Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки

«Затверджую»
Зав. кафедри теоретичної та
прикладної системотехніки
_____ д.т.н., проф. С. І. Шматков
«___» _____ 2023 р.

Пояснювальна записка

до кваліфікаційної роботи
магістра

на тему: «**НЕЙРОМЕРЕЖЕВІ МОДЕЛІ ОБРОБКИ ТЕКСТУ**»

Захищено на засіданні
Атестаційної комісії № 42
протокол № __ від __.12.2023 р.
Оцінка _____ / _____
Голова Атестаційної комісії
_____ **СКОБ Ю.О.**
(підпис)

Виконав:
студент 2 курсу, групи КУ – 61
Галузь знань: 15 – Автоматизація та приладобудування
Спеціальність: 151 – «Автоматизація та комп'ютерно-інтегровані технології»
ЗАХАРОВ Микита Олександрович
_____

Керівник:
к.т.н., доцент кафедри теоретичної та прикладної системотехніки
СТРІЛЕЦЬ Вікторія Євгенівна

Рецензент:
к.т.н., доцент, в.о. зав. кафедри теоретичної та прикладної інформатики
МЕНЯЙЛОВ Євген Сергійович

Харків – 2023

АНОТАЦІЯ

Пояснювальна записка до магістерської кваліфікаційної роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел і трьох додатків. Загальний обсяг роботи складає 127 сторінки, із яких 70 сторінок основної частини з 65 рисунками, 4 таблицями, 21 найменуваннями списку використаних джерел. Метою роботи є удосконалення оцінки ефективності використання нейромережевих моделей обробки текстових даних у задачах різного типу.

Для досягнення мети були розглянуті кілька чат-ботів на різних мовних нейромережевих моделях, а саме ChatGPT, BingAI, Claude 1 та Perplexity. Також було вибрано три алгоритми для дослідження можливостей чат-ботів: метод Гауса для розв'язання лінійних рівнянь із трьома невідомими, метод золотого перетину та фрактал Коха. Вибір саме цих алгоритмів для дослідження можливостей чат-ботів, дозволяє оцінити їхні функціональні можливості та застосовувати їх у різних сценаріях. Результати, отримані за допомогою різних критеріїв оцінки, допомагають визначити оптимальну нейромережеву модель для розв'язання конкретних завдань. Для виявлення ефективнішого з кодів були використані критерії, такі як timeit, time, cProfile, LineProfile та memory_profile. Для тестування коду використовувалась мова програмування Python.

У результаті роботи був запропонований підхід до оцінки ефективності роботи нейромережевих моделей обробки текстової інформації. Результати роботи можуть бути використані для подальшого вдосконалення чат-ботів і їхнього використання в різних задачах обробки текстових даних. Оцінка ефективності різних мовних моделей, таких як GPT-3.5 TURBO, GPT-4, Claude 1.2, важлива для розробки більш продуктивних і інтелектуальних чат-ботів, які можуть виконувати завдання відповідно до потреб користувачів.

Ключові слова: штучний інтелект, нейромережі, обробка природньої мови, оцінка ефективності.

ABSTRACT

The explanatory note to the master's qualification work consists of an introduction, three chapters, conclusions, a list of references and three appendices. The total volume of the work is 127 pages, including 70 pages of the main part with 65 figures, 4 tables, 21 references. The aim of the work is to evaluate the effectiveness of using neural network models for text data processing in various types of tasks.

To achieve this goal, several chatbots based on different language neural network models were considered, namely ChatGPT, BingAI, Claude 1, and Perplexity. Three algorithms were also chosen to study the capabilities of chatbots: the Gaussian method for solving linear equations with three unknowns, the golden ratio method, and the Koch fractal. The choice of these algorithms to study chatbot capabilities allows us to evaluate their functionality and apply them in different scenarios. The results obtained using different evaluation criteria help to determine the optimal neural network model for solving specific tasks. To identify the most efficient code, we used criteria such as timeit, time, cProfile, LineProfile, and memory_profile. The Python programming language was used to test the code.

As a result of the study, an approach to evaluating the efficiency of neural network models for processing textual information was proposed. The results can be used for further improvement of chatbots and their use in various text processing tasks. Evaluating the effectiveness of various language models, such as GPT-3.5 TURBO, GPT-4, Claude 1.2, is important for developing more productive and intelligent chatbots that can perform tasks according to user needs.

Keywords: artificial intelligence, neural networks, natural language processing, performance evaluation.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

ШНМ – штучна нейрона мережа

GPT – Generative Pre-trained Transformer

LaMDA – Language Model for Dialogue Applications

ReLU – Rectified Linear Unit

NLP – Natural Language Processing

RNNLM – Recurrent Neural Network Language Modeling

GloVe – Global Vectors

NLG – Natural Language Generation

NLU – Natural Language Understanding

BERT – Bidirectional Encoder Representations from Transformers

LSTM – Long short-term memory

GRUs – Gated recurrent units

CBOW – Continuous Bag of Words

NS – Negative Sampling

RLHF – Reinforcement Learning with Human Feedback

LLM – Large Language Models

ВСТУП

Сучасний світ відзначається широким впровадженням інформаційних і комп'ютерно-інтегрованих технологій в усі сфери життя, зокрема зростають потреби в створенні й вдосконаленні технологій обробки природньої мови, як усної, так і письмової.

Актуальність роботи. З розвитком потужності графічних процесорів та збільшенням завдань, які можуть бути вирішені із застосуванням сучасних технологій штучного інтелекту, розробка і впровадження методів і моделей обробки природньої мови і текстової інформації стає надзвичайно важливою. Протягом лише останнього часу вони набули великої популярності. Тому проведення дослідження сучасних чат-ботів, які здатні працювати в діалоговому режимі та реагувати на запити природними мовами, стає актуальним завданням.

Мета дослідження - удосконалення оцінки ефективності нейромережевих моделей GPT-3.5 TURBO, GPT-4, Claude 1.2 для обробки текстових даних у задачах різного типу.

Об'єкт дослідження є процес обробки текстових даних з використанням нейромережевих моделей GPT-3.5 TURBO, GPT-4, Claude 1.2.

Предмет дослідження: нейромережеві моделі обробки тексту, які здатні здійснювати автоматичний аналіз вхідних даних, виконувати різні операції та генерувати результати відповідно до заданих параметрів.

Завдання дослідження:

1. Аналіз нейромережих моделей для обробки текстової інформації.

2. Розробка методики тестування нейромережих моделей для обробки текстових даних.

3. Формулювання запитів для тестування нейромережевих моделей.

4. Дослідження ефективності роботи нейромережих моделей для обробки текстових даних.

Методи дослідження. Для проведення даного дослідження було використано мову програмування Python, яка відома своєю ефективністю у

тестуванні нейронних мереж. У процесі дослідження були задіяні наступні чат-боти: ChatGPT, який базується на мовній моделі GPT-3.5 TURBO; BingAI, який використовує мовну модель GPT-4 та має доступ до мережі Інтернет; Claude 1, побудований на мовній моделі Claude 1.2, і Perplexity, який базується на мовній моделі GPT-3.5 TURBO і має доступ до відкритої мережі Інтернет.

РОЗДІЛ 1
МОДЕЛІ НЕЙРОННИХ МЕРЕЖ
ДЛЯ ЗАДАЧ ОБРОБКИ ПРИРОДНЬОЇ МОВИ

Нейромережа – це абстрактна математична модель, яка моделює структуру та функціонування біологічних нейронних мереж. Її основна мета – вирішення різноманітних завдань, таких як класифікація, регресія, прогнозування та генерація, шляхом спільної роботи штучних нейронів, які організовані у графові структури і обмінюються інформацією через ваги зв'язків. Під час навчання, коли ваги та зміщення між нейронами налаштовуються, нейромережа здатна виявляти закономірності вхідних даних. Архітектура нейромережі складається з трьох основних типів шарів:

– вхідний шар який служить точкою входу для вхідних даних, таких як зображення або текст. Він не змінює дані, а просто передає їх до наступних шарів для подальшої обробки;

– приховані шари які здійснюють обробку вхідних даних та обмін інформацією між шарами. Їхні результати не впливають напряму на виході мережі, і їх кількість і структура можуть змінюватися в залежності від завдання та архітектури;

– вихідний шар який генерує результат, який передбачається нейромережею на основі вхідних даних. Результат може бути у формі класифікації, числового значення або іншої інформації, залежно від завдання.

Шари в нейромережі – це групи нейронів, які спільно виконують обробку інформації на різних етапах. Вони дозволяють мережі адаптуватися до різних завдань.

Ваги зв'язків між нейронами відіграють важливу роль у навчанні нейромережі, визначаючи силу впливу одного нейрона на інший. В процесі навчання ваги оптимізуються для мінімізації помилок передбачення мережі. Кожен нейрон також має зміщення, яке регулює його активацію незалежно від вхідного сигналу, допомагаючи мережі адаптуватися до різних даних. Функція

активації визначає активність кожного нейрона на основі вхідних сигналів, ваг та зміщення. Вона може бути лінійною або не лінійною, залежно від завдання та архітектури мережі. Деякі популярні функції активації включають сигмоїду, гіперболічний тангенс, ReLU та Softmax.

1.1 Біологічний нейрон

Розробка штучних нейронних мереж відбувається під впливом біології. Це означає, що при вивченні структури та алгоритмів мереж вчені користуються термінами та поняттями, які пов'язані з організацією роботи мозку. Однак ця аналогія, мабуть, має свої межі. Наше розуміння роботи мозку настільки обмежене, що нам мало відомо для того, щоб точно наслідувати його функції. Тому розробники нейронних мереж повинні виходити за рамки сучасних біологічних знань у пошуках структур, які здатні виконувати корисні функції. Основними компонентами мозку людини є нейрони – спеціальні клітини, які здатні запам'ятовувати та застосовувати попередній досвід до майбутніх дій. Ця особливість нейронів робить їх унікальними серед інших клітин організму. Нейрони вміють передавати імпульси через синаптичні зв'язки. Складаються вони з трьох головних компонентів: соми (тіла клітини), дендритів та аксона. Сома містить ядро, яке зберігає інформацію про властивості нейрона, і плазму, яка виробляє необхідні матеріали. Нейрон отримує сигнали (імпульси) через дендрити, і передає сигнали через аксон, який поділяється на волокна. Синапси розташовані на кінцях волокон і є функціональними вузлами, що сполучають два нейрони (волокно аксона одного нейрона і дендрит іншого).

