.

РОЗДІЛ 2. РОЗРОБКА ОСНОВНОГО ПЗ

2.1 Аналіз вимог для ПЗ

Роброблений локальний веб-додаток має основні цілі: проект створюється для наукової сфери та являється інструментом для розв’язання математичних рівнянь за допомогою ГА та можливістю налаштування його параметрів.

Вимоги користувачів

Користувачі мають доступ до наступних функцій:

1. Керування всіма параментрами ГА.
2. Отримання інформації про поточний результат.
3. Перегляд прогресу розв’язання рівняння.
4. Збереження результатів при поверненні на головну сторінку.

Функціональні вимоги:

1. Вільний доступ до ПЗ без необхідності попередньої реєстрації.
2. Перегляд прогресу розв’язання рівняння.
3. Можливість збереження інформації: Система повинна зберігати всю потрібну інформацію при поверненні на головну сторінку.

Нефункціональні вимоги:

1. Навчання роботи з системою:
   - час, потрібний для ознайомлення з ПЗ для звичайних користувачів – 15–20 хвилин, а для досвідчених – 5 – 10 хвилин;
   - час відповіді системи при стандартних налаштуваннях для запитів не повинен перевищувати 10 секунд на процесорі intel i7-11700F;
   - інтерфейс представлення повинен бути інтуїтивно зручним для користувача та не вимагати від нього додаткової підготовки;
   - надійність;
   - максимальна норма помилок та дефектів в роботі системи – 1 помилка на 1000 запитів користувача.

2. Можливість експлуатації:

– масштабування – система повинна мати вміти використовувати багатопоточність.

– оновлення версій – оновлення версій повинно здійснюватися в ручному режимі з втручання користувачів.

Для реалізації проекту була вибрана мова програмування Python.

2.3 Проектування та реалізація алгоритмів роботи системи

Основними модулями системи є модулі server, config, statistic, ga_interface та forms.

Після завантаження головної сторінки і встановлення параментрів за замовчуванням з модуля config (рис. 2.1) користувач може вибрати наступні дії: написати у вигляді строки рівняння у **форматі синтаксису python** з **обов'зковою невідомою x,** змінити налаштування ГА. Якщо дані форми пройшли валідацію на сторінці та за допомогою класу GAForm модуля forms, інформація передається в GAInterface де будуть проводитись подальні обчислення з використання бібліотеки PyGad.

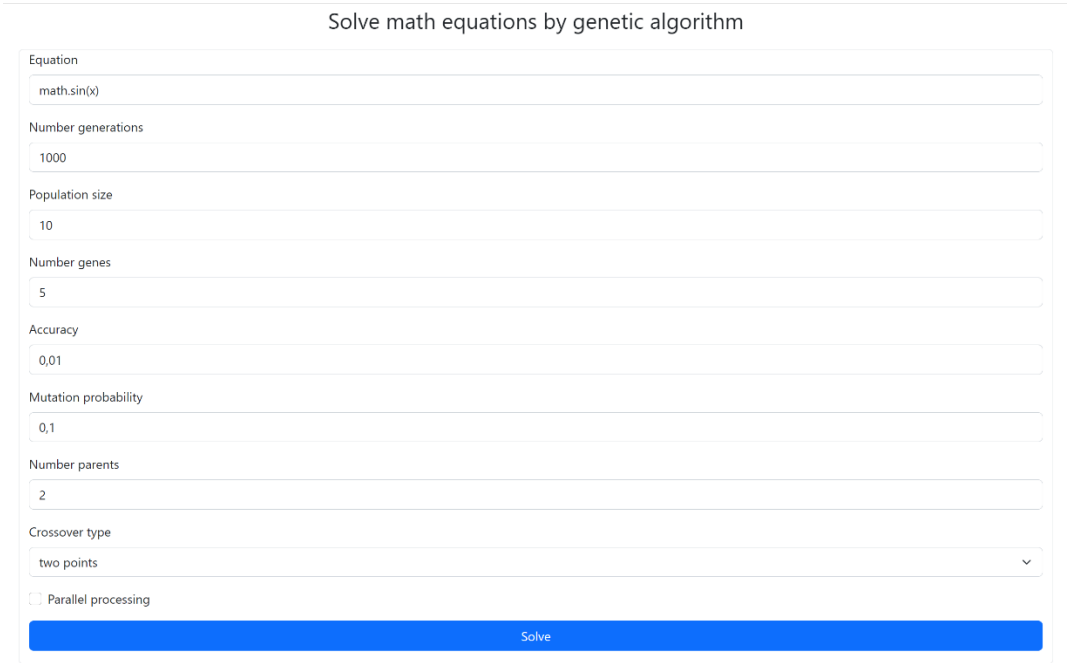

Рис. 2.1 – Головна сторінка

Під час обчислень користувач має очікувати результатів та не виконувати ніякі дії пов’язані із взаємодією із системою. Після успішного завершення обчислень його буде перенаправлено на сторінку з результатами обчислень де йому будуть доступні для читання отримані результати, які включають: розв’язок рівняння, саме рівняння, похибку, час у секундах, фітнес значення яке є оберненою величиною до похибки, налаштування ГА із головної сторінки та графік прогресу, який показує величину значення фітнес функції та кількості генерацій (рис. 2.2-2.4).

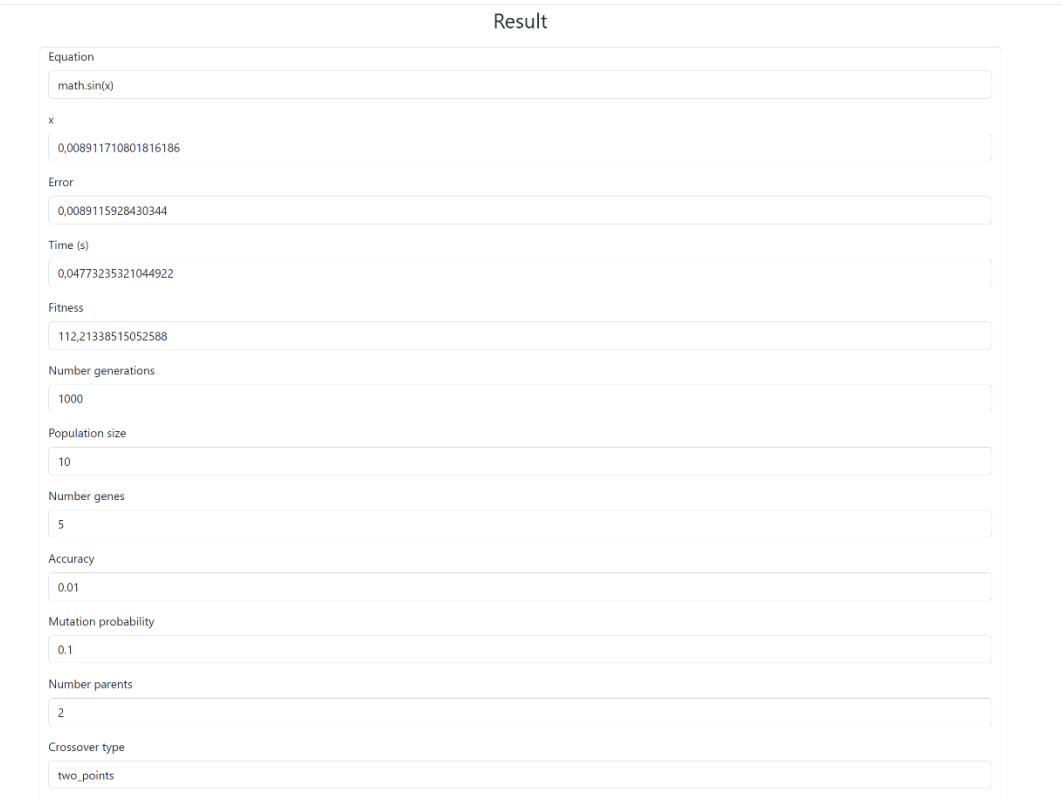

Рис. 2.2 – Сторінка з результатами

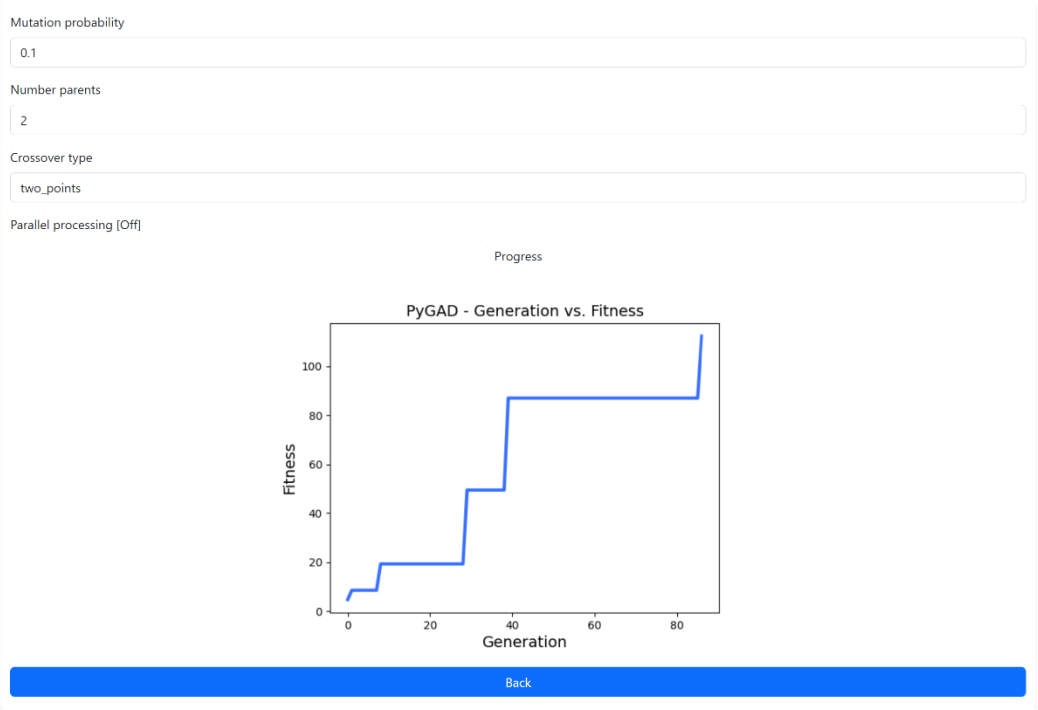

Рис. 2.3 – Сторінка з результатами (продовження)

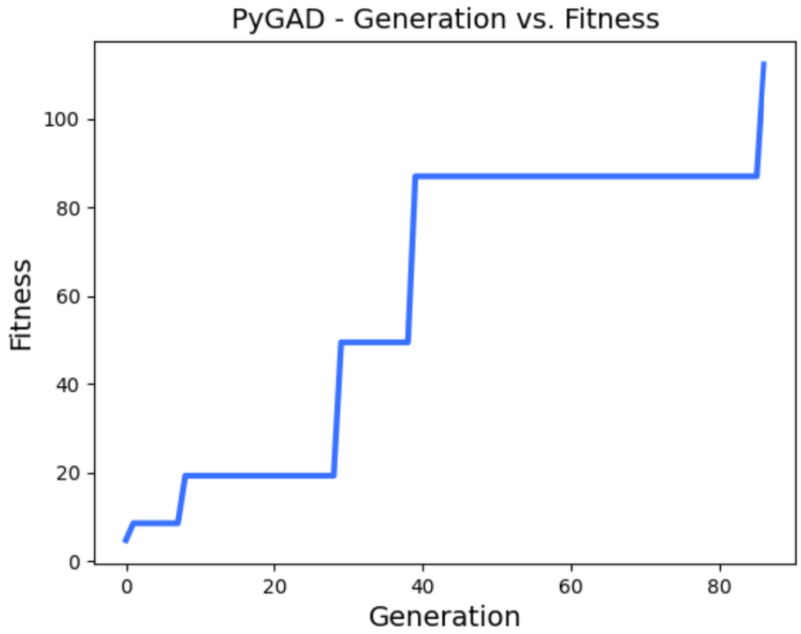

Рис. 2.4 – Графік прогресу

Проект складається з таких модулів та папок:

1. server – містить функції для обробки запитів, які надходять на сервер з браузера користувача на локальній машині.
2. config – містить клас Config для збереження констант для ГА та

проведення дослідження.

3. forms – модуль програми з єдиним класом GAForm для роботи з формою введененя в браузері на початковій сторінці.
4. ga_interface – модуль, в якому є клас GAIntarface для роботи з бібліотекою PyGad, є його обгорткою для зручності роботи.
5. Statistic – модуль для запуску екпериментy, також в ньому зберігається код для проведення досліджень. Містить клас AnalizerGA який використовується для дослідження розв'язку різних типів рівнянь. Клас AnalizerComputerAlgebra використовує модуль sympy для розв’язку рівнянь стандартними числовими методами комп’ютерної алгебри. Клас LinerEquationAnalyzerGA також використовується для дослідження залежностей основних параментрів ГА і якісних показників при розв’язанні лінійного рівняння виду ax+b=0, де a, b – дійсні числа, а x – невідоме. Також в цьому модулі містяться допоміжні функції для форматування даних та побудови графіків. Запуск експериментів можна виконувати із використанням багатопроцесорної обробки, що значно може пришвидшити обчислення.
6. tools – невеликий додатковий модуль, який містить допоміжні функції, такі як read_json_from_file для зчитування даних у форматі json з файлів з результатами експериментів та функція декоратора benchmark, який потрібен для визначення часу виконання коду.
7. Папка templates – папка для збереження шаблонів з кодом html для рендеру сторінкок
8. Папка static – зберігається зображення для показу на сторінці з результатами
9. Папка statistic – містить дві вкладені папки з назвою data та images для зберігання результатів експерименту та графіків.

Висновки до розділу 2

Було спроектовано, розроблено та протестовано програмне забезпечення

для розв’язування математичних рівнянь за допомогою ГА. Розроблене ПЗ відповідає всім поставленим вимогам та являється завершеним продуктом, готовим для використання.

Також було написано код, який не являється частиною основного ПЗ, а потрібен для проведення досліджень, він також був успішно спроектований, розроблений та протестований.

РОЗДІЛ 3. РЕЗУЛЬТАТИ ДОСЛІДЖЕНЬ

3.1 Постановка експерименту

Планування експерименту — процедура вибору числа та умов проведення дослідів, необхідних та достатніх для вирішення задачі досліджень із заданою точністю.

Розрізняють два підходи планування експерименту:

1. класичний, при якому по черзі змінюється кожен фактор до визначення часткового максимуму при постійних значеннях інших факторів,
2. статистичний, де одночасно змінюють багато факторів.