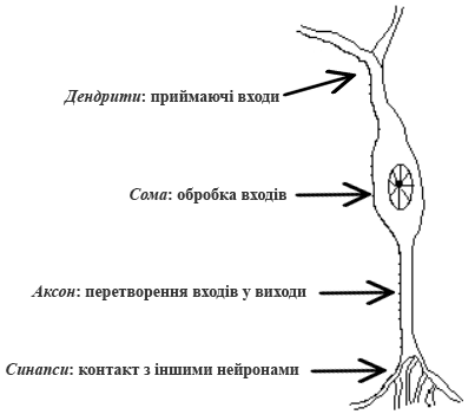


Рис. 1.1 – Схема біологічного нейрона

Коли імпульс досягає синаптичного закінчення, там виробляються хімічні речовини, відомі як нейротрансмітери, які переходять через синаптичну щілину і, в залежності від типу синапсу, можуть стимулювати або гальмувати здатність нейрона-приймача генерувати електричні імпульси. Ефективність синапсу налаштовується сигналами, які проходять через нього, і тому синапси навчаються у відповідності до активності процесів, у яких вони беруть участь. Нейрони спілкуються за допомогою коротких серій імпульсів, і повідомлення передається за допомогою частотно-імпульсної модуляції.

Однією з основних відмінностей між біологічними та штучними нейронами є швидкість передачі сигналів та навчання. Штучні нейрони передають інформацію зі швидкістю, що відповідає сучасним комп'ютерним процесорам, у той час як біологічні можуть передавати імпульси зі значною швидкістю, до 120 метрів за секунду. Крім того, штучні нейромережі можуть навчатися набагато швидше, завдяки використанню паралельних алгоритмів та оптимізації обчислень. З іншого боку, біологічні нейронні мережі мають значно більше кількість нейронів та зв'язків, що дає їм перевагу у аналізі складних ситуацій та розвитку адаптивних стратегій. У штучних нейромережах, незважаючи на їхній прогрес, поки що неможливо повністю відтворити всі функції людського мозку.

Біологічні нейрони є складнішими структурно, ніж штучні нейрони, і завдяки постійному розвитку технологій обчислень, розробники нейромереж мають нескінченні можливості для удосконалення моделей, що імітують біологічний мозок.

1.2 Штучні нейрони

Основним базовими модулем нейронних мереж є штучні нейрони. Цей компонент складається з набору вхідних сигналів з вагами, а також функції перетворення та функції активації. Функція активації, розташована на кінці, виконує роль аксона в біологічному нейроні. Зважені вхідні сигнали відповідають вхідним сигналам, які вводяться в біологічний нейрон, який

отримує електричні імпульси, пересуваючись через мозок, і обробляє їх, щоб передати їх наступним шарам нейронів.

Рис. 1.2 – Структура штучного нейрона

Вхідними сигналами штучного нейрона $x_i(i = \overline{1,N})$ є вихідні сигнали інших нейронів, кожний з яких узятий зі своєю вагою $w_i(i = \overline{1,N})$, аналогічною до синаптичної сили. Вхідний оператор $f_{\text{вх}}$ перетворює зважені входи й подає їх на оператор активації $f_{\text{а}}$. Вихідний сигнал нейрона у являє собою перетворений вихідним оператором $f_{\text{вих}}$ вихідний сигнал оператора активації. Таким чином, нелінійний оператор перетворення вектора вхідних сигналів *x* у вихідний сигнал у може бути записаний у такий спосіб:

$$y = f_{\text{вих}}\left(f_{\text{а}}\left(f_{\text{вих}}(x,w)\right)\right). \quad (1)$$

1.3 Архітектура ШНМ

Пов'язані між собою нейрони створюють штучну нейромережу. Таким чином, вона може бути представлена як пара *(M, V)*, де *M* – це множина нейронів, а *V* – множина зв'язків. Архітектура мережі визначається у формі графа, де вершини відповідають нейронам, а ребра – зв'язкам між ними. Важливою характеристикою нейромереж є велика кількість зв'язків, що з'єднує окремі нейрони. Групування нейронів у людському мозку дозволяє обробляти інформацію динамічно, інтерактивно і самоорганізовано. Біологічні нейромережі складаються з мікроскопічних компонентів, існують у тривимірному просторі і мають різноманітні з'єднання. Однак при створенні штучних нейромереж існують фізичні обмеження. Коли штучні нейрони

об'єднуються в мережу, вони створюють систему обробки інформації, яка дозволяє ефективно адаптувати модель до постійних змін у зовнішньому середовищі. Під час функціонування мережі вхідний вектор сигналів перетворюється у вихідний, і конкретний спосіб цього перетворення залежить від архітектури нейромережі, характеристик нейронних елементів, методів управління та синхронізації інформаційних потоків між нейронами. Важливим аспектом ефективності мережі є визначення оптимальної кількості нейронів та видів зв'язків між ними.

Загалом, штучні нейронні мережі складаються з кількох шарів, включаючи вхідний, який приймає зовнішні сигнали, вихідний, який відображає реакцію нейронів на комбінації вхідних сигналів, і у випадку багатошарових нейромереж – приховані шари. Така організація на рівні шарів аналогічна до локальної структури деяких відділів мозку.

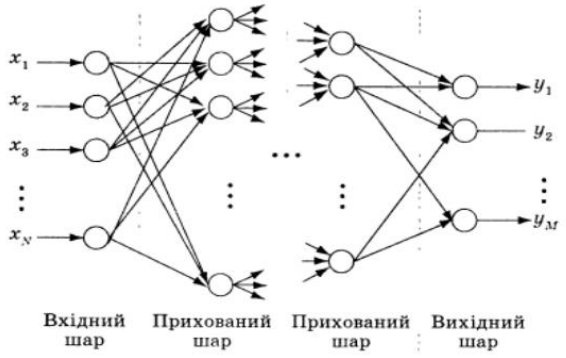


Рис. 1.3 – Структура ШНМ

Структура нейромережі – це спосіб зв'язків нейронів у нейромережі (рис. 1.7). Структура нейромережі та тип нейронів, в свою чергу, створюють архітектуру нейромережі. Найпоширенішою архітектурою є багатошарові мережі, де нейрони групуються в шари зі спільним вхідним сигналом (рис. 1.10). Зовнішні дані подаються на вхідний шар нейронної мережі (це можуть бути рецептори), а вихідні сигнали отримуються від останнього шару (це можуть бути ефектори). Крім вхідного та вихідного шарів, нейромережа також може мати один або декілька прихованих шарів нейронів, які не мають зв'язків з зовнішнім середовищем. Зв'язки між нейронами різних прошарків називають

проективними. Зв'язки між нейронами одного прошарку називають бічними (латеральними).

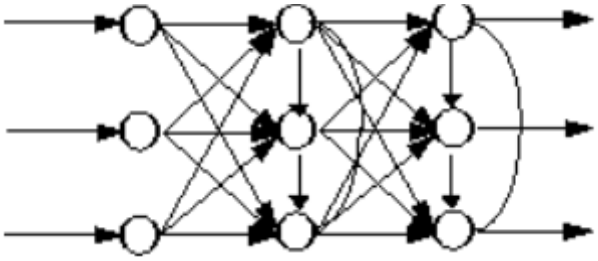


Рис. 1.4 – Схема багатошарової нейромережі

На рис. 1.10 показано типову структуру штучних нейромереж. Хоча існують мережі, що складаються лише з одного шару або навіть лише з одного елемента, більшість застосувань передбачають мережу, що має принаймні три типи шарів: вхідний, прихований та вихідний. Вхідний шар нейронів отримує дані або з вхідних файлів, або безпосередньо від датчиків. Вихідний шар передає інформацію безпосередньо до зовнішнього середовища, до подальшої обробки комп'ютером або до інших пристроїв. Між цими двома шарами може бути декілька прихованих шарів, які містять багато нейронів, організованих у різні зв'язані структури. Кожен прихований нейрон з'єднаний з іншими нейронами за допомогою входів і виходів. Важливою рисою нейромереж є напрямок зв'язку від одного нейрону до іншого: зв'язки, які йдуть від вхідних до вихідних шарів, називаються аферентними, а ті, які йдуть в зворотному напрямку, — еферентними. Зазвичай кожен нейрон прихованого шару отримує сигнали від всіх нейронів попереднього шару або від нейронів вхідного шару. Після обробки сигналів нейрон передає свій результат всім нейронам наступних шарів, забезпечуючи передачу вперед (feedforward) на вихід.

1.4 NLP-моделі

NLP або обробка природної мови, представляє собою окремий галузь штучного інтелекту, яка зосереджена на тому, як комп'ютери можуть розуміти, аналізувати і відтворювати природну мову, якою люди спілкуються. У науковому контексті існує тест Тьюрінга, який стверджує, що якщо людина,

взаємодіючі з комп'ютером, не може визначити, що вона спілкується з машиною, то ця машина вважається штучним інтелектом. Це визначило важливий пункт у розвитку NLP. Застосування NLP включає в себе щоденні взаємодії з мобільними пристроями, такими як Siri або Alexa, використання розумних особистих асистентів, таких як Cortana в операційній системі Windows, в автоматичному визначенні спам-пошти в Gmail, в онлайн-перекладачах і навіть у чат-ботах, таких як ChatGPT, BingAI, Claude 1 та Perplexity.

Люди використовують слова для спілкування, тоді як машини оперують числами. Кожен запит, який людина робить до машини голосом або текстом, потребує перекладу цього запиту на мову чисел, зрозуміння його сутності та наміру людини, інтерпретації цієї інформації, а потім вживання певних дій або надання відповіді. Цей процес подібний до того, як люди спілкуються між собою, за винятком того, що машинам потрібно більше пояснень, якщо мова йде про їхню логічну мову. Технологія обробки природної мови (NLP) призначена саме для цього. Як і в багатьох інших галузях штучного інтелекту, ми маємо справу з машинним навчанням, що передбачає навчання машини. Цей процес схожий на той, як люди вивчають мову, але машина робить це набагато швидше та ефективніше, працюючи з великим обсягом даних. Фахівці з обробки природної мови (NLP-інженери) готують дані до навчання моделі, роблять їх розмітку та структурують. Після очищення та обробки великої кількості даних ці дані подаються на вхід NLP-моделі, де відбувається процес навчання. Після завершення цього процесу модель вважається навченою і готовою для інтеграції в потрібні програмні процеси. Наявність такої навченої моделі у системі чи програмному продукті дозволяє їм розуміти природну мову людини. Однак з часом може виникнути необхідність в оновленні та навчанні моделі для розуміння нових актуальних даних. У цьому випадку модель перенавчають або поновлюють її навчання.