Суттєвими моментами є:

1. мінімізація числа дослідів;
2. одночасне варіювання всіма параметрами;
3. використання математичного апарата, який формалізує дії експериментатора;
4. вибір чіткої стратегії, що дозволяє приймати обґрунтовані рішення після кожної серії експериментів.

Дослідження в цій роботі можна розбити на 2 частини.

Перша – це дослідження залежності основних параметрів ГА та основних якісних показників (часу, похибки, відсотку невдач) на основі лінійного рівняння, тобто ці залежності будуть досліджуватись при розв’язанні рівняння виду a*x+b=0, a, b – дійсні числа, x – невідоме.

Друга – це порівняння з аналогом в швидкодії, в даному випадку це модуль sympy. Результатом є графіки с мінімальним, середнім та максимальним часом витраченим ГА на вирішення задачі.

Особливості проведення досліджень наступні. Кожен експеримент проводився від 100 до 1000 разів в залежності від мінімально необхідної

кількості для отримання адекватних результатів, так як ГА виконується кожного разу з іншим часом, тому потрібна велика кількість повторів для усереднення результату. Навіть при такій кількості результати можуть несуттєво відрізнятись. Чим більша буде кількість повторів експериментів, тим менше буде шуму на графіках з результатами. Для модуля sympy таких дій проводити не потрібно, в його роботі немає частин які залежать від випадковості. Обчислення зайняли 2-3 години на процесорі intel i7-11700F, що досить мало. Частина експерименту, де проводилося порівняння з аналогом, запускалась послідовно для максимізації частоти процесора. Друга частина, де досліджується залежність різних параметрів ГА, запускалась паралельно з використанням багатьох ядер процесора. Варто зазначити, що при зміні частоти процесора на 100-300МГц може впливати певним чином на результати. Дослідження проводились на одній частоті близько 4.4Ггц.

Для апроксимації та інтерполяції результатів нелінійним методом найменших квадратів (МНК) використовувались наступні функції, в залежності від даних для яких вони найкраще підходили:

1. Лінійна
2. Логарифмічна (натуральний логарифм)
3. Експоненціальна з натуральною основою
4. Поліном 4 степеня

Кожен експеримент проводився з деякими базовими налаштуваннями ГА та являв собою отримання деяких якісних показників. Вони приведені в таб. 3.1.

Таблиця 3.1 – Основні параметри ГА та експерименту

avg_time	Середній час в мілісекундах
min_time	Мінімальний час в мілісекундах
max_time	Максимальний час в мілісекундах
avg_generations_completed	Середня кількість ітерацій яка

	була потрібна для знаходження розв’язку
avg_error	Середня помилка
attempts	Кількість повторів експериментів, проведених для кожного значення
fails	Відсоток ситуацій коли алгоритм не зміг знайти розв’язок із заданою точністю
percent_fails	Відсоток невдач
fitness	Значення фітнес-функції
equation	Рівняння у текстовому виразі
x	Невідоме значення
error	Похибка відносно нуля з останнього експерименту
generations_completed	Кількість пройдених ітерацій та останього експерименту
num_generations	Максимальна кількість генерацій
num_parents_mating	Кількість родичів, які відбираються для схрещення
sol_per_pop	Величина популяції
num_genes	Кількість генів
crossover_type	Тип кросоверу
mutation_probability	Ймовірність мутації для кожного гена, набуває значень від 0 до 1 включно
parallel_processing	Кількість потоків або процесів на які розпаралелюються обчислення
accuracy	Точність якої намагається

	досягти алгоритм

3.2 Залежність парамнентів ГА та якісних показників

Далі буде показано графіки залежностей різних параменетрів ГА та якісних показників з параметрами ГА з якими вони запускались, а також деталями експерименту. Парамнти, які використовувались у дослідженні наведені у таб. 3.2-3.14, результати у вигляді графіків зображені на рис.3.1-рис.3.26.

Таблиця 3.2 – Параметри ГА при досліджені точності

attempts	1000
num_generations	1000
num_parents_mating	2
num_genes	6
sol_per_pop	20
crossover_type	two_points
mutation_probability	0.2
parallel_processing	1

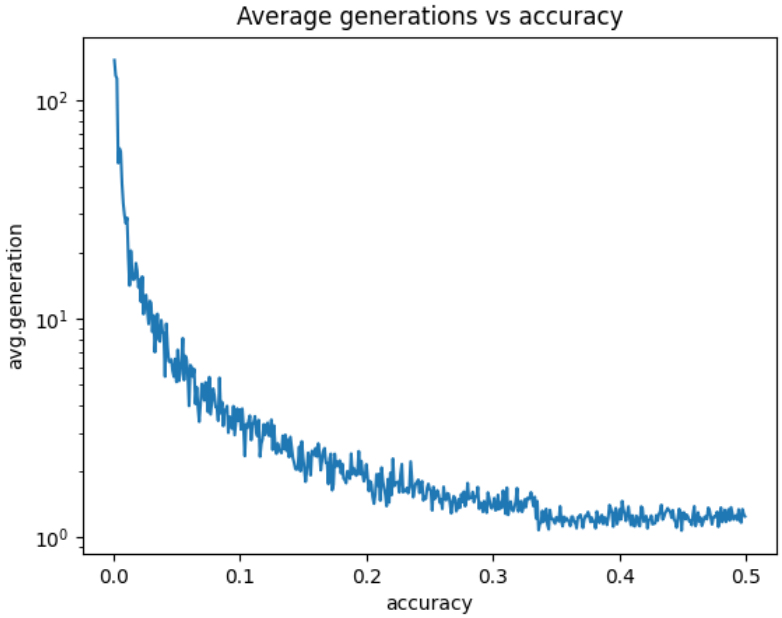

Рис. 3.1 – Залежність середньої кількості генерацій та точності

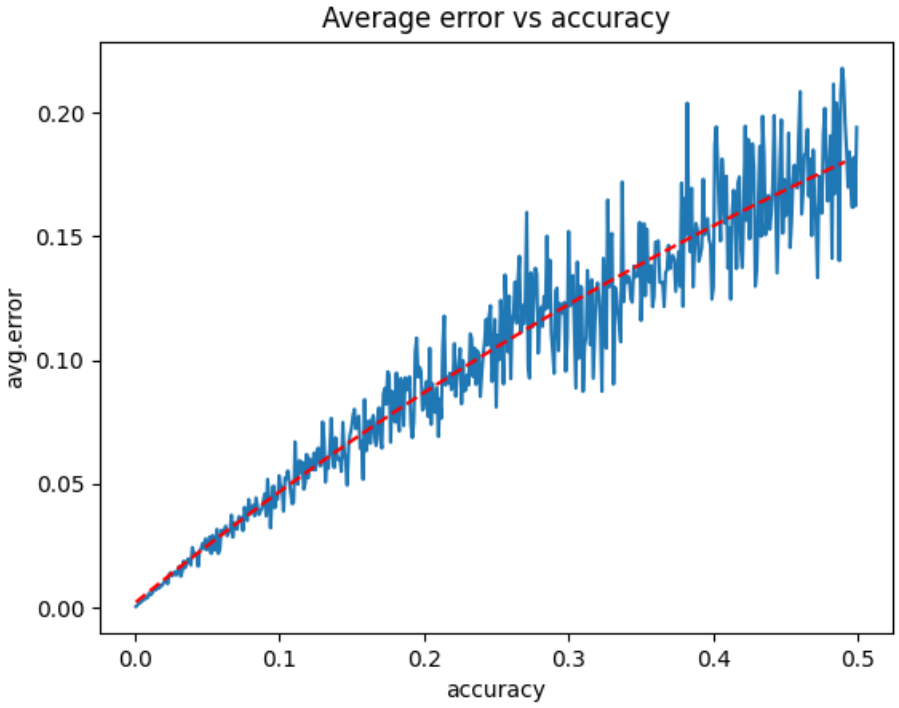

Рис. 3.2 – Залежність середньої помилки та точності

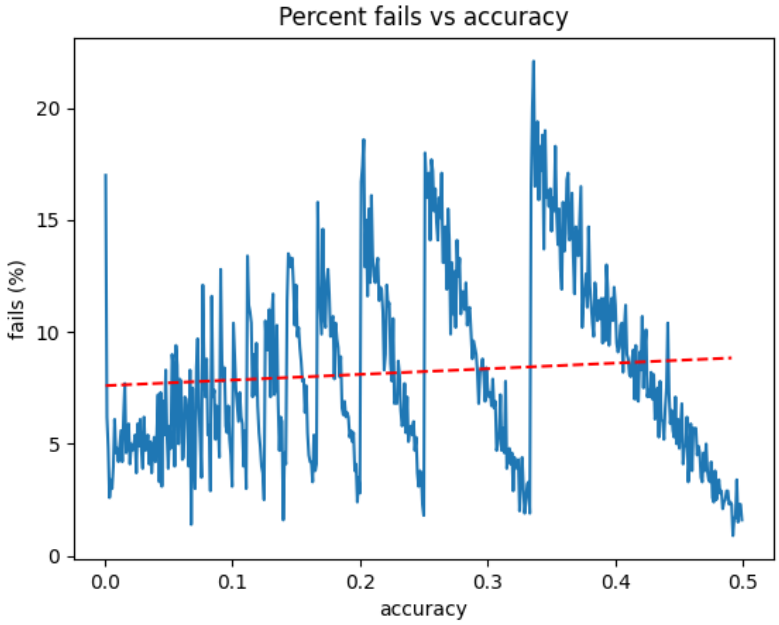

Рис. 3.3 – Залежність відсотку невдач та точності

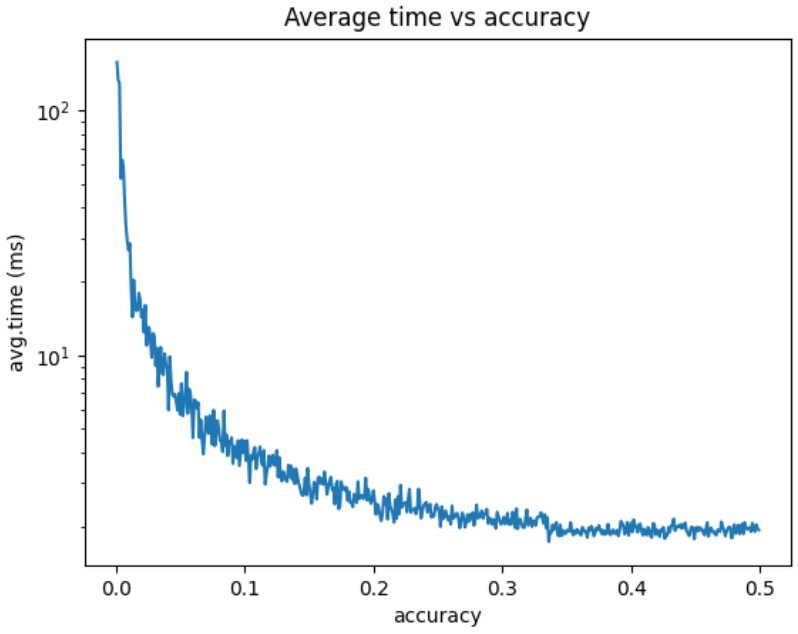

Рис. 3.4 – Залежність часу та точності

Таблиця 3.3 – Параметри ГА при досліджені типу кросоверу

attempts	100
num_generations	5000
num_parents_mating	2
num_genes	5
sol_per_pop	10
accuracy	0.01
mutation_probability	0.15
parallel_processing	1