На сучасному ринку існує широкий вибір попередньо навчених NLP-моделей, створених на основі обробки великих обсягів даних. Це дозволяє легко інтегрувати їх та налаштовувати під конкретні теми та цілі використання. Цей

підхід значно економить час і зусилля. До таких моделей відносяться, наприклад, модель трансформера, яку використовують у ChatGPT та інших, рекурентні нейромережеві мовні моделі (RNNLM), Word2vec, GloVe, fastText і багато інших.

1.5 Мовна модель трансформер

Трансформер (Transformer) – це сучасна модель нейронної мережі, розробленої для обробки послідовностей даних, включаючи текстові дані. Саме на основі цієї моделі була навчена GPT-3.5 та Claude 1.2, яку використовують в ChatGPT та Claude 1. В основі Трансформера лежить підхід sequence-to-sequence, але відмінною рисою є використання стеків з 6 енкодерів та 6 декодерів замість одного енкодера та декодера. Кожен енкодер і декодер мають однакову структуру, але власні параметри матриці (на відміну від рекурентних нейромереж, які використовують спільні параметри). Механізм self-attention, що використовується в декодері, дозволяє моделі фокусуватися на важливих частинах вхідної послідовності за потребою. Він отримує не лише останній прихований вектор від енкодера, але й всі проміжні вектори, що дозволяє використовувати більше інформації з вхідної послідовності.

Класична архітектура Transformer була спочатку розроблена для вирішення завдань в області обробки природної мови NLG, де вхідні та вихідні дані представлені як послідовності. Однак цей підхід, в основному, використовується в завданнях генерації природної мови NLU. В архітектурі Transformer енкодерні блоки обробляють вхідну послідовність та створюють вектори, які передаються в декодерну частину моделі для генерації наступного слова на основі цих векторів та інформації про попередні елементи послідовності. Проте, задача векторизації тексту, яка є частиною обробки природної мови (NLU), полягає в створенні фіксовано розмірних векторів для вхідної послідовності, які максимально інкапсулюють її зміст. В таких задачах декодер не є обов'язковим елементом, і головною частиною є енкодер. З огляду на ці вимоги, для базової архітектури була обрана модель BERT, (рис. 1.12), яка має лише енкодерні шари Transformer та декодер замінений кількома лінійними шарами та softmax-функцією. Модель BERT була розроблена Google в 2019 році

і тренується на двох етапах: попереднє тренування на великій кількості нерозмічених даних та дотренування (fine-tuning) на конкретних розмічених даних для вирішення певної задачі.

Ідея полягає у можливості розробки моделей для конкретних задач у сфері обробки природної мови без потреби збирати велику кількість розмічених даних та починати тренування з випадково ініціалізованими вагами моделі. Збір і підготовка якісних розмічених текстових даних для задач NLP може бути дорогим і тривалим процесом. Замість цього, модель може бути попередньо натренована на нерозмічених даних, таких як тексти зі всієї Вікіпедії або корпуси електронних книг. Один із прикладів такого претренування може включати передбачення наступного слова в послідовності, використовуючи контекстні слова. Після претренування модель не розв'язує конкретну задачу, але її ваги вже налаштовані ближче до необхідного розподілу, що значно скорочує час дотренування моделі на даних конкретної задачі та кількість необхідних розмічених даних для навчання. Окрім цього, обрана архітектура BERT має дві варіації:

– BERT-base: Зазвичай складається з 12 блоків енкодера, розмірність прихованого вектора – 768 та загальна кількість параметрів – 110 мільйонів;

– BERT-large: Зазвичай складається з 24 блоків енкодера, розмірність прихованого вектора – 1024 та загальна кількість параметрів – 340 мільйонів.

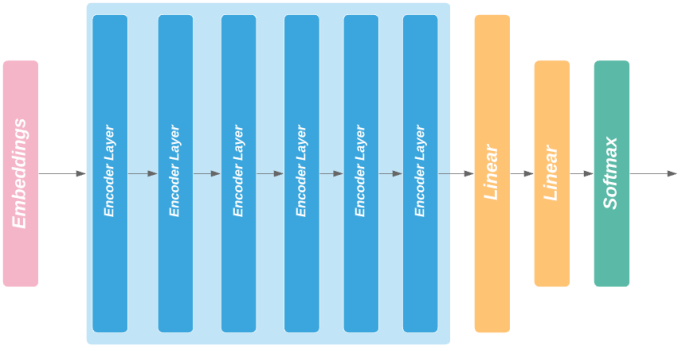


Рис. 1.5 – Структурна схема архітектури BERT

Модель отримує текст на вході у вигляді токенів, де кожному слову відповідає свій унікальний індекс. Трансформація цих токенів у векторне представлення відбувається всередині самої моделі, що означає, що навчання ваг ембедінг-шару відбувається паралельно з навчанням всієї моделі. Окрім токенів слів, BERT отримує на вході вектори, які відображають позиції слів у послідовності, а також ембедінги сегментів, які вказують на приналежність слів до певних сегментів або речень. Наприклад, речення можуть бути розділені на окремі сегменти для забезпечення логічного розрізнення. Усі три види ембедінгів об'єднуються і передаються на блоки енкодера моделі. BERT пройшов попереднє навчання на двох основних завданнях: передбачення наступного слова в послідовності і передбачення наступного речення в тексті. Під час попереднього навчання розробники використали два текстові корпуси – BooksCorpus (з 800 мільйонів слів) і корпус англійської Вікіпедії (з 2500 мільйонів слів).

Висновок за розділом 1

Після дослідження моделей нейронних мереж, використаних для задач обробки природньої мови, можна сказати, що нейромережа представляє собою абстрактну математичну модель, яка наслідує структуру та функціонування біологічних нейронних мереж. Основною її метою є вирішення різноманітних завдань, таких як класифікація, регресія, прогнозування та генерація, через взаємодію штучних нейронів, організованих у графові структури і обмінюються інформацією через ваги зв'язків.

Архітектура нейромережі включає в себе три основних типи шарів: вхідний, прихований та вихідний. Шари в нейромережі складаються з груп штучних нейронів, які спільно здійснюють обробку інформації на різних етапах. Кожен штучний нейрон представляє собою вузол нейронної мережі, що складається з набору вхідних сигналів і ваг зв'язків, а також використовує функцію перетворення та функцію активації.

У сучасному світі існує безліч різних моделей для обробки природної мови (NLP). Ця галузь штучного інтелекту зосереджена на тому, як комп'ютери

можуть аналізувати та відтворювати природну мову, якою користувачі спілкуються. До найвідоміших моделей належать мовна модель "трансформер", якою працюють ChatGPT, BingAI, Claude 1 та Perplexity.

Таким чином, нейромережі стають все більш популярними завдяки їхній ефективності у багатьох сферах, включаючи обробку текстової інформації, машинне навчання та аналіз даних. Різноманітні моделі, які базуються на штучних нейронах і навчаються на великих обсягах даних, розширюють можливості застосування штучного інтелекту та сприяють розвитку нових технологій. Нейромережі продовжують ставати об'єктом інтенсивного дослідження і розвитку, а їхні можливості ще далеко не вичерпані.

РОЗДІЛ 2
СЕРВІСИ ОБРОБКИ ПРИРОДНЬОЇ МОВИ

У кінці листопада 2022 року відбулася значуща подія у світі штучного інтелекту – запуск універсального чат-бота під назвою ChatGPT, що був створений компанією OpenAI. Впродовж неповного року з моменту його виходу на ринок, цей інноваційний інтелектуальний бот швидко завоював широку популярність та отримав підтримку для різноманітних сфер застосування. Ця нова технологічна хвиля досягнень обіцяє розпочати революцію у багатьох професійних галузях та відкриває перед людством нові перспективи щодо взаємодії між людьми та штучним інтелектом. ChatGPT став значущим кроком у напрямку розвитку розумних асистентів та систем штучного інтелекту, які можуть взаємодіяти з користувачами на більш глибокому та інтелектуальному рівні. Цей чат-бот відкриває нові можливості для обробки інформації, навчання та допомоги в багатьох галузях, включаючи медицину, освіту, бізнес, інформаційні технології та багато інших. Його запуск справедливо розглядається як момент перелому у розвитку штучного інтелекту та роботизації.

Після вражаючого успіху ChatGPT на початку 2023 року Microsoft у співпраці з OpenAI виводить на ринок свого власного чат-бота під назвою BingAI. Цей інноваційний бот, вбудований у їхній фірмовий браузер Edge, стає ключовою частиною їхньої стратегії для забезпечення користувачів сучасними інтелектуальними асистентами. Взявши до уваги його популярність, BingAI був інтегрований усередині осіннього оновлення Windows 11 23H2 під назвою Copilot. Це оновлення підняло функціонал BingAI на більш широкий рівень, роблячи його доступним як основний AI-помічник для користувачів у всій операційній системі. Тим часом, компанія Anthropic, заснована колишніми працівниками OpenAI, запускає свою власну модель для генерації тексту – Claude Instant, разом з чат-ботом під назвою Claude. У той самий період часу на ринку також з'явився чат-бот Perplexity, додатково розширюючи вибір інтелектуальних інтерфейсів та систем штучного інтелекту для користувачів. Ці

події свідчать про те, що ринок чат-ботів і інтелектуальних асистентів розвивається високими темпами і пропонує більше можливостей для взаємодії з технологією й інформацією та відображає важливий тренд в індустрії штучного інтелекту – зростаючий інтерес до розробки різноманітних інтелектуальних асистентів та текстових генераторів.

2.1 Веб-сервіс ChatGPT

ChatGPT – це розмовний чат-бот, який здатен на велику кількість завдань, включаючи спілкування, пошук помилок у коді, написання віршів, створення сценаріїв та обговорення тем тощо. Ця модель базується на GPT-3.5, одній з передових мовних моделей для безкоштовного використання і на основі GPT-4, більш розвиненої мовної моделі, доступної за додаткову плату. Вона вміє відповідати на запитання і виконувати завдання, що вимагають генерації тексту, а також надавати різні типи інформації та рекомендацій. Крім того, ChatGPT може взаємодіяти з користувачем у формі розмови, приймати і запам'ятовувати інформацію про попередні запити і уникати суперечливих тем. Окрім цього, існують платні доповнення, які дозволяють генерувати зображення, музику та відео, а також отримувати доступ до Інтернету.

Відповіді чат-бота можна налаштувати шляхом використання навідних запитань. Цей підхід базується на процесі навчання з підкріпленням та отриманні зворотного зв'язку від людини – RLHF. Це додатковий рівень навчання, завдяки якому ChatGPT навчається відповідати на запити відповідно до інструкцій та генерувати відповіді, які нагадують ті, що дає людина. Цей бот був розроблений за допомогою суперкомп'ютера Azure AI та піддавався навчанню на основі великого обсягу текстових даних з Інтернету. В процесі розвитку моделі нейромережу було перенавчено кілька разів, використовуючи власні відповіді бота, щоб зробити їх більш точними та відповідними. OpenAI прагнули зробити цю систему штучного інтелекту легкодоступною, точною та все більше схожою на відповіді, які може надавати людина.

ChatGPT працює на основі технології "мовної моделі" (Language Models) принцип дії якої можна порівняти його з T9 на мобільних телефонах, який

передбачає наступне слово на основі вже введеного тексту. Тобто, один із основних принципів ChatGPT є передбачення наступного слова в тексті, яке є дуже простою, але важливою дією. Взагалі, мовне моделювання полягає в тому, що модель визначає, яке слово може бути додане після вже наявного тексту. Для цього вона використовує ймовірності виникнення різних слів в даному контексті. Це лише рівняння y=kx+b, в якому T9 або ChatGPT обчислюють наступне слово (y) на основі попередніх слів (x), які вже введені. Основне завдання при тренуванні мовних моделей полягає в підборі коефіцієнтів для цих рівнянь, так щоб вони правильно передбачали тексти на основі попередніх слів.

Більші моделі, які мають багато параметрів, називаються великими мовними моделями LLM. На практиці, мовні моделі намагаються передбачити ймовірності різних слів, які можуть бути використані для продовження тексту. Це означає, що під час генерації наступного слова такі моделі обирають його "за допомогою випадковості", аналогічно до кидання кубика. Однак ця випадковість не є зовсім випадковою – вона керується ймовірностями, які вже вбудовані у модель під час її навчання на великій кількості текстових даних. Це означає, що навіть при однакових вхідних запитах одна і та ж модель може генерувати різні відповіді, схожі на те, як різні люди можуть реагувати по-різному. Навіть коли спробувати змусити нейронну мережу завжди обирати "найбільш ймовірне" наступне слово, то з великою ймовірність це не спрацює. Фактично, випадковість грає важливу роль у підвищенні різноманітності та якості генерованих відповідей. Гарна модель повинна зберігати у собі різноманітність і нюанси мови, а також враховувати контекст (попередню частину тексту, що описує ситуацію) для більш точної генерації відповідей.

Як вже було вказано раніше, ChatGPT опирається на мовну модель GPT, що розшифровується як «Generative Pre-trained Transformer» або «трансформатор, навчений генерації тексту». В цілому, Трансформер є загальним обчислювальним механізмом, який приймає вхідні послідовності даних і генерує вихідні послідовності, змінені певним алгоритмом. Оскільки практично будь-яку інформацію, включаючи текст, зображення та звук, можна

представити у вигляді числових послідовностей, Трансформери виявилися універсальними для вирішення різноманітних завдань. Вони відзначаються своєю зручністю та гнучкістю завдяки своїм простим модульним блокам, які можна легко масштабувати. У порівнянні з попередніми мовними моделями, які вимагали значно більше обчислювальних ресурсів та працювали послідовно, Трансформери значно краще справляються з завданнями, де потрібно аналізувати велику кількість даних швидко і ефективно. Завдяки цьому відбувся значний прорив в галузі обробки текстів, зокрема їх генерації. Поява GPT-1 у 2018 році продемонструвала, що архітектура Трансформера дозволяє створювати масштабовані та ефективні мовні моделі, відкриваючи безмежні можливості для подальшого розвитку і вдосконалення таких моделей.

2.1.1 Модель GPT-2

Перевірена на практиці технологія Трансформерів, особливо з GPT-1, виявилася дуже ефективною для роботи з об'ємними даними та «величезними» моделями, які включають в себе значну кількість параметрів. У 2019 році вчені з OpenAI вирішили, що час підвищити масштаби мовних моделей. Загалом, вони визначили два ключові напрямки для розширення можливостей GPT-2: розширення набору тренувальних даних (датасету) та збільшення розміру моделі (кількість параметрів). На той час не існувало великих, якісних та загальнодоступних наборів текстових даних для тренування мовних моделей, і тому команді вчених зі сфери штучного інтелекту довелося творчо підходити до їхнього знаходження. Вчені з OpenAI звернулися до найбільшого англомовного онлайн-форуму Reddit та завантажили всі гіперпосилання з повідомлень, які мали більше трьох вподобань. Загалом було отримано приблизно 8 мільйонів таких посилань, а завантажені тексти в сукупності складали 40 гігабайтів. Для порівняння: зібрані твори Вільяма Шекспіра, включаючи всі його п'єси, сонети і вірші, складаються з 850 000 слів. В середньому на одній сторінці книжки поміщається приблизно 300 англійських слів. Отже, 2800 сторінок тексту Вільяма Шекспіра займають приблизно 5,5 мегабайта в пам'яті комп'ютера. Це в 7300 разів менше, ніж обсяг тренувальних даних GPT-2. Враховуючи, що люди

в середньому читають приблизно сторінку за хвилину, знадобиться майже 40 років, щоб наблизитися до рівня засвоєння інформації GPT-2.

Але не тільки обсяг тренувальних даних грає ключову роль у створенні вражаючої мовної моделі. Модель також повинна бути досить складною та масштабною, щоб відповідати на велику кількість інформації ефективно. Раніше було зазначено, що в мовних моделях (в дуже спрощеному вигляді) можна представити рівняння, де *y* визначає наступне слово, а *x* – слова, що надходять на вхід, і модель намагається передбачити його ймовірність. Таким чином, модель має багато параметрів у рівнянні, яке описує її роботу. Наприклад, найбільша модель GPT-2 у 2019 році мала понад півтора мільярда таких параметрів. Навіть просто записуючи таку кількість чисел у файл і зберігаючи їх на комп'ютері, це займе 6 гігабайт пам'яті. З одного боку, ця кількість є значно меншою, ніж загальний обсяг текстових даних, на яких модель навчалася. З іншого боку, моделі не потрібно запам'ятовувати ці дані повністю; їм просто необхідно виявити певні закономірності (шаблони, правила), які можна витягти з текстів, написаних людьми. Ці параметри формуються під час тренування моделі, потім зберігаються і залишаються незмінними. Під час використання моделі до цього великого рівняння кожного разу підставляються різні значення x (слів у вхідному тексті), але параметри рівняння (числові коефіцієнти k при x) залишаються незмінними. Зазвичай, чим більше параметрів в рівнянні, що вбудоване в модель, тим краще модель здатна передбачити ймовірності, і, відповідно, тим правдоподібнішим буде згенерований текст.

У момент випуску моделі GPT-2, її текстові вирази раптово набули вражаючої якості, що призвело до обурення дослідників OpenAI і викликало обмеження доступу до моделі з побоювань щодо безпеки. Попередні моделі, зокрема GPT-1, здатні були відповісти на питання, наприклад, про відвідування банку або аптеки. Однак GPT-2 вже миттєво відтворила есе, де від імені підлітка обговорювалися фундаментальні зміни, необхідні для ефективного реагування на зміну клімату. Це есе було анонімно відправлене на конкурс і журі не виявило жодних вад. GPT-2 була представлена у 2019 році і виявилася справжним

досягненням, перевершуючи як свою попередницю за розміром тренувальних текстів, так і за кількістю параметрів у 10 разів. Цей кількісний стрибок дозволив моделі вивчити нові навички, від написання великих есе до вирішення складних завдань.

2.1.2 Модель GPT-3

У 2020 році вчені з OpenAI взяли ту ж саму модель і розширили її ще більше – майже в 100 разів. Загалом, нова версія під назвою GPT-3 може похвалитися вражаючою кількістю параметрів – аж 175 мільярдів, що в 116 разів перевищує обсяг попередньої версії, GPT-2. Нейромережа стала важити неймовірних 700 гігабайт. І, слід відзначити, набір даних для навчання GPT-3 також був значно поліпшений та збільшившись приблизно в 10 разів до 420 гігабайт, включаючи в себе безліч книг, Вікіпедію і текстовий контент з різних джерел в Інтернеті. Однак виникає цікавий факт: на відміну від GPT-2, сама модель тепер є більшого розміру (700 ГБ), ніж весь обсяг текстових даних для її навчання (420 ГБ). Схоже, це парадоксальне явище, де під час навчання «нейромозок» генерує інформацію про різноманітні взаємозалежності всередині даних, яка перевищує по обсягу вихідну інформацію. Такий процес узагальнення дозволяє моделі ще краще вирішувати завдання на створення текстів, зокрема в тих випадках, коли під час навчання мало було відповідних прикладів. Тепер не треба навчати модель розв'язувати конкретні завдання – відомостей про те, як вирішувати певну проблему, є вже достатньо, і GPT-3 розуміє, що потрібно від неї в кожному конкретному випадку.

І знову виявилося, що GPT-3 з легкістю випереджає багато спеціалізованих моделей, які існували до неї: наприклад, переклад текстів з французької або німецької мови на англійську став справлятися GPT-3 краще і ефективніше, ніж будь-які інші нейромережі, призначені саме для цього завдання. Ще більше зацікавлює той факт, що GPT-3 змогла вчити себе математиці.