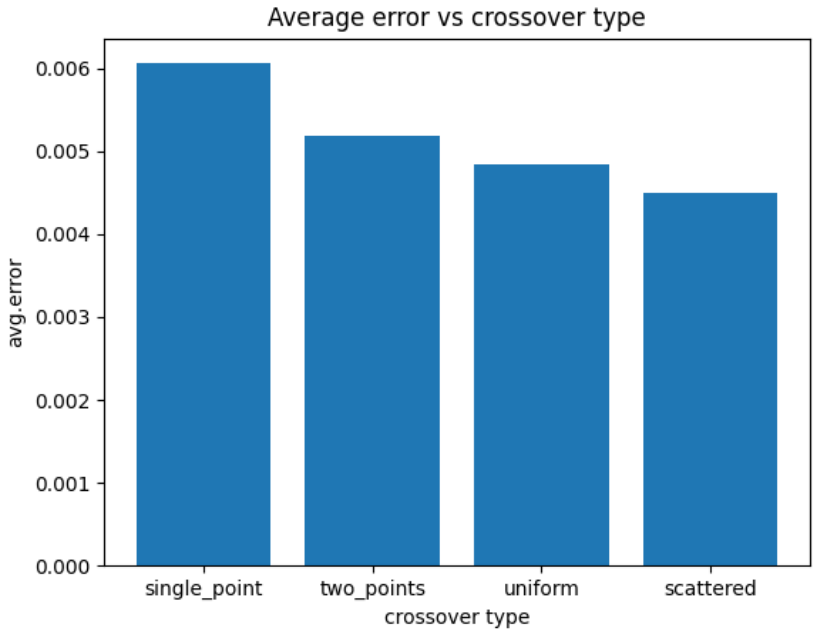

Рис. 3.5 – Залежність кросоверу та помилки

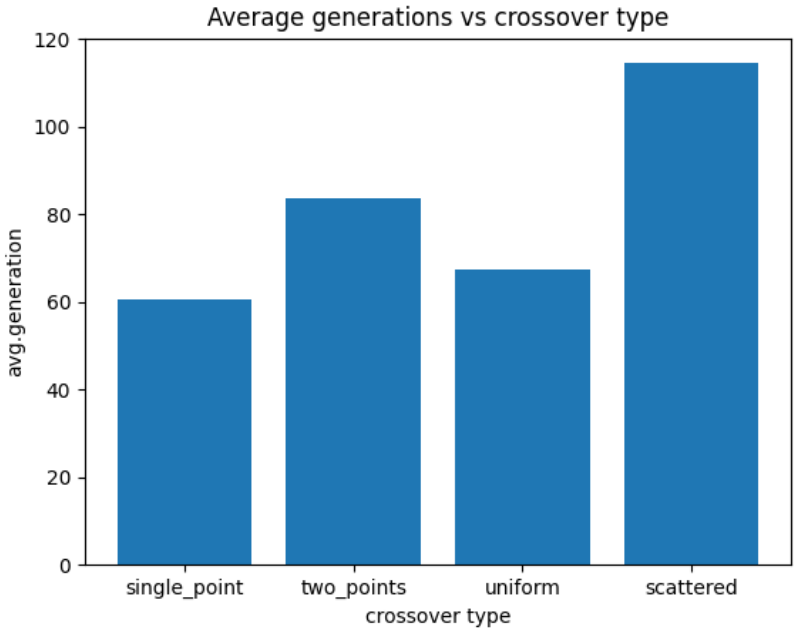

Рис. 3.6 – Залежність кросоверу та к-ті генерацій

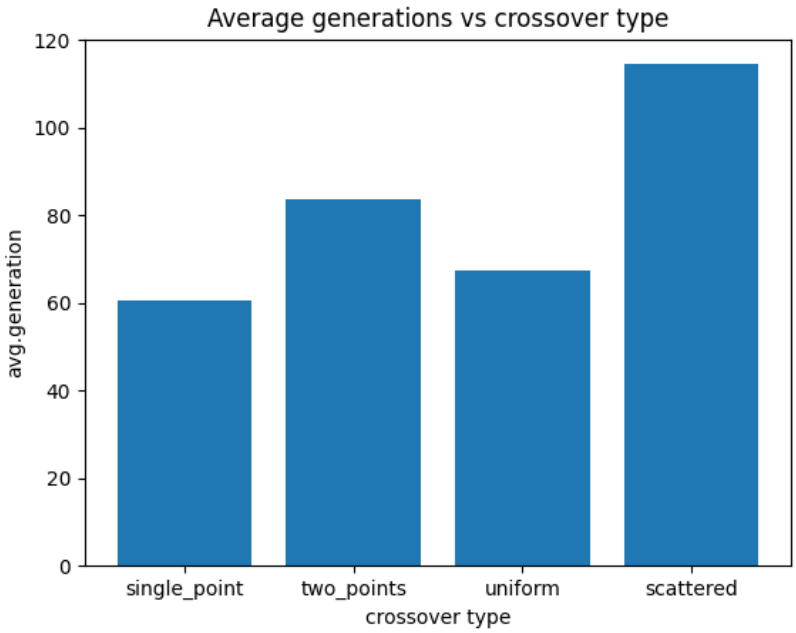

Рис. 3.7 – Залежність кросоверу та к-ті генерацій

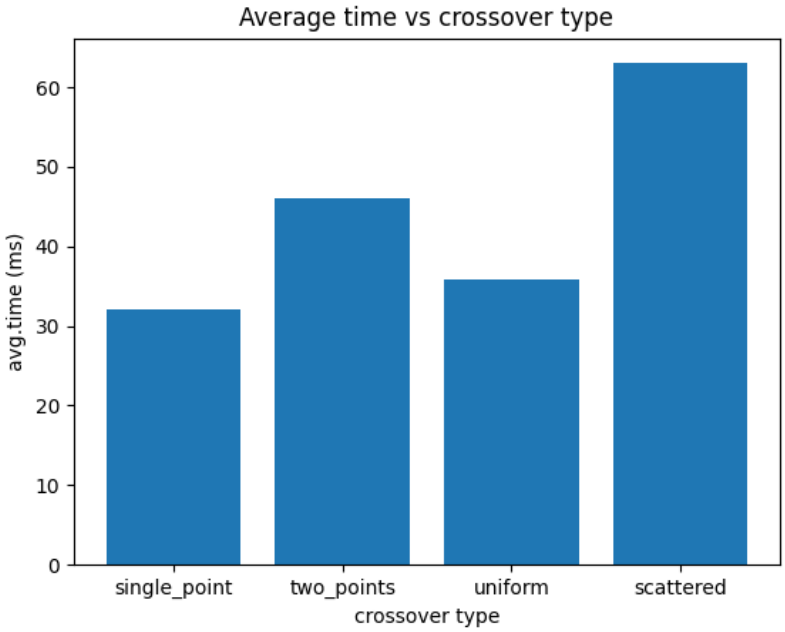

Рис. 3.8 – залежність кросоверу та часу

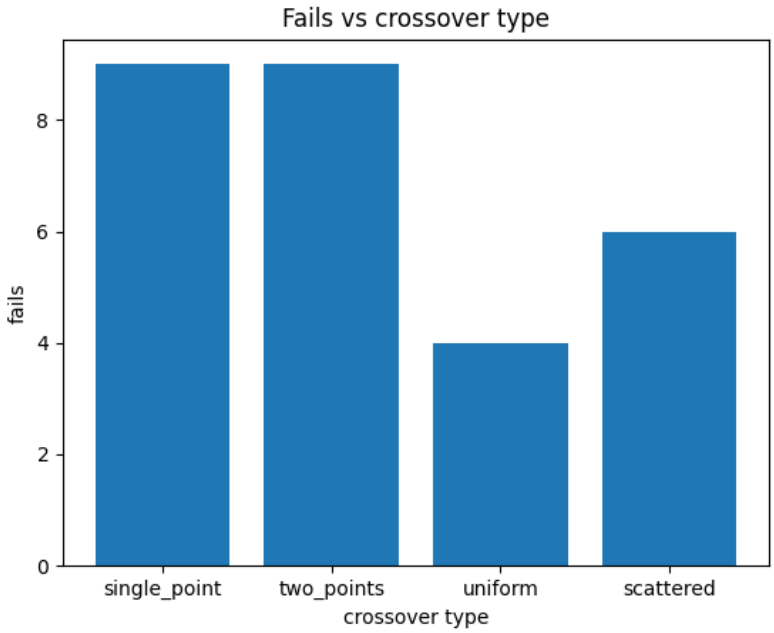

Рис. 3.9 – Залежність кросоверу та відсотку помилок

Таблиця 3.4 – Параметри ГА при досліджені к-ті генерацій

attempts	100
crossover_type	single_point

num_parents_mating	2
num_genes	6
sol_per_pop	20
accuracy	0.005
mutation_probability	0.1
parallel_processing	1

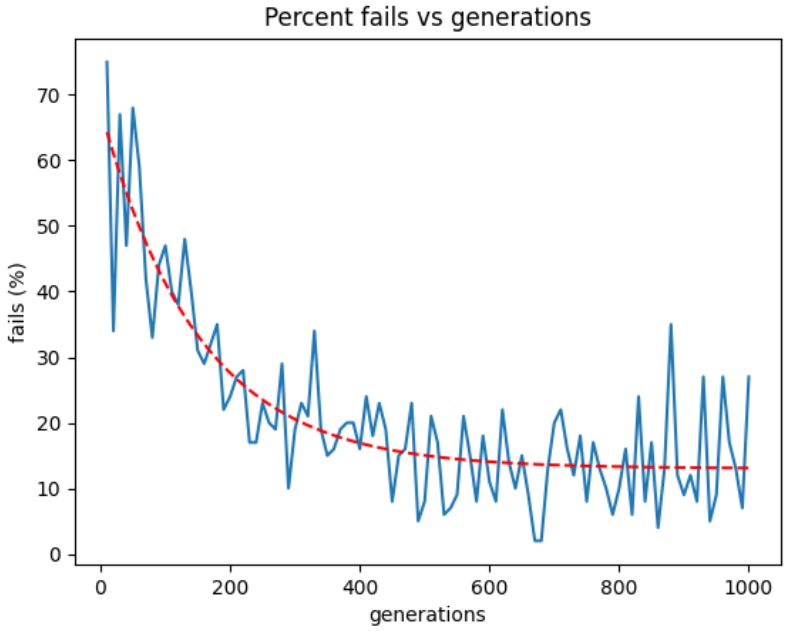

Рис. 3.10 – Залежність к-сті генерацій та відсотку помилок

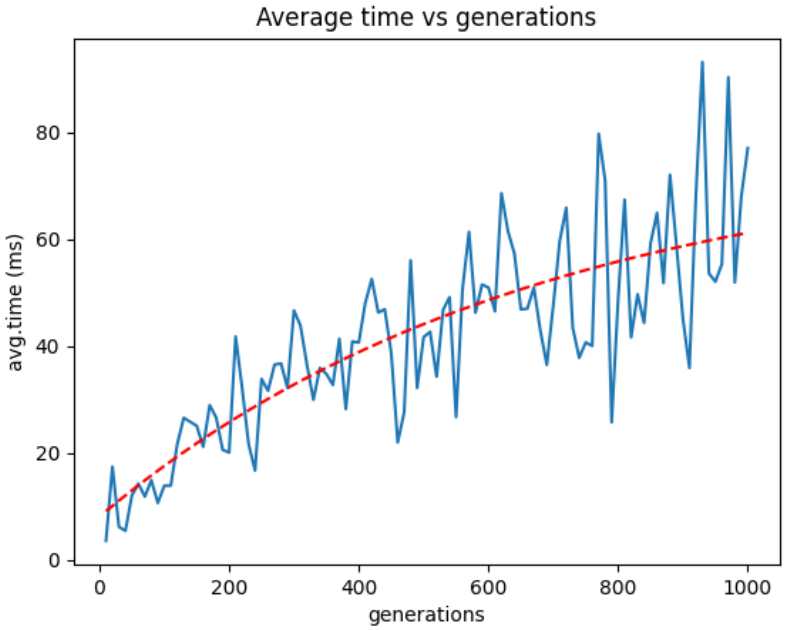

Рис. 3.11 – Залежність к-сті генерацій та часу

Таблиця 3.5 – Параметри ГА при досліджені величини мутації

attempts	100
crossover_type	two_points
num_parents_mating	2
num_genes	4
sol_per_pop	10
accuracy	0.01
num_generations	1000
parallel_processing	1

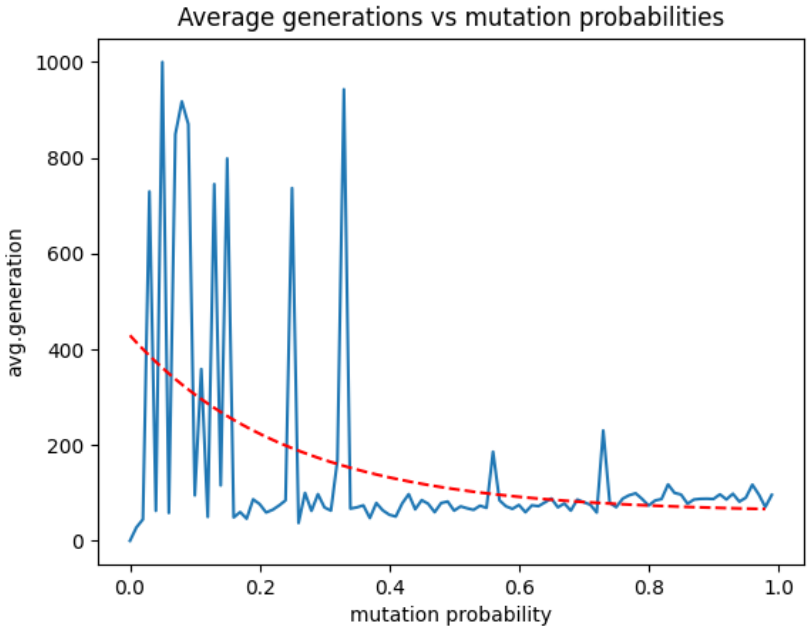

Рис. 3.12 – Залежність к-сті генерацій та мутації

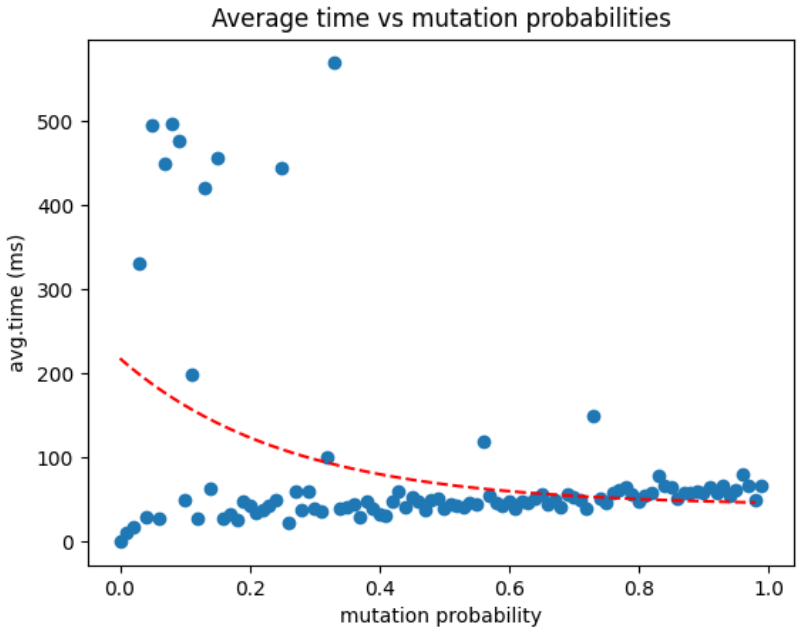

Рис. 3.13 – Залежність к-сті генерацій та часу

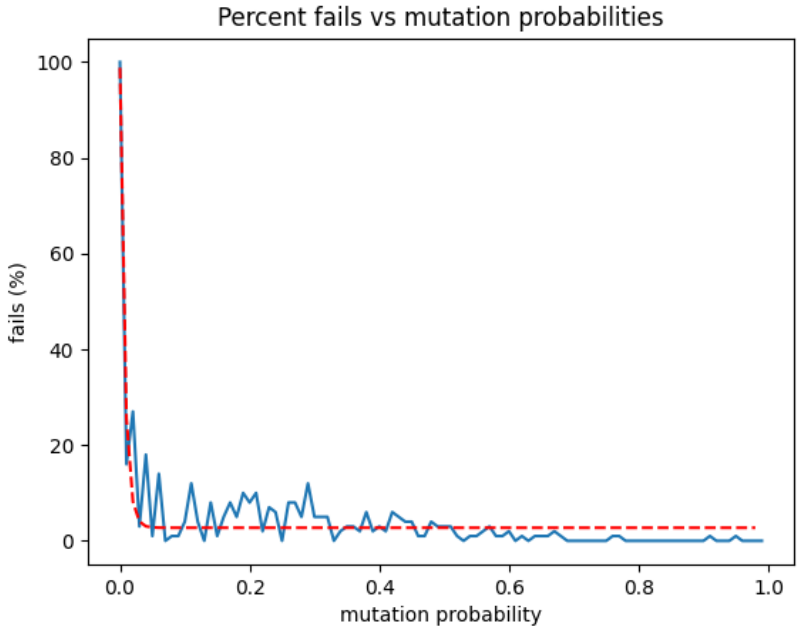

Рис. 3.14 – Залежність к-сті генерацій та відсотку помилок

Таблиця 3.6 – Параметри ГА при досліджені к-сті генів

attempts	1000
crossover_type	single_point
num_parents_mating	2
num_generations	1000
sol_per_pop	20
accuracy	0.01
parallel_processing	1
mutation_probability	0.15

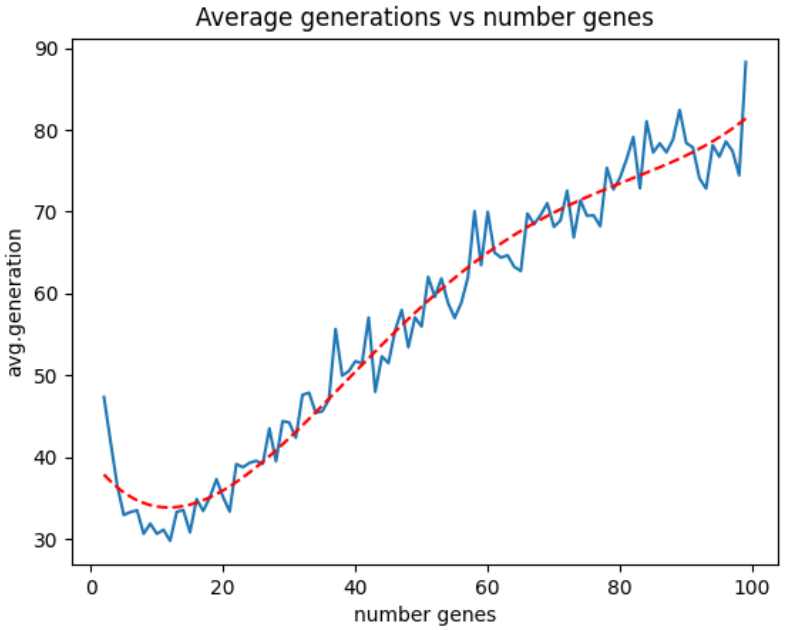

Рис. 3.15 – Залежність к-сті генів та к-сті генерацій

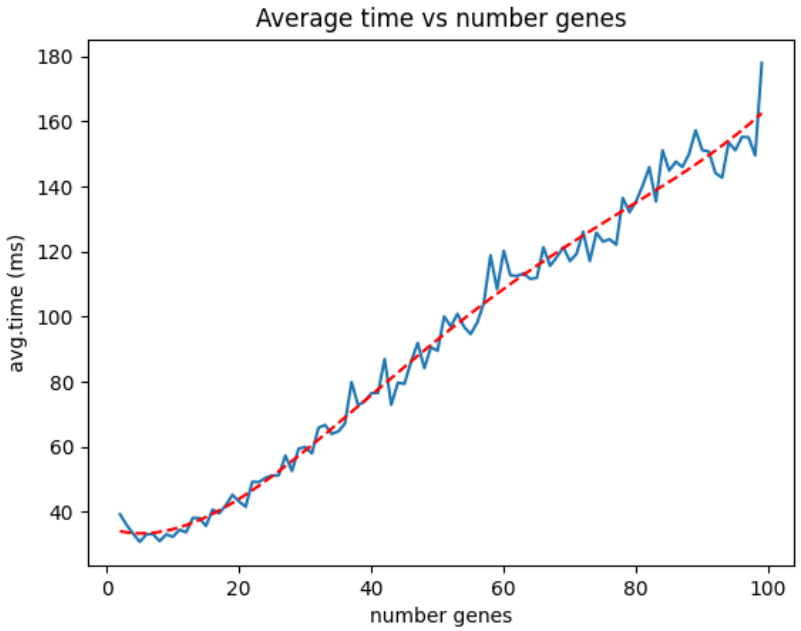

Рис. 3.16 – залежність к-сті генів та часу

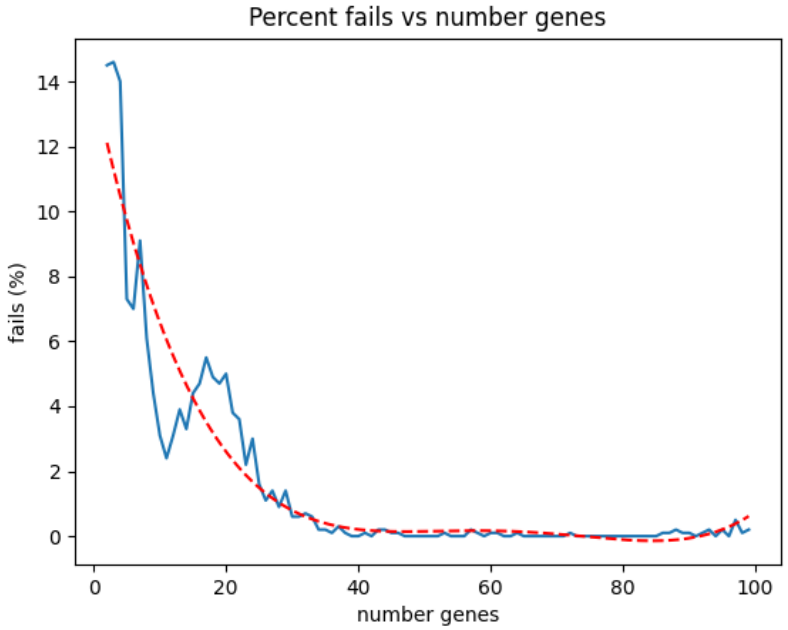

Рис. 3.17 – Залежність к-сті генів та відсотку невдач

Таблиця 3.7 – Параметри ГА при досліджені розміру популяції

attempts	1000
crossover_type	uniform
num_parents_mating	2
num_generations	1000
accuracy	0.01
num_genes	4
parallel_processing	1
mutation_probability	0.1

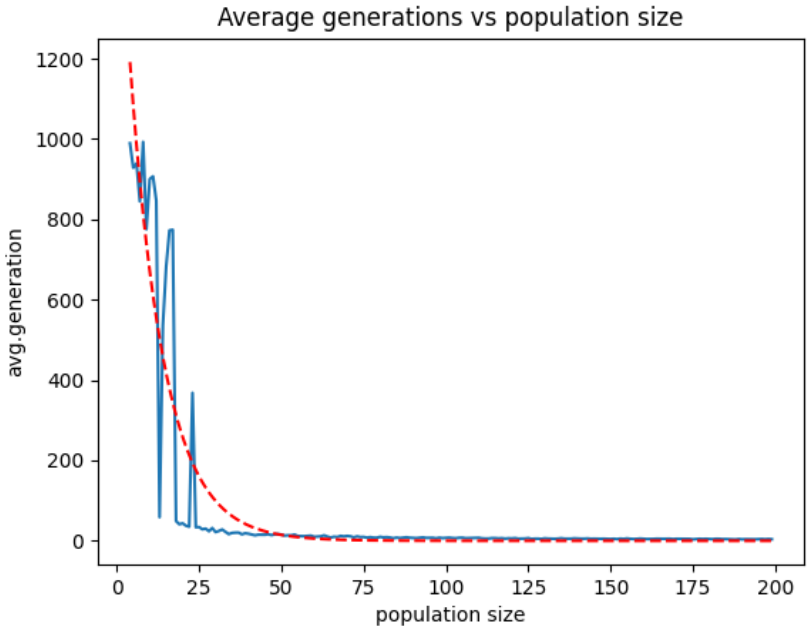

Рис. 3.18 – Залежність розміру популяції та к-сті генерацій

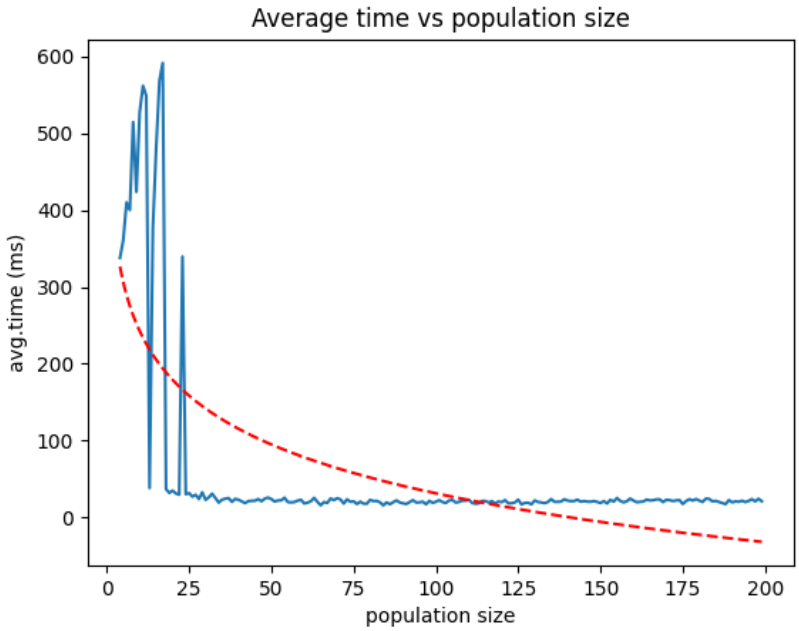

Рис. 3.19 – Залежність розміру популяції та часу

3.3 Порівняння швидкодії з аналогом

Нижче зображені результати порівняння швидкодії ГА та аналога sympy.

Таблиця 3.8 – Параметри ГА при показникового рівняння

attempts	500
crossover_type	single_point
num_parents_mating	2
num_generations	1000
sol_per_pop	20
accuracy	0.1
parallel_processing	1
mutation_probability	0.3
num_genes	2
equation	$5 * e^x$-10=0

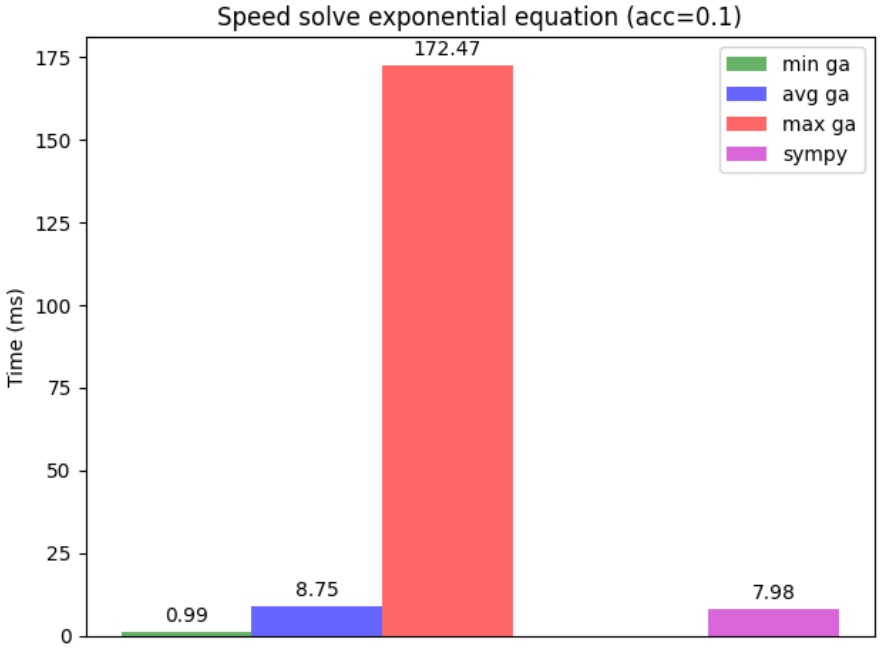

Рис. 3.20 – Швидкість розв’язку показникового рівняння

Таблиця 3.9 – Параметри ГА при розв’язанні лінійного рівняння

attempts	500
crossover_type	single_point

num_parents_mating	2
num_generations	1000
sol_per_pop	20
accuracy	0.1
parallel_processing	1
mutation_probability	0.3
num_genes	2
equation	5*x-3

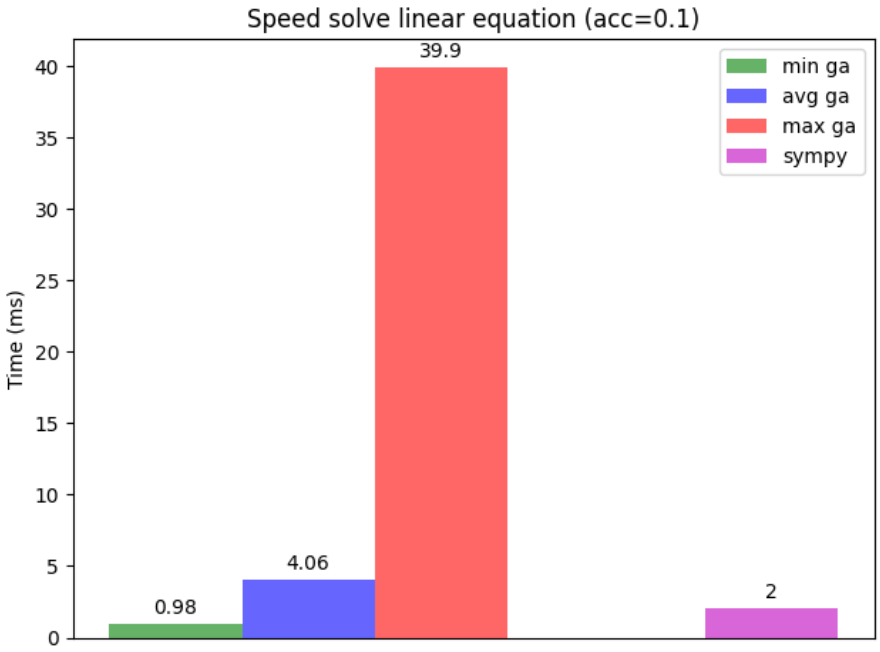

Рис. 3.21 – швидкість розв’язку лінійного рівняння

Таблиця 3.10 – Параметри ГА при розв’язанні рівняння з коренем

attempts	500
crossover_type	two_points
num_parents_mating	2
num_generations	1000
sol_per_pop	20

accuracy	0.1
parallel_processing	1
mutation_probability	0.15
num_genes	10
equation	$4 \cdot \sqrt{2 \cdot x} - 50 = 0$