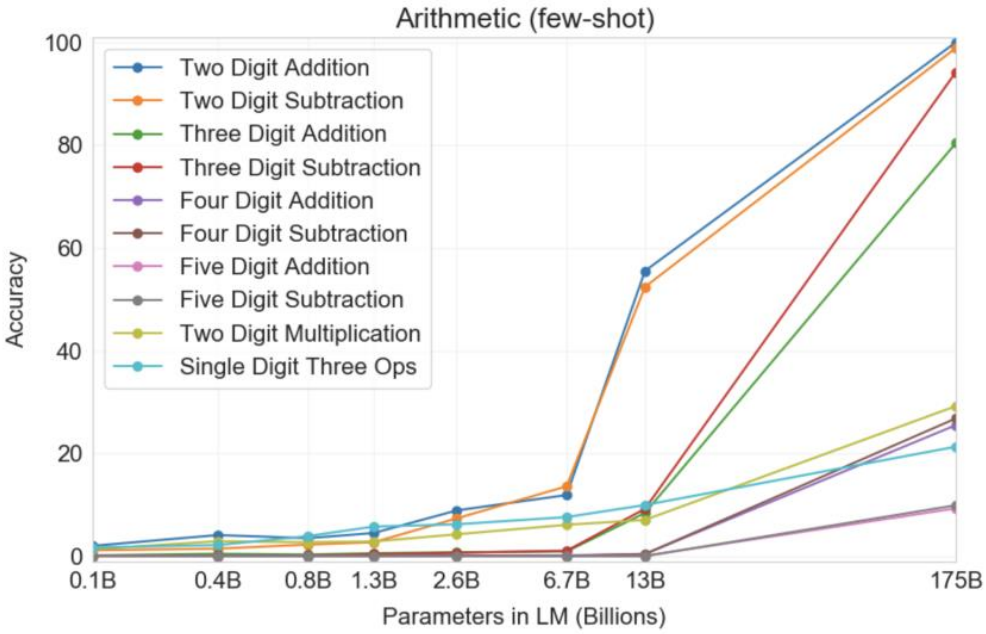


Рис. 2.1 – Графік самонавчання GPT-3

На рис. 2.1 показана точність відповідей нейромереж з різними кількостями параметрів на завдання, пов'язані з арифметикою, включаючи додавання та віднімання, а також множення чисел, навіть п'ятизначних. Як можна помітити, під час переходу від моделей з 10 мільярдами параметрів до 100 мільярдів, нейромережі раптово й дуже відзначено починають «засвоювати» математику. Мовну модель навчали продовжувати текст словами і в ході цього процесу виявилося, що вона спроможна самостійно розуміти, що, наприклад, коли їй подають «378 + 789 = », відповіддю є «1167», а не інше число. На рис. 2.1 видно, що під час збільшення розміру моделі спочатку не відбувається суттєвих змін, але потім настає значний прорив, і GPT-3 починає «розуміти», як вирішувати різноманітні завдання.

Тепер модель уміє розв'язувати задачі і це по суті, процес коли ми надаємо їй певний початковий текст (промпт) та вона продовжує його. Якщо її продовження відповідає нашим очікуванням, це означає, що модель успішно виконала завдання. Точніше прописаний промпт, який описує нашу мету докладніше, допомагає моделі зрозуміти, що вимагається від неї. Без такого опису або прикладів в промпті, модель, можливо, зрозуміє загальну ідею завдання, але може надати менш якісні відповіді. Конкретний та докладний

промпт допомагає GPT краще оцінювати імовірності слів, які вона генерує у відповідь. Додавання лише однієї фрази перед запитом, такої як «будь ласка» або «давайте думати крок за кроком» різко підвищує якість відповідей моделі. Цей «режим міркування» є однією з нововведень, що з'явилися в великій моделі GPT-3 після подолання планки в сто мільярдів параметрів. Попередні моделі з меншою кількістю параметрів не володіли такими навичками без спеціальних напутніх слів.

2.1.3 Модель GPT-3.5

Мовні моделі не схожі на людей і часто доводиться подавати їм чіткі вказівки та докладати зусиль, щоб пояснити очевидні речі. Наприклад, фраза «давай подумаємо крок за кроком» є прикладом такого напутнього тексту. Було б чудово, щоб мовні моделі могли самостійно генерувати більш розгорнуті та відповідні інструкції на запити, без додаткових вказівок від людей. Зрозуміло, що відповіді мають бути точними та корисними, але також безпечними та нетоксичними. Це викликає проблеми «вирівнювання ШІ» (AI alignment), які мають численні етичні аспекти. Головний виклик полягає в тому, що складні ситуації такого роду не завжди можна чітко формалізувати. Однак дослідники знайшли спосіб, щоб покращити відповіді мовних моделей – це надавати їм зворотний зв'язок. Модель InstructGPT (також відома як GPT-3.5) була вдосконалена, навчаючи її за допомогою оцінок живих експертів RLHF. Власне кажучи, група людей оцінювала численні відповіді мовної моделі, щоб з'ясувати, наскільки вони відповідають очікуванням, враховуючи поставлений запит. Цим же експертам доручали навчити мовну модель ще одному завданню: як налагодити згенеровану відповідь так, щоб вона отримувала найвищу оцінку від живої людини. З погляду загального процесу навчання моделі, цей останній крок «навчання на живих людей» становив не більше 1% від усього процесу. Але саме ця доробка виявилася ключовою для зроблення останніх моделей з серії GPT насправді вражаючими. Виявляється, що GPT-3 вже мала всі необхідні знання, такі як розуміння різних мов, знання історичних подій, вміння розрізняти стилі різних авторів і так далі. Проте лише завдяки зворотньому зв'язку від самих

людей модель змогла вивчити, як використовувати ці знання так, як це вважається «правильним» з людської точки зору. Після цього, в листопаді 2022-го вийшла модель ChatGPT і з технічного погляду там не було ніяких особливих нововведень, максимум що модель дотренували на додатковому діалоговому наборі даних. Адже є речі, які специфічні саме для роботи «ШІ-асистента» у форматі діалогу: наприклад, якщо запит користувача незрозумілий, то можна поставити уточнювальне запитання і так далі. Також у нього з'явився зручний інтерфейс взаємодії і відкритий публічний доступ.

2.1.4 Модель GPT-4

14 березня 2023 року OpenAI разом с Microsoft представила наступну версію мовної моделі GPT-4. Однією з її ключових інновацій є здатність розуміти не лише текст, але і зображення. GPT-4 може розпізнати, що саме представлено на фотографії та навіть пояснити її зміст. Ця здатність інтерпретувати візуальну інформацію перетворила GPT-4 в потужний інструмент для створення підписів до зображень, надання візуальних відповідей на питання та генерації вмісту. Інтеграція текстового та візуального розуміння дала GPT-4 потенціал для революційних змін у різних галузях, таких як реклама, дизайн і електронна комерція.

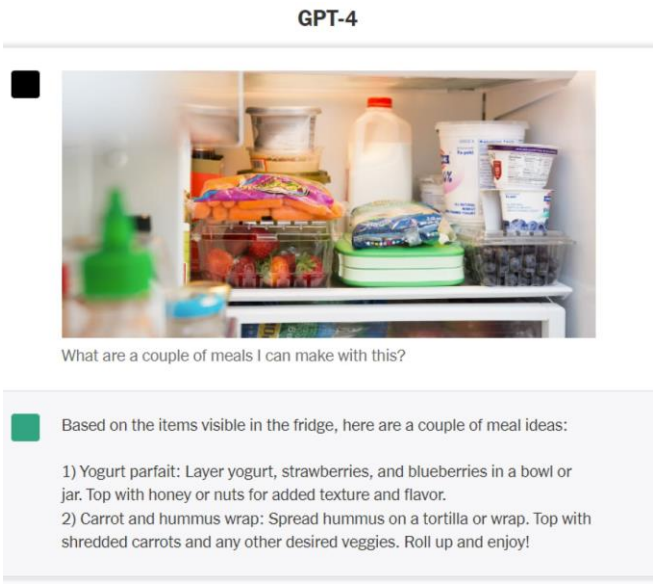


Рис. 2.2 – Приклад роботи GPT-4 з фото

GPT-4 володіє вражаючою пам'яттю. Порівняно з попередньою версією, яка могла запам'ятати лише 8 000 слів, GPT-4 без зусиль вміщує в пам'ять 25 000 слів. Ця розширена пам'ять дозволяє моделі легко згадувати інформацію, включаючи дані з попередніх розділів розмови, навіть після деякого часу. Така здатність поліпшує здатність GPT-4 генерувати послідовні та зв'язні відповіді, роблячи його важливим інструментом для завдань у галузі обробки природної мови. Також, OpenAI вже анонсувала версію «супер-ШІ», яка вміщує близько 64 000 слів. Крім цього, GPT-4 вже може відповідати на запитання 26 різними мовами. У порівнянні з англійською версією GPT-3.5 та іншими LLM, GPT-4 виявив кращу продуктивність у 24 з 26 перевірених мов, включаючи ті, які є мовами з обмеженим ресурсом, такими як латиська, валлійська та суахілі. Ця інновація в галузі обробки природної мови істотно підвищить якість багатомовного комунікації та доступність. Це значний крок у напрямку подолання мовних бар'єрів у різних сферах, таких як освіта, охорона здоров'я та бізнес.

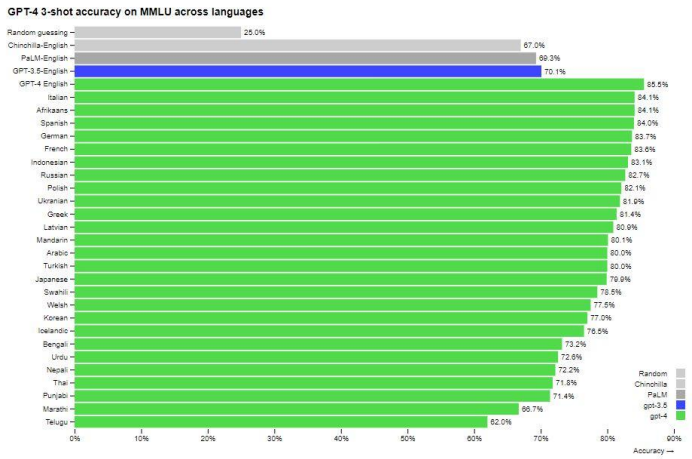


Рис. 2.3 – Діаграма продуктивності розуміння мов GPT-4

GPT-4 був підданий більш ніж 20 різноманітних тестів, охоплюючи математику, фізику і хімію. В окремих випадках, модель продемонструвала результати, які перевершили результати більш ніж 88% тих, що були протестовані. Цей прорив підкреслює потенціал GPT-4 у зміні правил гри в таких

галузях, як освіта, наукові дослідження і обслуговування клієнтів. Проте перед повною інтеграцією його в ці сфери потрібні подальші тестування та доробки.