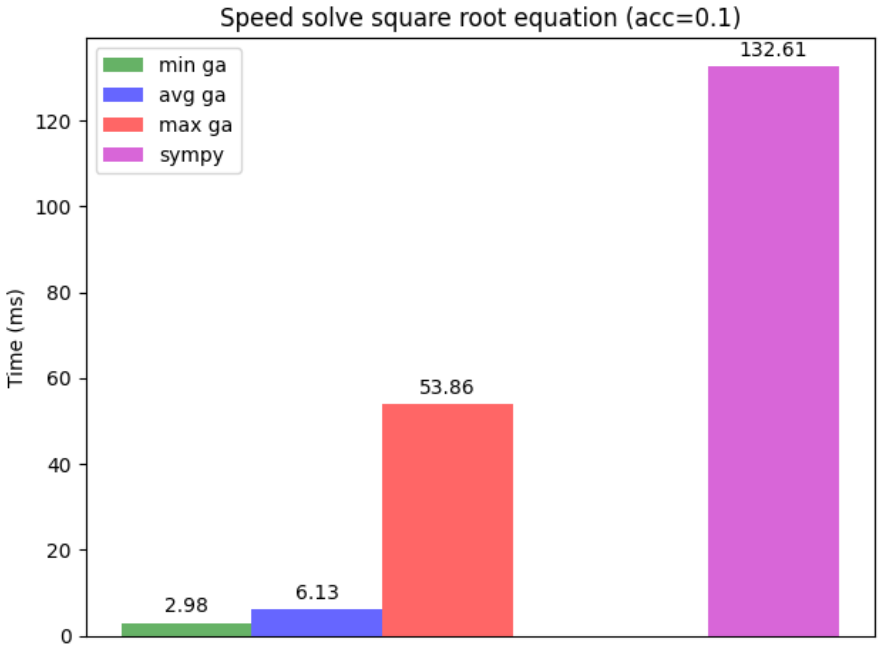

Рис. 3.22 – Швидкість розв’язку рівняння з коренем

Таблиця 3.11 – Параметри ГА при розв’язанні рівняння синусу

attempts	500
crossover_type	two_points
num_parents_mating	2
num_generations	1000
sol_per_pop	40
accuracy	0.1
parallel_processing	1

mutation_probability	0.3
num_genes	4
equation	4*sin(-2*x)+2=0

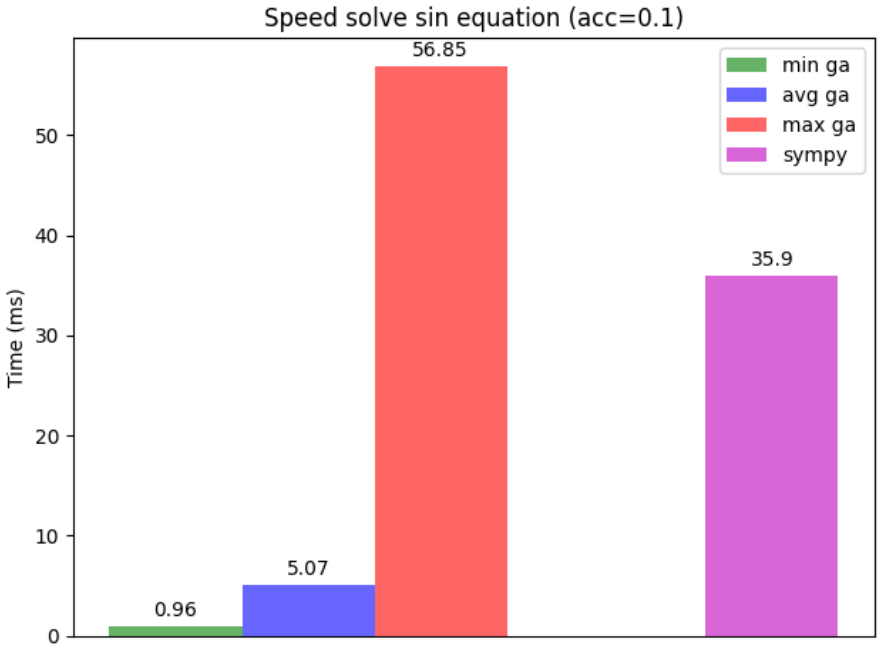

Рис. 3.23 – Швидкість розв’язку рівняння синусу

Таблиця 3.12 – Параметри ГА при розв’язанні рівняння косинусу

attempts	500
crossover_type	single_point
num_parents_mating	2
num_generations	1000
sol_per_pop	20
accuracy	0.1
parallel_processing	1
mutation_probability	0.3
num_genes	3
equation	7*cos(2*x)+3=0

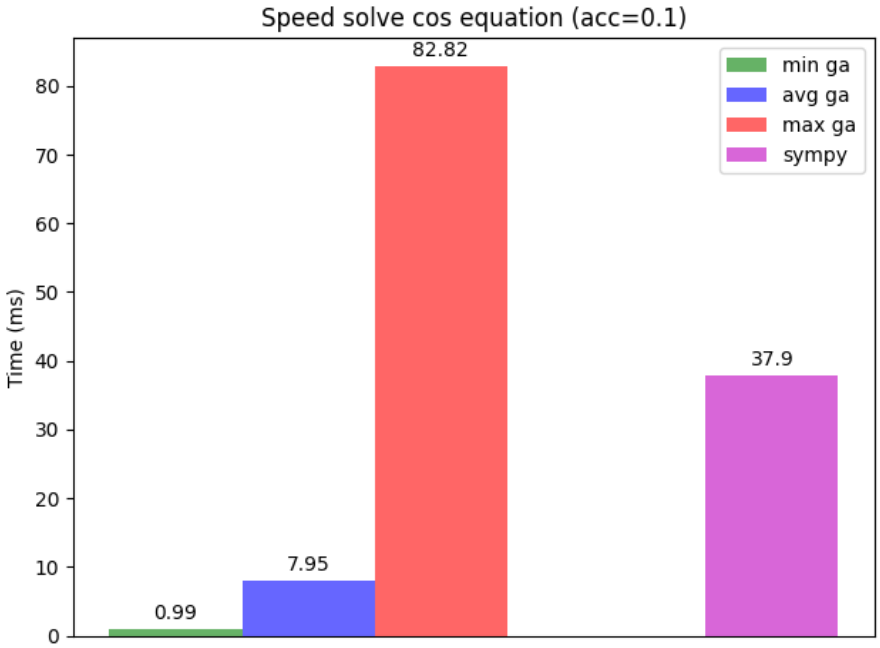

Рис. 3.24 – Швидкість розв’язку рівняння синусу

Таблиця 3.13 – Параметри ГА при розв’язанні рівняння тангенсу

attempts	500
crossover_type	single_point
num_parents_mating	2
num_generations	1000
sol_per_pop	20
accuracy	0.1
parallel_processing	1
mutation_probability	0.3
num_genes	2
equation	-5*tg(3*x)-2=0

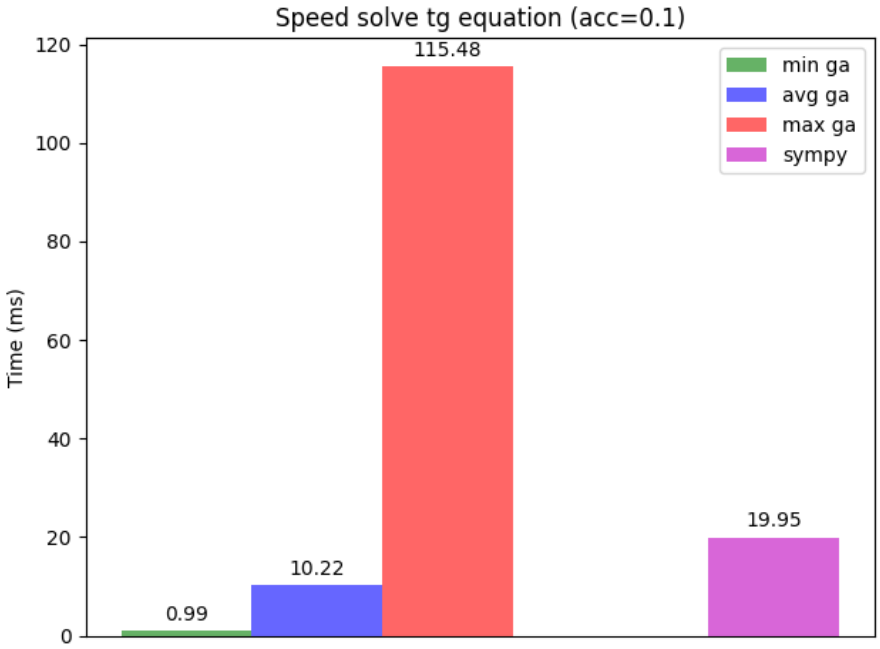

Рис. 3.25 – Швидкість розв’язку рівняння тангенсу

Таблиця 3.14 – Параметри ГА при розв’язанні рівняння котангенсу

attempts	500
crossover_type	single_point
num_parents_mating	2
num_generations	1000
sol_per_pop	20
accuracy	0.1
parallel_processing	1
mutation_probability	0.3
num_genes	2
equation	4*ctg(8*x)-10=0

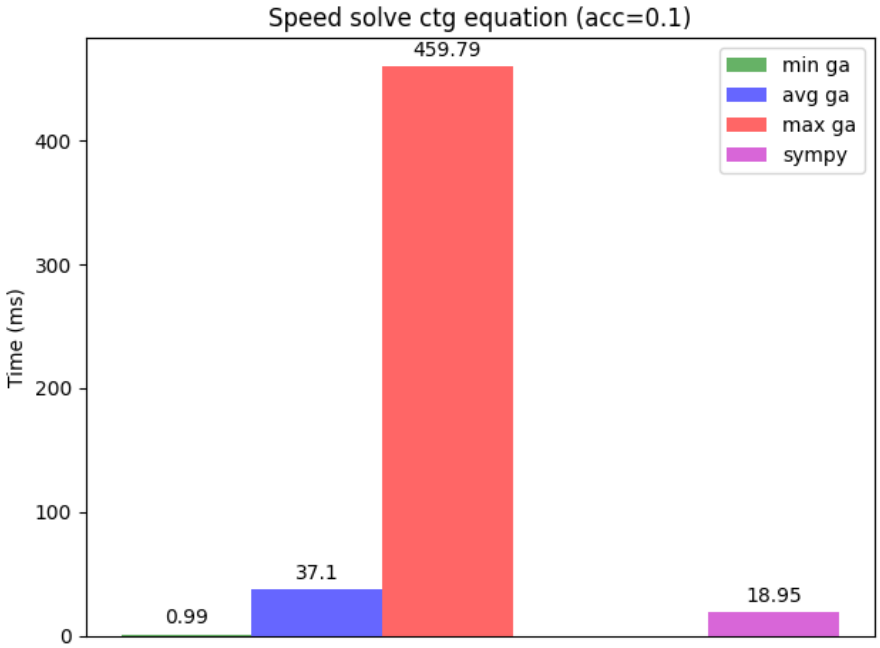

Рис. 3.26 – Швидкість розв’язку рівняння котангенсу

Висновки до розділу 3

В 3 розділі було продемонстровано результати досліджень, показано залежності між різнимим параментрами ГА та основними якісними показниками. Також було для прикладу взято деякі рівняння з випадковими коефіцієнтами, які були розв’язані за допомогою ГА та порівнянні у швидкодії із аналогом – sympy із заданою точністю.

Висновки

Проаналізовано отримані результати, на основі яких можна сказати, що ГА можна застосовувати для деяких рівнянь, коли не потрібна висока точність.

Ефективність застосування ГА для таких задач сильно залежить від початкових налаштувань алгоритму. ГА може бути використаний для наближеного розв’язання рівнянь як альтернатива відомим числовим методам. Швидкодія також залежить від коефіцієнтів. Наведені результати показуються тільки один варіант коефіцієнтів, не можна сказати що отримані результати будуть такі ж при інших числах.

Список використаних джерел

Електронні ресурси:

Віддаленого доступу:

1. Документація по python 3 [Електронний ресурс] https://docs.python.org/3.9/tutorial/
2. Документація по веб-фреймворку flask [Електронний ресурс] https://flask.palletsprojects.com/en/2.2.x/
3. Документація по бібліотеці для візуалізації даних Matplotlib [Електронний ресурс] https://matplotlib.org/
4. Документація по бібліотеці для роботи з комп'ютерною алгеброю sympy [Електронний ресурс] https://docs.sympy.org/latest/install.html
5. Документація по бібліотеці яка реалізує генетичний алгоритм Pygad [Електронний ресурс] https://pygad.readthedocs.io/en/latest/
6. Загально про роботу генетичного алгоритму [Електронний ресурс] https://en.wikipedia.org/wiki/Genetic_algorithm
7. Документація по бібліотеці scipy [Електронний ресурс] https://docs.scipy.org/doc/scipy/getting_started.html#getting-started-ref
8. Документація по бібліотеці яка реалізує генетичний алгоритм Pygad [Електронний ресурс] https://pygad.readthedocs.io/en/latest/
9. Метод найменших квадратів [Електронний ресурс] https://en.wikipedia.org/wiki/Least_squares#Non-linear_least_squares
10. Детально про генетичний алгоритм [Електронний ресурс] http://www.znannya.org/?view=ga_general

Тези конференцій:

XI щорічна наукова конференція «Наукові підсумки 2022» (20.12.2022) [Електронний ресурс] https://entc.com.ua/download/Збірник тез 11 Наукової конференції_НАУКОВІ ПІДСУМКИ 2022 РОКУ_.pdf

ДОДАТОК А. Код програми

config.py

```
import os


class Config:
 SECRET_KEY = os.getenv('SECRET_KEY', 'SECRET_KEY')
 NUM_GENERATIONS = 1000
 NUM_PARENTS_MATING = 2
 NUM_GENES = 5
 SOL_PER_POP = 10
 CROSSOVER_TYPE = 'two_points'
 MUTATION_PROBABILITY = 0.15
 PARALLEL_PROCESSING = 1
 ACCURACY = 0.01
 ATTEMPTS = 100
 PATH_TO_STATISTIC_DATA = 'statistics/data/'
 PATH_TO_STATISTIC_IMG = 'statistics/images/'
```

forms.py

```
from wtforms import Form, BooleanField, StringField, IntegerField, FloatField, validators


class GAForm(Form):
 equation = StringField(validators=[validators.DataRequired(), validators.Length(min=1)])
 num_generations = IntegerField(validators=[validators.DataRequired(), validators.NumberRange(min=0)])
 sol_per_pop = IntegerField(validators=[validators.DataRequired(), validators.NumberRange(min=1)])
 num_genes = IntegerField(validators=[validators.DataRequired(), validators.NumberRange(min=0)])
 accuracy = FloatField(validators=[validators.DataRequired()])
 mutation_probability = FloatField(validators=[validators.DataRequired(), validators.NumberRange(min=0, max=1)])
 num_parents_mating = IntegerField(validators=[validators.DataRequired()])
 crossover_type = StringField(validators=[validators.DataRequired()])
 parallel_processing = BooleanField()
```

ga_interface.py

```
import pygad, math
from tools import benchmark


class GAInterface:
 def __init__(self, equation):
 self.equation = equation # equation should = 0
 self.ga_instance = None
 self.kwargs = None

 def build_solver(self, **kwargs):
 self.kwargs = kwargs

 def fitness_func(solution, solution_idx):
 x = sum(solution) # x use in eval(equation)
 func_val = eval(self.equation)
 fitness_val = 1 / func_val + 0.00000000001
 return fitness_val

 accuracy = kwargs.pop('accuracy')
 stop_criteria = 'reach_' + str(int(1 / accuracy))
 self.ga_instance = pygad.GA(**kwargs,
 fitness_func=fitness_func,
 stop_criteria=stop_criteria)
 self.kwargs['accuracy'] = accuracy

 @benchmark
 def run_solver(self):
 self.ga_instance.run() if self.ga_instance else None

 def get_result(self):
 solution, sol_fitness, sol_idx = self.ga_instance.best_solution(self.ga_instance.last_generation_fitness)
 x = sum(solution)
 return {
 'x': x,
 'fitness': sol_fitness,
 'equation': self.equation,
 'error': eval(self.equation),
 'generations_completed': self.ga_instance.generations_completed,
 **self.kwargs
```

```
 }

 def get_progress_figure(self):
 return self.ga_instance.plot_fitness()
```

server.py

```
import os

from flask import Flask, render_template, request, redirect, url_for, session
from config import Config
from forms import GAForm
from ga_interface import GAInterface

app = Flask(__name__)
app.config['SECRET_KEY'] = Config.SECRET_KEY


@app.get('/')
def get_main_page():
 view_data = {
 'num_generations': Config.NUM_GENERATIONS,
 'num_parents_mating': Config.NUM_PARENTS_MATING,
 'num_genes': Config.NUM_GENES,
 'sol_per_pop': Config.SOL_PER_POP,
 'crossover_type': Config.CROSSOVER_TYPE,
 'mutation_probability': Config.MUTATION_PROBABILITY,
 'parallel_processing': Config.PARALLEL_PROCESSING,
 'accuracy': Config.ACCURACY
 }
 return render_template('index.html', data=view_data)


@app.get('/result')
def get_result_page():
 data = session['result']
 print(data)
 return render_template('result.html', data=data)


@app.route('/process_data', methods=['POST'])
def process_data():
 ga_form = GAForm(request.form)
 if not ga_form.validate():
 return redirect(url_for('get_main_page'))
 ga_data = ga_form.data
 ga = GAInterface(ga_data.pop('equation'))
 ga_data['parallel_processing'] = os.cpu_count() if ga_data.pop('parallel_processing') else 1
 if Config.PARALLEL_PROCESSING > 1 and ga_data['parallel_processing'] > 1:
 ga_data['parallel_processing'] = Config.PARALLEL_PROCESSING
 ga.build_solver(**ga_data)
 execution_time = ga.run_solver()
 result_data = ga.get_result()
 ga.get_progress_figure().savefig('static/my_plot.png')
 result_data['execution_time'] = execution_time
 session['result'] = result_data
 return redirect(url_for('get_result_page'))


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

statistic.py

```
import os
import numpy as np
import json
import sympy
from multiprocessing import Process
from sympy import Symbol, solve, Pow, exp, cos, tan
from ga_interface import GAInterface
from tools import benchmark, read_json_from_file
from numpy import arange
from scipy.optimize import curve_fit
from matplotlib import pyplot
from config import Config


class AnalyserGA:
```

```
def __init__(self, attempts, path_to_result):
self.attempts = attempts
self.result = {}
self.path_to_result = path_to_result

def linear_equation(self, a, b, save=False):
equation = f'{a}*x+{b}'
ga = GAInterface(equation)
ga.build_solver(num_generations=1000,
num_parents_mating=2,
sol_per_pop=20,
num_genes=2,
accuracy=0.1,
crossover_type='single_point',
mutation_probability=0.3,
parallel_processing=1)
result = self._run(ga)
self.result['linear_equation'] = result
if save:
self.save(self.path_to_result + 'linear_equation.json', result)

def sqrt_x(self, a, b, c, save=False):
equation = f'{a}*math.sqrt(x*{b})+{c}'
ga = GAInterface(equation)
ga.build_solver(num_generations=1000,
num_parents_mating=2,
sol_per_pop=50,
num_genes=10,
accuracy=0.1,
gene_space={"low": 1, "high": 30},
crossover_type='two_points',
mutation_probability=0.15,
parallel_processing=1)
result = self._run(ga)
self.result = result
if save:
self.save(self.path_to_result + 'sqrt_x.json')

def polynomial_2(self, a, b, c, save=False):
equation = f'{a}*(x**2)+{b}*x+{c}'
ga = GAInterface(equation)
ga.build_solver(num_generations=1000,
num_parents_mating=2,
sol_per_pop=50,
num_genes=10,
accuracy=0.1,
crossover_type='two_points',
mutation_probability=0.2,
parallel_processing=1)
result = self._run(ga)
self.result['polynomial_2'] = result
if save:
self.save(self.path_to_result + 'polynomial_2.json', result)

def polynomial_3(self, a, b, c, d, save=False):
equation = f'{a}*(x**3)+{b}*(x**2)+{c}*x+{d}'
ga = GAInterface(equation)
ga.build_solver(num_generations=1000,
num_parents_mating=3,
sol_per_pop=50,
num_genes=10,
accuracy=0.1,
crossover_type='two_points',
mutation_probability=0.2,
parallel_processing=1)
result = self._run(ga)
self.result['polynomial_3'] = result
if save:
self.save(self.path_to_result + 'polynomial_3.json', result)

def polynomial_4(self, a, b, c, d, e, save=False):
equation = f'{a}*(x**4)+{b}*(x**3)+{c}*(x**2)+{d}*x+{e}'
ga = GAInterface(equation)
ga.build_solver(num_generations=1000,
num_parents_mating=3,
sol_per_pop=50,
num_genes=10,
accuracy=0.1,
crossover_type='two_points',
```

```
mutation_probability=0.2,
parallel_processing=1)
result = self._run(ga)
self.result['polynomial_4'] = result
if save:
self.save(self.path_to_result + 'polynomial_4.json', result)

def polynomial_5(self, a, b, c, d, e, f, save=False):
equation = f'{a}*(x**5)+{b}*(x**4)+{c}*(x**3)+{d}*(x**2)+{e}*x+{f}'
ga = GAInterface(equation)
ga.build_solver(num_generations=1000,
num_parents_mating=2,
sol_per_pop=50,
num_genes=10,
accuracy=0.1,
crossover_type='two_points',
mutation_probability=0.2,
parallel_processing=1)
result = self._run(ga)
self.result['polynomial_5'] = result
if save:
self.save(self.path_to_result + 'polynomial_5.json', result)

def exponential_equation(self, a, b, c, save=False):
equation = f'{a}*(math.e**(x*{b}))+{c}'
ga = GAInterface(equation)
ga.build_solver(num_generations=1000,
num_parents_mating=2,
sol_per_pop=20,
num_genes=2,
accuracy=0.1,
crossover_type='single_point',
mutation_probability=0.3,
parallel_processing=1)
result = self._run(ga)
self.result['exponential_equation'] = result
if save:
self.save(self.path_to_result + 'exponential_equation.json', result)

def sin_x(self, a, b, c, save=False):
equation = f'{a}*math.sin({b}*x)+{c}'
ga = GAInterface(equation)
ga.build_solver(num_generations=1000,
num_parents_mating=2,
sol_per_pop=20,
num_genes=4,
accuracy=0.1,
crossover_type='two_points',
mutation_probability=0.3,
parallel_processing=1)
result = self._run(ga)
self.result['sin_x'] = result
if save:
self.save(self.path_to_result + 'sin_x.json', result)

def cos_x(self, a, b, c, save=False):
equation = f'{a}*math.cos({b}*x)+{c}'
ga = GAInterface(equation)
ga.build_solver(num_generations=1000,
num_parents_mating=2,
sol_per_pop=20,
num_genes=3,
accuracy=0.1,
crossover_type='single_point',
mutation_probability=0.3,
parallel_processing=1)
result = self._run(ga)
self.result['cos_x'] = result
if save:
self.save(self.path_to_result + 'cos_x.json', result)

def tg_x(self, a, b, c, save=False):
equation = f'{a}*math.tan({b}*x)+{c}'
ga = GAInterface(equation)
ga.build_solver(num_generations=1000,
num_parents_mating=2,
sol_per_pop=20,
num_genes=2,
accuracy=0.1,
```

```
crossover_type='single_point',
mutation_probability=0.3,
parallel_processing=1)
result = self._run(ga)
self.result['tg_x'] = result
if save:
self.save(self.path_to_result + 'tg_x.json', result)

def ctg_x(self, a, b, c, save=False):
equation = f'{a}/math.tan({b}*x)+{c}'
ga = GAInterface(equation)
ga.build_solver(num_generations=1000,
num_parents_mating=2,
sol_per_pop=20,
num_genes=2,
accuracy=0.1,
crossover_type='single_point',
mutation_probability=0.3,
parallel_processing=1)
result = self._run(ga)
self.result['ctg_x'] = result
if save:
self.save(self.path_to_result + 'ctg_x.json', result)

def save(self, path_to_result, data=None):
with open(path_to_result, 'w') as file:
file.write(json.dumps(data if data else self.result, indent=4))

def _run(self, ga):
execution_times, generations, ga_result, errors = [], [], {}, []
fails = 0
for _ in range(self.attempts):
execution_time = ga.run_solver()['execution_time']
ga_result = ga.get_result()
if ga_result.get('error') > ga_result.get('accuracy'):
fails += 1
continue
execution_times.append(execution_time)
generations.append(ga_result.pop('generations_completed'))
errors.append(ga_result.get('error'))

avg_time = sum(execution_times) / len(execution_times) if execution_times else 0
min_time = min(execution_times) if execution_times else 0
max_time = max(execution_times) if execution_times else 0
avg_generations_completed = sum(generations) / len(generations) if generations else 0
avg_error = sum(errors) / len(errors) if errors else 0
return {
'avg_time': avg_time,
'min_time': min_time,
'max_time': max_time,
'avg_generations_completed': avg_generations_completed,
'avg_error': avg_error,
'attempts': self.attempts,
'fails': fails,
'percent_fails': 100 * fails / self.attempts,
**ga_result
}


class AnalyzerComputerAlgebra:
def __init__(self):
self.x = Symbol('x')
self.result = {}

@benchmark
def _run(self, func, *args):
res_lst = func(*args)
return [float(res) for res in res_lst if complex(res).imag == 0]

def linear_equation(self, a, b):
res = self._run(solve, a * self.x + b, self.x)
self.result['linear_equation'] = res

def sqrt_x(self, a, b, c):
res = self._run(solve, a * Pow(self.x * b, 0.5) + c, self.x)
self.result['sqrt_x'] = res

def polynomial_2(self, a, b, c):
res = self._run(solve, a * Pow(self.x, 2) + b * self.x + c, self.x)
```

```
self.result['polynomial_2'] = res

def polynomial_3(self, a, b, c, d):
res = self._run(solve, a * Pow(self.x, 3) + b * Pow(self.x, 2) + c * self.x + d, self.x)
self.result['polynomial_3'] = res

def polynomial_4(self, a, b, c, d, e):
res = self._run(solve, a * Pow(self.x, 4) + b * Pow(self.x, 3) + c * Pow(self.x, 2) + d * self.x + e, self.x)
self.result['polynomial_4'] = res

def polynomial_5(self, a, b, c, d, e, f):
res = self._run(solve, a * Pow(self.x, 5) + b * Pow(self.x, 4) + c * Pow(self.x, 3) + d * Pow(self.x,
2) + e * self.x + f,
self.x)
self.result['polynomial_5'] = res

def exponential_equation(self, a, b, c):
res = self._run(solve, a * exp(b * self.x) + c, self.x)
self.result['exponential_equation'] = res

def sin_x(self, a, b, c):
res = self._run(solve, a * cos((sympy.pi / 2) - b * self.x) + c, self.x)
self.result['sin_x'] = res

def cos_x(self, a, b, c):
res = self._run(solve, a * cos(b * self.x) + c, self.x)
self.result['cos_x'] = res

def tg_x(self, a, b, c):
res = self._run(solve, a * tan(b * self.x) + c, self.x)
self.result['tg_x'] = res

def ctg_x(self, a, b, c):
res = self._run(solve, a / tan(b * self.x) + c, self.x)
self.result['ctg_x'] = res

def save(self, path_to_result):
with open(path_to_result + 'AnalyzerComputerAlgebra.json', 'w') as file:
file.write(json.dumps(self.result, indent=4))


class LinerEquationAnalyzerGA:
def __init__(self, attempts, a, b, path_to_result):
self.attempts = attempts
self.a = a
self.b = b
self.equation = f'{self.a}*x+{self.b}'
self.result = {}
self.path_to_result = path_to_result

def analyze_generations(self, save=False):
ga = GAInterface(self.equation)
ga_const_config = {
'num_parents_mating': 2,
'sol_per_pop': 20,
'num_genes': 6,
'accuracy': 0.005,
'crossover_type': 'single_point',
'mutation_probability': 0.1,
'parallel_processing': 1
}
self.result['analyze_generation'] = dict(base_config={'attempts': self.attempts, **ga_const_config},
generations=[])

for num_generation in range(10, 1010, 10):
print('Generation:', num_generation)
ga.build_solver(num_generations=num_generation, **ga_const_config)
result = self._run(ga)
needed_data = {
'num_generations': num_generation,
'avg_generation': result.get('avg_generations_completed'),
'avg_error': result.get('avg_error'),
'error': result.get('error'),
'fails': result.get('fails'),
'percent_fails': result.get('percent_fails'),
'avg_time': result.get('avg_time')
}
self.result['analyze_generation']['generations'].append(needed_data)
```

```
if save:
self.save(self.path_to_result + 'analyze_generations.json', self.result['analyze_generation'])

def analyze_population_size(self, save=False):
ga = GAInterface(self.equation)
ga_const_config = {
'num_generations': 1000,
'num_parents_mating': 2,
'num_genes': 4,
'accuracy': 0.01,
'crossover_type': 'uniform',
'mutation_probability': 0.1,
'parallel_processing': 1
}
self.result['analyze_population_size'] = dict(base_config={'attempts': self.attempts, **ga_const_config},
population_size=[])

for population_size in range(4, 200, 1):
print('Pop.size:', population_size)
ga.build_solver(sol_per_pop=population_size, **ga_const_config)
result = self._run(ga)
needed_data = {
'population_size': population_size,
'avg_generation': result.get('avg_generations_completed'),
'avg_error': result.get('avg_error'),
'error': result.get('error'),
'fails': result.get('fails'),
'percent_fails': result.get('percent_fails'),
'avg_time': result.get('avg_time')
}
self.result['analyze_population_size']['population_size'].append(needed_data)

if save:
self.save(self.path_to_result + 'analyze_population_size.json', self.result['analyze_population_size'])

def analyze_num_genes(self, save=False):
ga = GAInterface(self.equation)
ga_const_config = {
'num_generations': 1000,
'num_parents_mating': 2,
'sol_per_pop': 20,
'accuracy': 0.01,
'crossover_type': 'single_point',
'mutation_probability': 0.15,
'parallel_processing': 1
}
self.result['analyze_num_genes'] = dict(base_config={'attempts': self.attempts, **ga_const_config},
num_genes=[])

for num_genes in range(2, 100, 1):
print('Num.genes:', num_genes)
ga.build_solver(num_genes=num_genes, **ga_const_config)
result = self._run(ga)
needed_data = {
'num_genes': num_genes,
'avg_generation': result.get('avg_generations_completed'),
'avg_error': result.get('avg_error'),
'error': result.get('error'),
'fails': result.get('fails'),
'percent_fails': result.get('percent_fails'),
'avg_time': result.get('avg_time')
}
self.result['analyze_num_genes']['num_genes'].append(needed_data)

if save:
self.save(self.path_to_result + 'analyze_num_genes.json', self.result['analyze_num_genes'])

def analyze_crossover_types(self, save=False):
ga = GAInterface(self.equation)
crossover_types = ['single_point', 'two_points', 'uniform', 'scattered']
ga_const_config = {
'num_generations': 5000,
'num_parents_mating': 2,
'num_genes': 5,
'sol_per_pop': 10,
'accuracy': 0.01,
'mutation_probability': 0.15,
'parallel_processing': 1
}
```