Основні різниці між GPT-4 та GPT-3 такі:

– GPT-4 виявився менш схильним до фактичних та інших помилок, які були перевірені за допомогою різних тестів. Наприклад, при стандартному іспиті на ліцензію адвоката в США, GPT-4 показав значно кращі результати, увійшовши до топ 10% учасників, в той час як GPT-3 опинився в нижній десятці;

– ця нова модель може створювати та редагувати тексти, включаючи пісні, п’єси тощо, на близькому до людського рівні. Вона також здатна імітувати авторський стиль або інший стиль за бажанням користувача.

– модель може розпізнавати ілюстрації та взаємодіяти з ними, надаючи, наприклад, рецепти на основі продуктів, зображених на малюнку. Крім того, вона використовується для опису навколишнього середовища для людей із вадами зору, співпрацюючи з програмою Be My Eyes.

– GPT-4 може обробляти одночасно більше 25,000 слів, що вдвічі більше, ніж GPT-3, що відкриває можливості для роботи з великими обсягами тексту.

– модель стала більш багатомовною, здатною працювати з 25 мовами, включаючи італійську, українську та корейську, з високою точністю, хоча найкраще працює англійською.

– GPT-4 доступний у платній версії ChatGPT Plus та через API, і вже використовується такими компаніями, як Duolingo, Stripe, Morgan Stanley, Khan Academy та уряд Ісландії, а також однією з провідних програмних компаній США, Salesforce.

У вересні 2023 року в закритому бета-тестуванні застосунку ChatGPT на платформі Android було додано нову функцію «розмови» з чатом. Користувачі можуть вказати запит голосом та отримати реальну голосову відповідь. Ця функція використовує нейромережу для створення голосу, що звучить природно, зберігаючи акцент, паузи та інші аспекти мови. ChatGPT підтримує 26 мов та допомагає іноземцям вивчати правильну вимову незнайомих слів. GPT-4 вже використовується в різних сервісах, включаючи BingAI.

2.2 Веб-сервіс BingAI

У лютому 2023 року Microsoft представила оновлений варіант свого браузера Edge та впровадила потужну нейромережу у вигляді чат-бота в Bing. BingAI – це розширена пошукова система, яка використовує штучний інтелект для не лише знаходження інформації, але й для короткого узагальнення тексту. Цей інструмент дозволяє швидко знаходити відповіді на питання, включаючи ситуації, коли раніше потрібно було переглядати багато статей без успіху. Проте, звичайний пошук без використання штучного інтелекту також залишається доступним.

Чат-бот Bing AI представляє значний крок у галузі штучного інтелекту і має ряд відмінностей в порівнянні з популярним інтелектуальним асистентом ChatGPT:

– нові функції. Чат-бот BingAI є новою функцією в пошуковій системі Bing, яка забезпечує кращі результати, більш повні відповіді та здатність генерувати контент;

– інтеграція в пошукову систему. Чат-бот BingAI містить результати індексації великої кількості сайтів, може переформулювати запитання користувача в пошукові запити та підбирати відповідний контент;

– оновлення даних. На відміну від ChatGPT, який базується на даних, актуальних станом на 2021 рік, чат-бот BingAI активно оновлює свою базу даних щодня, забезпечуючи актуальність інформації;

– надання посилань. BingAI надає клікабельні посилання та адреси джерел інформації під віконцем відповіді, що додатково підтверджує достовірність інформації. У відміну від цього, ChatGPT може надавати посилання тільки за запитом користувача, і іноді ці посилання можуть бути недостовірними;

– підказки для додаткової інформації. Чат-бот Bing AI виводить варіанти додаткових запитів, які допомагають користувачам отримати більше відомостей з обраної теми;

– генерація фото. Використовуючи технологію Dalle-3 від тих самих OpenAI BingAI може згенерувати будь-яку картинку за запитом, на відміну від ChatGPT, який вміє так лише через сторонні плагіни в платній підписці.

В осінньому обновленні Windows 11 23H2 Microsoft офіційоно інтегрує нового AI-помічника прямо в операційну систему, під назвою Windows Copilot. До цього він був доступний лише в інсайдерській збірці Windows 11 23493 на каналі розробників.

Отже, користувач має можливість виконувати дії та налаштовувати параметри свого ПК, а також безперешкодно з'єднуватися зі своїми улюбленими програмами. Тут він може надати команду, яку Copilot намагатиметься точно виконати. Користувач може управляти налаштуваннями свого комп'ютера, легко змінювати налаштування дисплея, такі як активація темного режиму, Wi-Fi та Bluetooth, просто надаючи відповідні команди своєму ПК. Крім того, можна, наприклад, запитати Windows Copilot: «Я хочу стати більш продуктивним. Як я можу налаштувати свій ПК?». Інтелектуальний асистент надасть рекомендації, які сприяють підвищенню продуктивності та покращенню концентрації користувача. Що ще важливіше, помічник має здатність обробляти документи. Це означає, що людина можете просто перетягнути текстовий файл на панель Windows Copilot, а він в свою чергу, здатний переписати, узагальнити або пояснити його вміст. Завдяки неймовірним можливостям Windows Copilot, користувач може легко вставляти зображення, текст і відео у свій чат та просити поділитися цими матеріалами у груповому чаті Microsoft Teams. Оскільки команда розробників продовжує працювати над розширенням нових плагінів, незабаром Windows Copilot буде інтегровано з усіма програмами. Окрім роботи з функціями та налаштуваннями Windows, завдяки інтеграції Bing Chat, користувач може взаємодіяти з Windows Copilot на різні теми та навіть ставити складні запитання. Наприклад, можна попросити Copilot розповісти про поточну погоду в Гренландії, найкращий час для відпочинку там, а також дізнатися можливі варіанти готелів і авіарейсів.

Головна мета Windows Copilot полягає в тому, щоб зробити кожного власника Windows 11 досвідченим користувачем, який легко може налаштовувати параметри Windows, підвищувати продуктивність та використовувати можливості штучного інтелекту, що надаються Windows Copilot у різних програмах за допомогою потужних плагінів.

2.3 Веб-сервіс Claude 1

У тому ж 2023 році стартап Anthropic, створений колишніми співробітниками OpenAI, представив оновлену версію своєї моделі для генерації тексту під назвою Claude Instant та чат-бота Claude. Anthropic отримав підтримку від Google, яка інвестувала $300 млн у них. Чат-бот Claude може генерувати тексти, відповідати на запитання, а також писати та редагувати код. Користувачі можуть налаштувати його, вказавши бажаний стиль відповідей, характер та поведінку. Anthropic прагне створити «корисний та об'єктивний» штучний інтелект. На даний момент існує три види мовних моделей: Claude 1, Claude 2 та Claude Instant.

Claude 2 є наступником Claude 1 та навчився на більшому обсязі даних для кращої роботи з великими обсягами контексту, в той час як Claude Instant – швидша та більш ефективна модель, ідеально підходить для неформальних розмов, аналізу тексту, узагальнення і відповідей на запити, базовані на документах.

Таблиця 2.1

Основні відмінності між Claude 1, Claude 2 і Claude Instant

Характеристика	Claude 1	Claude 2	Claude Instant
Розмір моделі	137B параметрів	175B параметрів	100K токенів
Можливості	складний діалог, створення креативного контенту, детальні інструкції	всі функції Claude 1 + академічні функції та багато іншого	невимушений діалог, аналіз та узагальнення тексту, питання-відповіді до документів

Продовження табл. 2.1

Вартість за токен	$32.68 за мільйон токенів	$32.68 / мільйон токенів	$5.51 за мільйон токенів
Продуктивність	добре справляється зі складними завданнями	відмінно справляється зі складними завданнями	добре справляється зі простими завданнями
Контекстне вікно	75 000 токенів	100 000 токенів	75 000 токенів

Claude проходить навчання на виборчому наборі даних, який усуває потенційно шкідливий контент, і Anthropic надзирала за його роботою, щоб забезпечити безпеку. Важливо відзначити, що дані для навчання Claude були відсічені в грудні 2022 року, хоча деякі події можуть бути відомі і на початку 2023 року. Оновлені дані дають Claude перевагу над ChatGPT, який припинив отримання нових даних у вересні 2021 року, і це одна з причин, чому Claude вважається кращим варіантом.

Ось декілька обґрунтувань, чому Claude перевершує інших чат-ботів зі штучним інтелектом:

– розуміння. Завдяки розширеним можливостям обробки даних і оновленим даним, Claude може надати корисну інформацію про ручно введені дані;

– швидкість і точність. Claude здатний обробляти великі обсяги даних швидко і точно, залишаючи точність незмінною, використовуючи передові алгоритми та методи машинного навчання;

– нешкідливість. Незважаючи на свою потужність і складність, Claude є безпечним і створений для допомоги людям, розширюючи їхні можливості;

– конституційний ШІ. Anthropic використовує метод конституційного ШІ для навчання Claude, що використовує правила та принципи для управління поведінкою Claude та робить його більш передбачуваним, контрольованим і зручним для користувачів.

Claude обладнаний численними високоякісними функціями чат-бота, а його унікальний підхід до навчання робить його ймовірно безпечнішим, ніж інші інструменти штучного інтелекту.

2.4 Веб-сервіс Perplexity

Починаючи з 2002 року, проводилася інтенсивна розробка нейромережі під назвою Perplexity, призначеної для оцінки якості генерації тексту моделями. Показником ефективності цих моделей є оцінка перплексії, яка визначає, наскільки точно модель передбачає наступне слово в тексті. На практиці, нижче значення перплексії вказує на кращу роботу моделі. Важливо зауважити, що перплексія не вимірює точність або повноту моделі, але допомагає визначити, наскільки успішно модель адаптується до нових даних. Перплексія розраховується як зворотна величина ймовірності послідовності тестових слів, і чим вища ймовірність, тим нижча перплексія. Іншими словами, перплексія визначає здатність моделі передбачати послідовності слів, які не були використані в процесі її навчання на тренувальному наборі даних. Основною перевагою метрики Perplexity є її здатність вимірювати ефективність моделі в її основному завданні. Зменшення значення перплексії вказує на те, що модель працює краще в генерації тексту та інших лінгвістичних завданнях.