```
self.result['analyze_crossover_types'] = dict(base_config={'attempts': self.attempts, **ga_const_config},
crossover_types=[])

for crossover_type in crossover_types:
print('Crossover type:', crossover_type)
ga.build_solver(crossover_type=crossover_type, **ga_const_config)
result = self._run(ga)
needed_data = {
'crossover_type': crossover_type,
'num_genes': result.get('num_genes'),
'avg_generation': result.get('avg_generations_completed'),
'avg_error': result.get('avg_error'),
'error': result.get('error'),
'fails': result.get('fails'),
'percent_fails': result.get('percent_fails'),
'avg_time': result.get('avg_time')
}
self.result['analyze_crossover_types']['crossover_types'].append(needed_data)

if save:
self.save(self.path_to_result + 'analyze_crossover_types.json', self.result['analyze_crossover_types'])

def analyze_mutation_probability(self, save=False):
ga = GAInterface(self.equation)
ga_const_config = {
'num_generations': 1000,
'num_parents_mating': 2,
'num_genes': 4,
'sol_per_pop': 10,
'accuracy': 0.01,
'crossover_type': 'two_points',
'parallel_processing': 1
}
self.result['analyze_mutation_probability'] = dict(base_config={'attempts': self.attempts, **ga_const_config},
mutation_probabilities=[])

for value in range(100):
mutation_probability = value / 100
print('Mutation probability:', mutation_probability)
ga.build_solver(mutation_probability=mutation_probability, **ga_const_config)
result = self._run(ga)
needed_data = {
'mutation_probability': mutation_probability,
'avg_generation': result.get('avg_generations_completed'),
'avg_error': result.get('avg_error'),
'error': result.get('error'),
'fails': result.get('fails'),
'percent_fails': result.get('percent_fails'),
'avg_time': result.get('avg_time')
}
self.result['analyze_mutation_probability']['mutation_probabilities'].append(needed_data)

if save:
self.save(self.path_to_result + 'analyze_mutation_probability.json',
self.result['analyze_mutation_probability'])

def analyze_accuracy(self, save=False):
ga = GAInterface(self.equation)
ga_const_config = {
'num_generations': 1000,
'num_parents_mating': 2,
'num_genes': 6,
'sol_per_pop': 20,
'crossover_type': 'two_points',
'mutation_probability': 0.2,
'parallel_processing': 1,
}
self.result['analyze_accuracy'] = dict(base_config={'attempts': self.attempts, **ga_const_config},
accuracy=[])

for accuracy in np.arange(0.001, 0.5, 0.001):
print('Accuracy:', accuracy)
ga.build_solver(accuracy=accuracy, **ga_const_config)
result = self._run(ga)
needed_data = {
'accuracy': accuracy,
'avg_generation': result.get('avg_generations_completed'),
'avg_error': result.get('avg_error'),
'error': result.get('error'),
```

```
'fails': result.get('fails'),
'percent_fails': result.get('percent_fails'),
'avg_time': result.get('avg_time')
}
self.result['analyze_accuracy']['accuracy'].append(needed_data)

if save:
self.save(self.path_to_result + 'analyze_accuracy.json', self.result['analyze_accuracy'])

def _run(self, ga):
execution_times, generations, ga_result, errors = [], [], {}, []
fails = 0
for _ in range(self.attempts):
execution_time = ga.run_solver()['execution_time']
ga_result = ga.get_result()
if ga_result.get('error') > ga_result.get('accuracy'):
fails += 1
continue
execution_times.append(execution_time)
generations.append(ga_result.pop('generations_completed'))
errors.append(ga_result.get('error'))

avg_time = sum(execution_times) / len(execution_times) if execution_times else 0
min_time = min(execution_times) if execution_times else 0
max_time = max(execution_times) if execution_times else 0
avg_generations_completed = sum(generations) / len(generations) if generations else 0
avg_error = sum(errors) / len(errors) if errors else 0
return {
'avg_time': avg_time,
'min_time': min_time,
'max_time': max_time,
'avg_generations_completed': avg_generations_completed,
'avg_error': avg_error,
'attempts': self.attempts,
'fails': fails,
'percent_fails': 100 * fails / self.attempts,
**ga_result
}

def save(self, path_to_result, data=None):
with open(path_to_result, 'w') as file:
file.write(json.dumps(data if data else self.result, indent=4))


def build_graphs_line(data_x, data_y, func_type='pol'):
"""func type: pol, exp, log, lin"""

def fit_func(x, a, b, c, d, e):
if func_type == 'pol':
return a * x ** 4 + b * x ** 3 + c * x ** 2 + d * x + e
if func_type == 'exp':
return (a - c) * np.exp(-x / b) + c
if func_type == 'log':
return a * np.log(b * x) + c
if func_type == 'lin':
return a * x + b
raise Exception('func type not found, choose one of next ["pol", "exp", "log", "lin"]')

popt, _ = curve_fit(fit_func, data_x, data_y, maxfev=8000)
x_line = arange(min(data_x), max(data_x), 0.01)
y_line = fit_func(x_line, *popt)
return x_line, y_line


def create_plot(path_to_save, data_x, data_y, title, label_x, label_y, fit_func='pol', scale_x=False, scale_y=False,
include_fit_func=True, scatter=False):
x_line, y_line = build_graphs_line(data_x, data_y, fit_func)
fig, ax = pyplot.subplots()
if scale_y:
ax.set_yscale('log')
if scale_x:
ax.set_xscale('log')

if scatter:
ax.scatter(data_x, data_y)
if not scatter:
ax.plot(data_x, data_y)

if include_fit_func:
```

```
 ax.plot(x_line, y_line, '--', color='red')
 pyplot.title(title)
 pyplot.xlabel(label_x)
 pyplot.ylabel(label_y)
 pyplot.draw()
 pyplot.savefig(path_to_save)


def create_bar(path_to_save, data_x, data_y, title, label_x, label_y):
 fig, ax = pyplot.subplots()
 ax.bar(data_x, data_y)
 pyplot.title(title)
 pyplot.xlabel(label_x)
 pyplot.ylabel(label_y)
 pyplot.draw()
 pyplot.savefig(path_to_save)


def create_multibar(path_to_save, min_time, avg_time, max_time, comp_alg_time, title):
 x = np.arange(1) # the label locations
 width = 0.25 # the width of the bars

 fig, ax = pyplot.subplots()
 rects1 = ax.bar(x - 0.5 * width, round(min_time, 2), width, color='g', alpha=0.6, label='min ga')
 rects2 = ax.bar(x + 0.5 * width, round(avg_time, 2), width, color='b', alpha=0.6, label='avg ga')
 rects3 = ax.bar(x + 1.5 * width, round(max_time, 2), width, color='r', alpha=0.6, label='max ga')
 rects4 = ax.bar(x + 4 * width, round(comp_alg_time, 2), width, alpha=0.6, color='m', label='sympy')

 ax.set_ylabel('Time (ms)')
 ax.set_title(title)
 ax.set_xticks([], [])
 ax.legend()

 ax.bar_label(rects1, padding=3)
 ax.bar_label(rects2, padding=3)
 ax.bar_label(rects3, padding=3)
 ax.bar_label(rects4, padding=3)
 fig.tight_layout()
 pyplot.draw()
 pyplot.savefig(path_to_save)


def get_formatted_data(path, list_key, x_data_key):
 data = read_json_from_file(path)
 data_x, data_avg_gen, data_avg_error, data_per_fails, data_avg_time = [], [], [], [], []
 for item in data[list_key]:
 data_x.append(item[x_data_key])
 data_avg_gen.append(item['avg_generation'])
 data_avg_error.append(item['avg_error'])
 data_per_fails.append(item['percent_fails'])
 data_avg_time.append(item['avg_time'] * 1000)
 return data_x, data_avg_gen, data_avg_error, data_per_fails, data_avg_time


def get_multibar_formatted_data(file, data_comp_alg):
 data_ga = read_json_from_file(Config.PATH_TO_STATISTIC_DATA + file)
 avg_time, min_time, max_time = data_ga['avg_time'] * 1000, data_ga['min_time'] * 1000, data_ga['max_time'] * 1000
 accuracy = data_ga['accuracy']
 comp_alg_time = data_comp_alg[file.removesuffix('.json')]['execution_time'] * 1000
 return min_time, avg_time, max_time, comp_alg_time, accuracy


def create_images():
 files = os.listdir(Config.PATH_TO_STATISTIC_DATA)
 data_comp_alg = read_json_from_file(Config.PATH_TO_STATISTIC_DATA + 'AnalyzerComputerAlgebra.json')
 for file in files:
 if file == 'analyze_accuracy.json':
 data_x, data_avg_gen, data_avg_error, data_per_fails, data_avg_time = get_formatted_data(
 Config.PATH_TO_STATISTIC_DATA + file, 'accuracy', 'accuracy')

 create_plot(Config.PATH_TO_STATISTIC_IMG + 'accuracy-avg_generation.png', data_x, data_avg_gen,
 'Average generations vs accuracy', 'accuracy', 'avg.generation', 'exp')
 create_plot(Config.PATH_TO_STATISTIC_IMG + 'accuracy-avg_generation2.png', data_x, data_avg_gen,
 'Average generations vs accuracy',
 'accuracy', 'avg.generation', 'exp', scale_y=True, include_fit_func=False)

 create_plot(Config.PATH_TO_STATISTIC_IMG + 'accuracy-error.png', data_x, data_avg_error,
 'Average error vs accuracy', 'accuracy', 'avg.error', 'exp')
```

```
create_plot(Config.PATH_TO_STATISTIC_IMG + 'accuracy-fails.png', data_x, data_per_fails,
'Percent fails vs accuracy', 'accuracy', 'fails (%)', 'lin')

create_plot(Config.PATH_TO_STATISTIC_IMG + 'accuracy-time.png', data_x, data_avg_time,
'Average time vs accuracy', 'accuracy', 'avg.time (ms)', 'exp')
create_plot(Config.PATH_TO_STATISTIC_IMG + 'accuracy-time2.png', data_x, data_avg_time,
'Average time vs accuracy', 'accuracy', 'avg.time (ms)', scale_y=True, include_fit_func=False)

elif file == 'analyze_crossover_types.json':
data_x, data_avg_gen, data_avg_error, data_per_fails, data_avg_time = get_formatted_data(
Config.PATH_TO_STATISTIC_DATA + file, 'crossover_types', 'crossover_type')

create_bar(Config.PATH_TO_STATISTIC_IMG + 'crossover_type-avg_generation.png', data_x, data_avg_gen,
'Average generations vs crossover type', 'crossover type', 'avg.generation')

create_bar(Config.PATH_TO_STATISTIC_IMG + 'crossover_type-avg_error.png', data_x, data_avg_error,
'Average error vs crossover type', 'crossover type', 'avg.error')

create_bar(Config.PATH_TO_STATISTIC_IMG + 'crossover_type-per_fails.png', data_x, data_per_fails,
'Fails vs crossover type', 'crossover type', 'fails')

create_bar(Config.PATH_TO_STATISTIC_IMG + 'crossover_type-avg_time.png', data_x, data_avg_time,
'Average time vs crossover type', 'crossover type', 'avg.time (ms)')

elif file == 'analyze_generations.json':
data_x, data_avg_gen, data_avg_error, data_per_fails, data_avg_time = get_formatted_data(
Config.PATH_TO_STATISTIC_DATA + file, 'generations', 'num_generations')

create_plot(Config.PATH_TO_STATISTIC_IMG + 'generations-error.png', data_x, data_avg_error,
'Average error vs generations', 'generations', 'avg.error', 'lin')

create_plot(Config.PATH_TO_STATISTIC_IMG + 'generations-fails.png', data_x, data_per_fails,
'Percent fails vs generations', 'generations', 'fails (%)', 'exp')

create_plot(Config.PATH_TO_STATISTIC_IMG + 'generations-time.png', data_x, data_avg_time,
'Average time vs generations', 'generations', 'avg.time (ms)', 'exp')


elif file == 'analyze_mutation_probability.json':
data_x, data_avg_gen, data_avg_error, data_per_fails, data_avg_time = get_formatted_data(
Config.PATH_TO_STATISTIC_DATA + file, 'mutation_probabilities', 'mutation_probability')

create_plot(Config.PATH_TO_STATISTIC_IMG + 'mutation_probabilities-avg_generation.png', data_x,
data_avg_gen,
'Average generations vs mutation probabilities', 'mutation probability', 'avg.generation',
'exp')

create_plot(Config.PATH_TO_STATISTIC_IMG + 'mutation_probabilities-per_fails.png', data_x, data_per_fails,
'Percent fails vs mutation probabilities', 'mutation probability', 'fails (%)', 'exp')

create_plot(Config.PATH_TO_STATISTIC_IMG + 'mutation_probabilities-avg_time.png', data_x, data_avg_time,
'Average time vs mutation probabilities', 'mutation probability', 'avg.time (ms)', 'exp',
scatter=True)

elif file == 'analyze_num_genes.json':
data_x, data_avg_gen, data_avg_error, data_per_fails, data_avg_time = get_formatted_data(
Config.PATH_TO_STATISTIC_DATA + file, 'num_genes', 'num_genes')

create_plot(Config.PATH_TO_STATISTIC_IMG + 'num_genes-avg_generation.png', data_x,
data_avg_gen,
'Average generations vs number genes', 'number genes', 'avg.generation',
'pol')

create_plot(Config.PATH_TO_STATISTIC_IMG + 'num_genes-avg_error.png', data_x,
data_avg_error,
'Average error vs number genes', 'number genes', 'avg.error',
'pol')

create_plot(Config.PATH_TO_STATISTIC_IMG + 'num_genes-per_fails.png', data_x, data_per_fails,
'Percent fails vs number genes', 'number genes', 'fails (%)', 'pol')

create_plot(Config.PATH_TO_STATISTIC_IMG + 'num_genes-avg_time.png', data_x, data_avg_time,
'Average time vs number genes', 'number genes', 'avg.time (ms)', 'pol',
scatter=False)

elif file == 'analyze_population_size.json':
data_x, data_avg_gen, data_avg_error, data_per_fails, data_avg_time = get_formatted_data(
Config.PATH_TO_STATISTIC_DATA + file, 'population_size', 'population_size')
```