Важливо підкреслити, що Perplexity може бути використана не тільки для оцінки роботи нейронних мереж, але й для порівняння різних моделей між собою. Це допомагає обрати найоптимальнішу модель для конкретного завдання та досягти найкращих результатів. Крім того, Perplexity є відносно простою метрикою, що полегшує її використання та розуміння результатів оцінки моделі. Ця метрика також може бути застосована до різних мов і мовних моделей.Таким чином, Perplexity є корисним інструментом для оцінки якості роботи нейронних мереж та вибору найбільш підходящої моделі для конкретного завдання.

У 2023 році, в зв'язку з загальним інтересом, Perplexity також представляє свого чат-бота на основі GPT-3.5 (або GPT-4 для зареєстрованих користувачів), який доступний для використання у мережі. Як і інші чат-боти Perplexity використовує обробку природної мови для точного розуміння користувацького

запиту та його інтенції. Після аналізу запиту система використовує машинне навчання для надання найбільш релевантних результатів. Ця процедура виконується значно швидше, ніж у традиційних пошукових системах, і забезпечує користувачу більш точні результати. Perplexity також має ряд корисних функцій, які покращують процес пошуку. Наприклад, система підтримує функцію «Збережені пошукові запити», яка дозволяє користувачам зберігати свої запити для майбутнього використання. Це особливо зручно для тих, хто проводить дослідження у певній темі та потребує швидкого доступу до інформації. Результати, надані Perplexity, також більш докладні порівняно з традиційними пошуковими системами. Замість простого списку посилань на веб-сайти, Perplexity також надає розширену інформацію, пов'язану із запитом, включаючи пов'язані теми, зображення, відео та інше. Це поліпшує загальний процес пошуку та дозволяє користувачам швидше знаходити необхідну інформацію.

Загалом, Perplexity представляє собою відмінний вибір для тих, хто бажає отримати більш ефективну та точну пошукову систему. Завдяки обробці природної мови та можливостям машинного навчання, користувачі отримують найбільш релевантні та вичерпні результати, а додаткові функції роблять процес пошуку ще ефективнішим.

Висновок за розділом 2

2022 та 2023 роки були значущими для розвитку індустрії штучного інтелекту та інтелектуальних асистентів. Запуск ChatGPT компанією OpenAI став ключовою подією, яка відкрила двері для нових можливостей у різних сферах, включаючи медицину, освіту, бізнес та інші. Ця інновація відзначається своєю інтелектуальністю та потенціалом для покращення взаємодії між людьми та штучним інтелектом.

Після успіху ChatGPT, Microsoft виводить на ринок свого чат-бота BingAI, інтегруючи його в браузер Edge та операційну систему Windows 11. Це свідчить про стрімкий розвиток індустрії та зростаючий інтерес до інтелектуальних асистентів. Компанія Anthropic також додає свій внесок до цього ринку,

запускаючи модель для генерації тексту, Claude Instant, та чат-бота Claude. Також на ринку з'явився чат-бот Perplexity, розширюючи вибір інтелектуальних інтерфейсів та систем штучного інтелекту.

Ці події відображають важливий тренд в індустрії штучного інтелекту – зростаючий інтерес до розробки різноманітних інтелектуальних асистентів та текстових генераторів. Ринок інтелектуальних асистентів активно розвивається і надає більше можливостей для взаємодії з технологією та інформацією, відкриваючи нові горизонти для інновацій і покращень у різних галузях життя.

РОЗДІЛ 3
ДОСЛІДЖЕННЯ МОДЕЛЕЙ ОБРОБКИ ТЕКСТУ

3.1 Вибір алгоритмів для дослідження

Спочатку для дослідження було вибрано три алгоритми:

– метод Гауса для розв'язання лінійних рівнянь із трьома невідомими;

– павутинні та біфуркаційні діаграми;

– фрактал Коха.

Вибір даних алгоритмів для використання як промтів у дослідженні пов'язаний з:

– різноманітністю обчислювальних завдань. Обрані алгоритми містять різні типи обчислень. Метод Гауса використовується для розв'язання лінійних рівнянь, що є базовим математичним завданням. Павутинні та біфуркаційні діаграми використовуються для візуалізації складних динамічних систем. Фрактал Коха представляє собою графічний об'єкт і може бути використаний для тестування здатності моделей до аналізу складних візуальних даних;

– тестуванням різних аспектів інтелектуального аналізу. Дослідження охоплює як числові обчислення (метод Гауса), так і візуальний аналіз (павутинні, біфуркаційні діаграми, фрактал Коха). Це дозволить визначити, наскільки добре моделі можуть виконувати різні завдання, що вимагають різних видів інтелектуального аналізу.

– креативністю і складністю завдань. Павутинні, біфуркаційні діаграми і фрактал Коха є складними графічними структурами, і їхній аналіз вимагає креативних підходів та вміння розуміти складні візуальні патерни. Такі завдання дозволять оцінити здатність нейромережевих моделей до розв'язання складних завдань;

– загальною унікальністю. Вибір таких унікальних і різноманітних завдань дозволяє визначити, наскільки добре моделі можуть справлятися з нестандартними завданнями, що може бути важливим аспектом для практичного застосування.

Проте, під час спроби візуалізації павутинної діаграми, жодна з використовуваних нейронних мереж не досягла задовільного результату. На рис. 3.1 показаний вірний результат побудови біфуркаційної діаграми для логістичної функції. А на рис. 3.2 – 3.4 результати отримані різними нейромережевими моделями.