```
create_plot(Config.PATH_TO_STATISTIC_IMG + 'population_size-avg_generation.png', data_x,
data_avg_gen,
'Average generations vs population size', 'population size', 'avg.generation',
'exp')

create_plot(Config.PATH_TO_STATISTIC_IMG + 'population_size-avg_error.png', data_x,
data_avg_error,
'Average error vs population size', 'population size', 'avg.error',
'lin')

create_plot(Config.PATH_TO_STATISTIC_IMG + 'population_size-per_fails.png', data_x, data_per_fails,
'Percent fails vs population size', 'population size', 'fails (%)', 'log')

create_plot(Config.PATH_TO_STATISTIC_IMG + 'population_size-avg_time.png', data_x, data_avg_time,
'Average time vs population size', 'population size', 'avg.time (ms)', 'log',
scatter=False)

elif file == 'cos_x.json':
min_time, avg_time, max_time, comp_alg_time, accuracy = get_multibar_formatted_data(file, data_comp_alg)

create_multibar(Config.PATH_TO_STATISTIC_IMG + file.replace('.json', '.png'), min_time, avg_time, max_time,
comp_alg_time, f'Speed solve cos equation (acc={accuracy})')
elif file == 'ctg_x.json':
min_time, avg_time, max_time, comp_alg_time, accuracy = get_multibar_formatted_data(file, data_comp_alg)

create_multibar(Config.PATH_TO_STATISTIC_IMG + file.replace('.json', '.png'), min_time, avg_time, max_time,
comp_alg_time, f'Speed solve ctg equation (acc={accuracy})')
elif file == 'exponential_equation.json':
min_time, avg_time, max_time, comp_alg_time, accuracy = get_multibar_formatted_data(file, data_comp_alg)

create_multibar(Config.PATH_TO_STATISTIC_IMG + file.replace('.json', '.png'), min_time, avg_time, max_time,
comp_alg_time, f'Speed solve exponential equation (acc={accuracy})')
elif file == 'linear_equation.json':
min_time, avg_time, max_time, comp_alg_time, accuracy = get_multibar_formatted_data(file, data_comp_alg)

create_multibar(Config.PATH_TO_STATISTIC_IMG + file.replace('.json', '.png'), min_time, avg_time, max_time,
comp_alg_time, f'Speed solve linear equation (acc={accuracy})')

elif file == 'polynomial_2.json':
min_time, avg_time, max_time, comp_alg_time, accuracy = get_multibar_formatted_data(file, data_comp_alg)

create_multibar(Config.PATH_TO_STATISTIC_IMG + file.replace('.json', '.png'), min_time, avg_time, max_time,
comp_alg_time, f'Speed solve n=2 polynomial equation (acc={accuracy})')
elif file == 'polynomial_3.json':
min_time, avg_time, max_time, comp_alg_time, accuracy = get_multibar_formatted_data(file, data_comp_alg)

create_multibar(Config.PATH_TO_STATISTIC_IMG + file.replace('.json', '.png'), min_time, avg_time, max_time,
comp_alg_time, f'Speed solve n=3 polynomial equation (acc={accuracy})')
elif file == 'polynomial_4.json':
min_time, avg_time, max_time, comp_alg_time, accuracy = get_multibar_formatted_data(file, data_comp_alg)

create_multibar(Config.PATH_TO_STATISTIC_IMG + file.replace('.json', '.png'), min_time, avg_time, max_time,
comp_alg_time, f'Speed solve n=4 polynomial equation (acc={accuracy})')
elif file == 'polynomial_5.json':
min_time, avg_time, max_time, comp_alg_time, accuracy = get_multibar_formatted_data(file, data_comp_alg)

create_multibar(Config.PATH_TO_STATISTIC_IMG + file.replace('.json', '.png'), min_time, avg_time, max_time,
comp_alg_time, f'Speed solve n=5 polynomial equation (acc={accuracy})')
elif file == 'sin_x.json':
min_time, avg_time, max_time, comp_alg_time, accuracy = get_multibar_formatted_data(file, data_comp_alg)

create_multibar(Config.PATH_TO_STATISTIC_IMG + file.replace('.json', '.png'), min_time, avg_time, max_time,
comp_alg_time, f'Speed solve sin equation (acc={accuracy})')
elif file == 'sqrt_x.json':
min_time, avg_time, max_time, comp_alg_time, accuracy = get_multibar_formatted_data(file, data_comp_alg)

create_multibar(Config.PATH_TO_STATISTIC_IMG + file.replace('.json', '.png'), min_time, avg_time, max_time,
comp_alg_time, f'Speed solve square root equation (acc={accuracy})')
elif file == 'tg_x.json':
min_time, avg_time, max_time, comp_alg_time, accuracy = get_multibar_formatted_data(file, data_comp_alg)

create_multibar(Config.PATH_TO_STATISTIC_IMG + file.replace('.json', '.png'), min_time, avg_time, max_time,
comp_alg_time, f'Speed solve tg equation (acc={accuracy})')


def run_statistic():
 analyser = AnalyserGA(Config.ATTEMPTS, Config.PATH_TO_STATISTIC_DATA)
 analyser.linear_equation(5, -3, save=True)
 analyser.sqrt_x(4, 2, -50, save=True)
```

```
 analyser.polynomial_2(2, 5, -15, save=True)
 analyser.polynomial_3(10, -1, -10, 4, save=True)
 analyser.polynomial_4(-4, -7, 5, 4, -2, save=True)
 analyser.polynomial_5(-3, -4, -7, 5, 5, 1, save=True)
 analyser.exponential_equation(5, 1, -10, save=True)
 analyser.sin_x(4, -2, 2, save=True)
 analyser.cos_x(7, 2, 3, save=True)
 analyser.tg_x(-5, 3, -2, save=True)
 analyser.ctg_x(4, 8, -10, save=True)

 anal_comp_alg = AnalyzerComputerAlgebra()
 anal_comp_alg.linear_equation(5, -3)
 anal_comp_alg.sqrt_x(4, 2, -50)
 anal_comp_alg.polynomial_2(2, 5, -15)
 anal_comp_alg.polynomial_3(10, -1, -10, 4)
 anal_comp_alg.polynomial_4(-4, -7, 5, 4, -2)
 anal_comp_alg.polynomial_5(-3, -4, -7, 5, 5, 1)
 anal_comp_alg.exponential_equation(5, 1, -10)
 anal_comp_alg.sin_x(4, -2, 2)
 anal_comp_alg.cos_x(7, 2, 3)
 anal_comp_alg.tg_x(-5, 3, -2)
 anal_comp_alg.ctg_x(4, 8, -10)
 anal_comp_alg.save(Config.PATH_TO_STATISTIC_DATA)

 liner_analyzer = LinerEquationAnalyzerGA(Config.ATTEMPTS, 5, -3, Config.PATH_TO_STATISTIC_DATA)
 Process(target=liner_analyzer.analyze_generations, args=(True,)).start()
 Process(target=liner_analyzer.analyze_population_size, args=(True, )).start()
 Process(target=liner_analyzer.analyze_num_genes, args=(True, )).start()
 Process(target=liner_analyzer.analyze_crossover_types, args=(True, )).start()
 Process(target=liner_analyzer.analyze_mutation_probability, args=(True, )).start()
 Process(target=liner_analyzer.analyze_accuracy, args=(True, )).start()


if __name__ == '__main__':
 run_statistic()
 create_images()
```

tools.py

```
import json
import time


def benchmark(func):
 def wrapper(*args, **kwargs):
 start = time.time()
 res = func(*args, **kwargs)
 finish = time.time()
 execution_time = finish - start
 # print('Execution time: ', execution_time)
 return {'execution_time': execution_time, 'func_res': res}

 return wrapper


def read_json_from_file(path):
 with open(path, 'r') as file:
 return json.loads(file.read())
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css" rel="stylesheet"
 integrity="sha384-Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9Bv1WTRi" crossorigin="anonymous">
</head>
<body>
<p class="text-center fs-3">Solve math equations by genetic algorithm</p>
<div class="container border rounded">
 <form method="post" action="/process_data">
 <div class="mb-3">
 <label for="equation" class="form-label">Equation</label>
 <input type="text" required name="equation" class="form-control" id="equation">
 </div>
 <div class="mb-3">
 <label for="numGeneration" class="form-label">Number generations</label>
```

```
 <input type="number" value="{{ data.num_generations }}" required name="num_generations" class="form-control" id="numGeneration">
 </div>
 <div class="mb-3">
 <label for="populationSize" class="form-label">Population size</label>
 <input type="number" value="{{ data.sol_per_pop }}" required name="sol_per_pop" class="form-control" id="populationSize">
 </div>
 <div class="mb-3">
 <label for="numGenes" class="form-label">Number genes</label>
 <input type="number" value="{{ data.num_genes }}" required name="num_genes" class="form-control" id="numGenes">
 </div>
 <div class="mb-3">
 <label for="accuracy" class="form-label">Accuracy</label>
 <input type="number" value="{{ data.accuracy }}" required step="any" name="accuracy" class="form-control" id="accuracy">
 </div>
 <div class="mb-3">
 <label for="mutationProbability" class="form-label">Mutation probability</label>
 <input type="number" value="{{ data.mutation_probability }}" required step="any" name="mutation_probability" class="form-control"
id="mutationProbability">
 </div>
 <div class="mb-3">
 <label for="numParents" class="form-label">Number parents</label>
 <input type="number" value="{{ data.num_parents_mating }}" required name="num_parents_mating" class="form-control" id="numParents">
 </div>
 <div class="mb-3">
 <label for="crossoverType" class="form-label">Crossover type {{ data.parallel_processing.0 }}</label>
 <select required name="crossover_type" id="crossoverType" class="form-select" aria-label="Default select example">
 <option value="single_point" {% if data.crossover_type == 'single_point' %} selected {% endif %}>single point</option>
 <option value="two_points" {% if data.crossover_type == 'two_points' %} selected {% endif %}>two points</option>
 <option value="uniform" {% if data.crossover_type == 'uniform' %} selected {% endif %}>uniform</option>
 <option value="scattered" {% if data.crossover_type == 'scattered' %} selected {% endif %}>scattered</option>
 </select>
 </div>
 <div class="mb-3 form-check">
 <input {% if data.parallel_processing > 1 %} checked {% endif %} type="checkbox" name="parallel_processing" class="form-check-input"
id="parallelProcessing">
 <label class="form-check-label" for="parallelProcessing">Parallel processing</label>
 </div>
 <div class="d-grid gap-2 mb-3">
 <button class="btn btn-primary" type="submit">Solve</button>
 </div>
 </form>
</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/js/bootstrap.bundle.min.js"
 integrity="sha384-OERcA2EqjJCMA+/3y+gxIOqMEjwtxJY7qPCqsdltbNJuaOe923+mo//f6V8Qbsw3"
 crossorigin="anonymous"></script>
</body>
</html>
```

result.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css" rel="stylesheet"
 integrity="sha384-Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9Bv1WTRi" crossorigin="anonymous">
</head>
<body>
<p class="text-center fs-3">Result</p>
<div class="container border rounded">
 <div class="mb-3">
 <label for="equation" class="form-label">Equation</label>
 <input type="text" readonly value="{{ data.equation }}" name="equation" class="form-control" id="equation">
 </div>
 <div class="mb-3">
 <label for="x" class="form-label">x</label>
 <input type="number" readonly value="{{ data.x }}" name="x" class="form-control" id="x">
 </div>
 <div class="mb-3">
 <label for="error" class="form-label">Error</label>
 <input type="number" readonly value="{{ data.error }}" name="error" class="form-control" id="error">
 </div>
 <div class="mb-3">
 <label for="time" class="form-label">Time (s)</label>
 <input type="number" readonly value="{{ data.execution_time.execution_time }}" name="time" class="form-control" id="time">
 </div>
 <div class="mb-3">
```

```
<label for="fitness" class="form-label">Fitness</label>
<input type="number" readonly value="{{ data.fitness }}" name="fitness" class="form-control" id="fitness">
</div>
<div class="mb-3">
<label for="numGeneration" class="form-label">Number generations</label>
<input type="number" readonly value="{{ data.num_generations }}" name="numGeneration" class="form-control" id="numGeneration">
</div>
<div class="mb-3">
<label for="populationSize" class="form-label">Population size</label>
<input type="number" readonly value="{{ data.sol_per_pop }}" name="populationSize" class="form-control" id="populationSize">
</div>
<div class="mb-3">
<label for="numGenes" class="form-label">Number genes</label>
<input type="number" readonly value="{{ data.num_genes }}" name="numGenes" class="form-control" id="numGenes">
</div>
<div class="mb-3">
<label for="accuracy" class="form-label">Accuracy</label>
<input type="text" readonly value="{{ data.accuracy }}" name="accuracy" class="form-control" id="accuracy">
</div>
<div class="mb-3">
<label for="mutationProbability" class="form-label">Mutation probability</label>
<input type="text" readonly value="{{ data.mutation_probability }}" name="mutationProbability" class="form-control" id="mutationProbability">
</div>
<div class="mb-3">
<label for="numParents" class="form-label">Number parents</label>
<input type="number" readonly value="{{ data.num_parents_mating }}" name="numParents" class="form-control" id="numParents">
</div>
<div class="mb-3">
<label for="crossoverType" class="form-label">Crossover type</label>
<input type="text" readonly value="{{ data.crossover_type }}" name="crossoverType" class="form-control" id="crossoverType">
</div>
<div class="mb-3">
<label class="form-check-label">Parallel processing
{% if data.parallel_processing > 1 %} [On], threads {{ data.parallel_processing }} {% else %} [Off] {% endif %}
</label>
</div>
<div class="mb-3 text-center">
<p>Progress</p>
<img src="../static/my_plot.png" class="img-fluid">
</div>
<div class="d-grid gap-2 mb-3">
<button class="btn btn-primary" onclick="window.location.pathname = '/'" type="button">Back</button>
</div>
</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/js/bootstrap.bundle.min.js"
integrity="sha384-OERcA2EqjJCMA+/3y+gxIOqMEjwtxJY7qPCqsdltbNJuaOe923+mo//f6V8Qbsw3"
crossorigin="anonymous"></script>
</body>
</html>
```