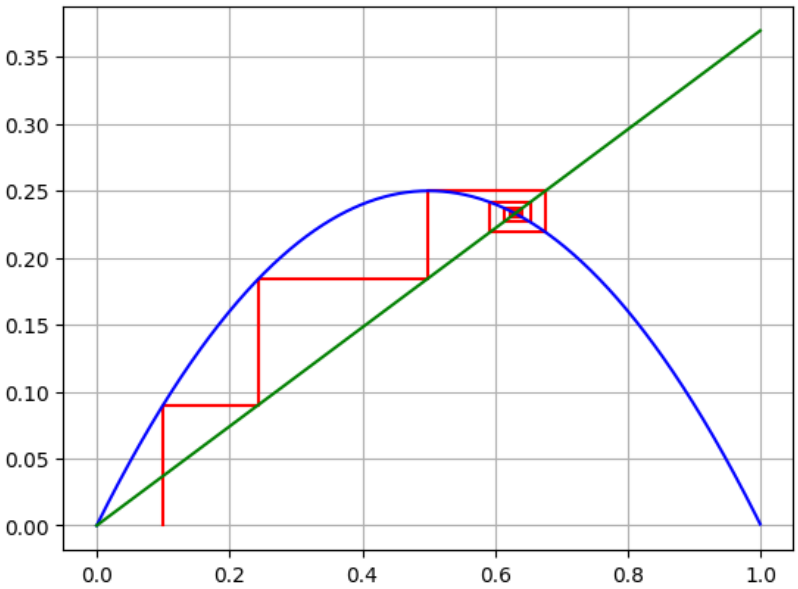


Рис. 3.1 – Правильний результат візуалізації павутинної діаграми λ = 4

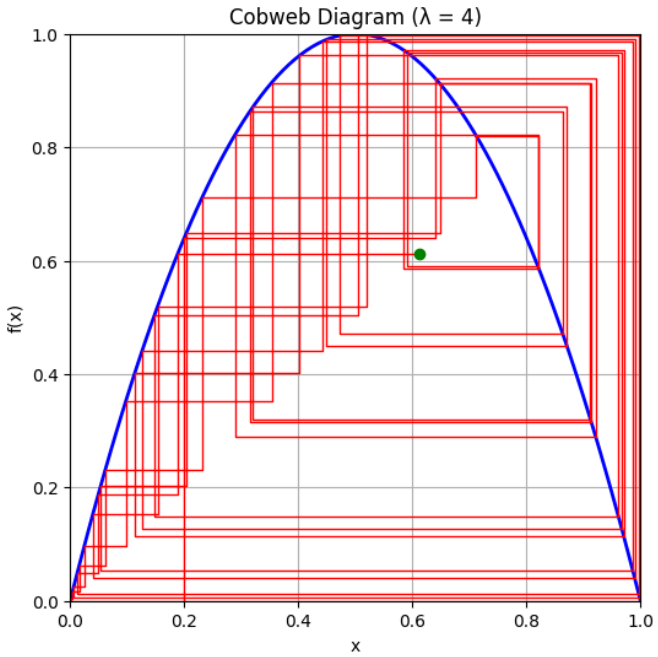


Рис. 3.2 – Результат ChatGPT

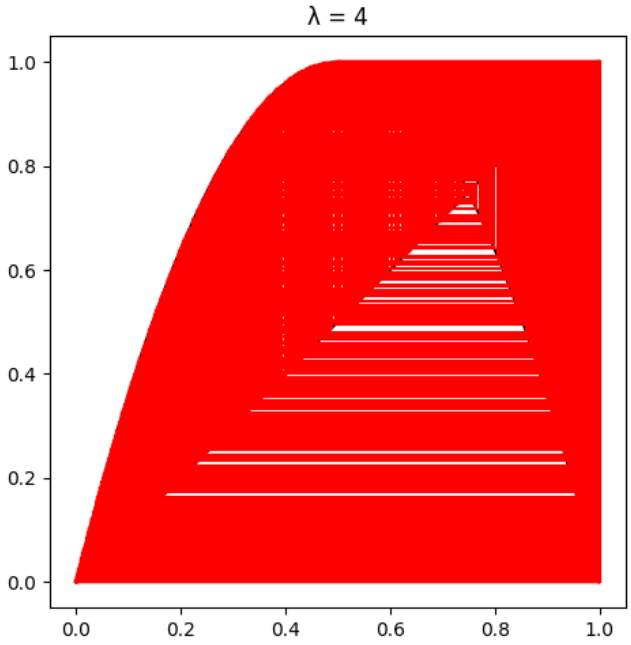


Рис. 3.3 – Результат BingAI

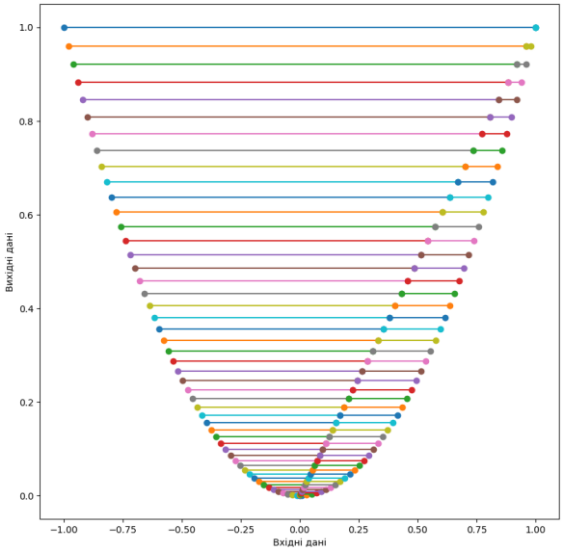


Рис. 3.4 – Результат Bard

Причиною невдалого відображення була недостатня навченість моделей GPT-3.5 TURBO, GPT-4 та LaMDA у генерації павутинних та біфуркаційних діаграм (рис. 3.2 – 3.4). Тому, павутинні та біфуркаційні діаграми були замінені на метод золотого перетину через, що він має просту та інтуїтивно зрозумілу математичну основу, яка легко інтегрується в аналіз тексту. Його застосування не потребує складних обчислень, що спрощує роботу з ним в рамках математичного аналізу тексту.

3.1.1 Метод Гауса для розв'язання системи лінійних рівнянь із трьома невідомими

Метод Гауса, також відомий як метод послідовного виключення невідомих, є одним із фундаментальних та універсальних інструментів в області розв'язання систем лінійних алгебраїчних рівнянь (СЛАР). Цей метод, який отримав свою назву на честь видатного математика Карла Фрідріха Гауса, здатний розв'язувати складні системи лінійних рівнянь, де декілька змінних взаємодіють між собою. Він є важливим інструментом в численних наукових і інженерних дисциплінах. Використовується в фізиці для моделювання складних фізичних систем, у фінансах для аналізу портфелів і ризиків, а також в інших галузях, де важливо розв'язувати системи лінійних рівнянь.

Метод Гауса полягає в послідовному виключенні невідомих, шляхом застосування елементарних операцій над рівняннями. Це дозволяє звести складну систему до більш простих рівнянь і визначити значення невідомих з точністю. Завдяки цьому методу, інженери, науковці та аналітики можуть ефективно моделювати та розв'язувати складні проблеми, що вимагають аналізу систем лінійних рівнянь. Він є важливим інструментом у розв'язанні реальних завдань та розширює можливості численних наукових галузей.

Для ефективного порівняння результатів у ході численних запитів, був розроблений універсальний запит для методу Гауса: «Створи мені код на Python для вирішення систем лінійних рівнянь методом Гауса у яких кількість невідомих величин у системі дорівнює 3. Систему лінійних рівнянь маю вводити вручну. В кінці обов'язково має бути зроблена перевірка.» Цей експеримент містив запит на створення універсального Python-коду, що дозволяло б візуалізувати та легко тестувати код, із специфікацією щодо методу Гауса та кількості невідомих величин (три), а також зазначенням, що самі рівняння повинні вводитися вручну, і обов'язковою перевіркою результатів. Для об'єктивного порівняння на вхід до системи рівнянь будуть подаватися такі значення: 1, 2, 3, 5, 2, 3, 5, 5, 5, 6, 7, 8. На виході маємо отримати $x_1 = -4.25$, $x_2 = 5.75$, $x_3 = -0.75$ та пройдену перевірку.

Запит в ChatGPT для методу Гауса. Після подання універсального запиту в ChatGPT був отриманий результат, показаний на рис. 3.5.

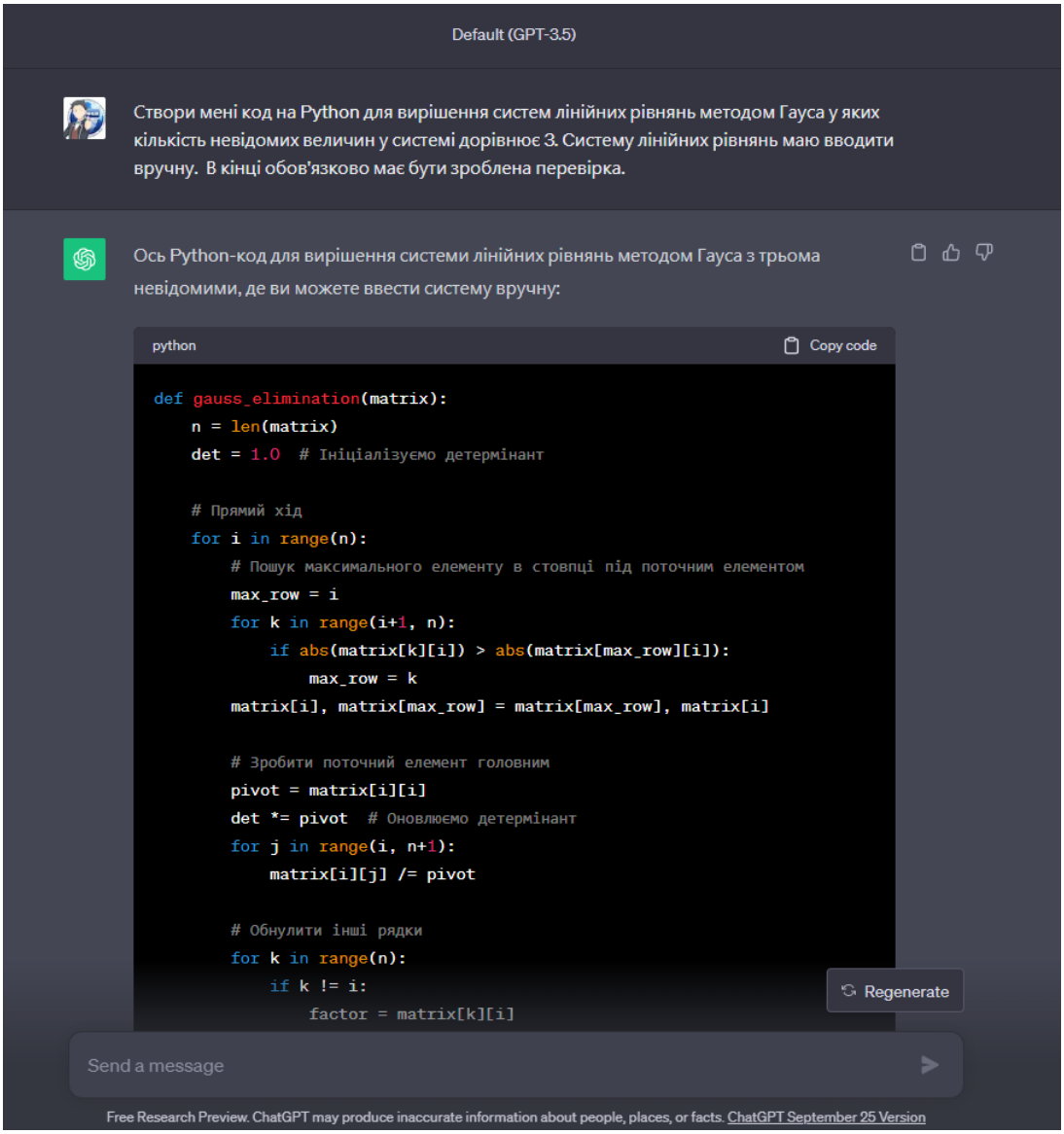


Рис. 3.5 – Результат після запиту в ChatGPT

Як можна побачити з кода, ChatGPT пішов класичним методом та реалізував функцію gauss_elimination, в якій зробив прямий та зворотній хід. Цей код реалізує метод Гауса для вирішення системи лінійних рівнянь вигляду Ax = b, де A – квадратна матриця, x – вектор невідомих, b – вектор правих частин. Для початку, він визначає розмір матриці **A** і ініціалізує детермінант як 1.0. Далі застосовує прямий хід, тобто перетворюємо матрицю A на верхньотрикутну за допомогою елементарних перетворень рядків. При цьому оновлює детермінант і

ділимо кожен рядок на головний елемент. Застосовує зворотний хід, тобто знаходить розв'язки системи з верхньотрикутної матриці, починаючи з останнього рядка. Повертає детермінант і розв'язки як результат функції. Зчитує коефіцієнти системи від користувача і зберігає їх у матрицю. Викликаємо функцію «gauss_elimination» з матрицею і виводить результат на екран. Перевіряє правильність розв'язків, підставляючи їх у початкову систему.

Отримавши код від ChatGPT та скомпілювавши його через Google Collab отримали такі результати

```
Введіть коефіцієнти системи рівнянь (Ax = b):
Елемент [1][1]: 1
Елемент [1][2]: 2
Елемент [1][3]: 3
Елемент [1][4]: 5
Елемент [2][1]: 2
Елемент [2][2]: 3
Елемент [2][3]: 5
Елемент [2][4]: 5
Елемент [3][1]: 5
Елемент [3][2]: 6
Елемент [3][3]: 7
Елемент [3][4]: 8

Результат:
Детермінант матриці: 4.0
x1 = -4.25
x2 = 5.750000000000001
x3 = -0.7500000000000004

Перевірка:
1.0x1 + 0.0x2 + 0.0x3 = -4.25
0.0x1 + 1.0x2 + 0.0x3 = 5.750000000000001
0.0x1 + 0.0x2 + 1.0x3 = -0.7500000000000004
```

Рис 3.6 – Результати коду ChatGPT

На рис. 3.6 представлені результати коду, у якому бачимо, що він запропонував послідовно ввести дані в матрицю, яка являє собою невідомі коефіцієнти в системі рівнянь. Після цього був отриманий правильно порахований результат та детермінант матриці. В кінці отримали пройдену перевірку. Тобто, можна зробити висновок, що ChatGPT повністю впорався з поставленою задачею з першої спроби.

Запит в BingAI для методу Гауса. Після подання універсального запиту в BingAI отримали такий результат – рис. 3.7.

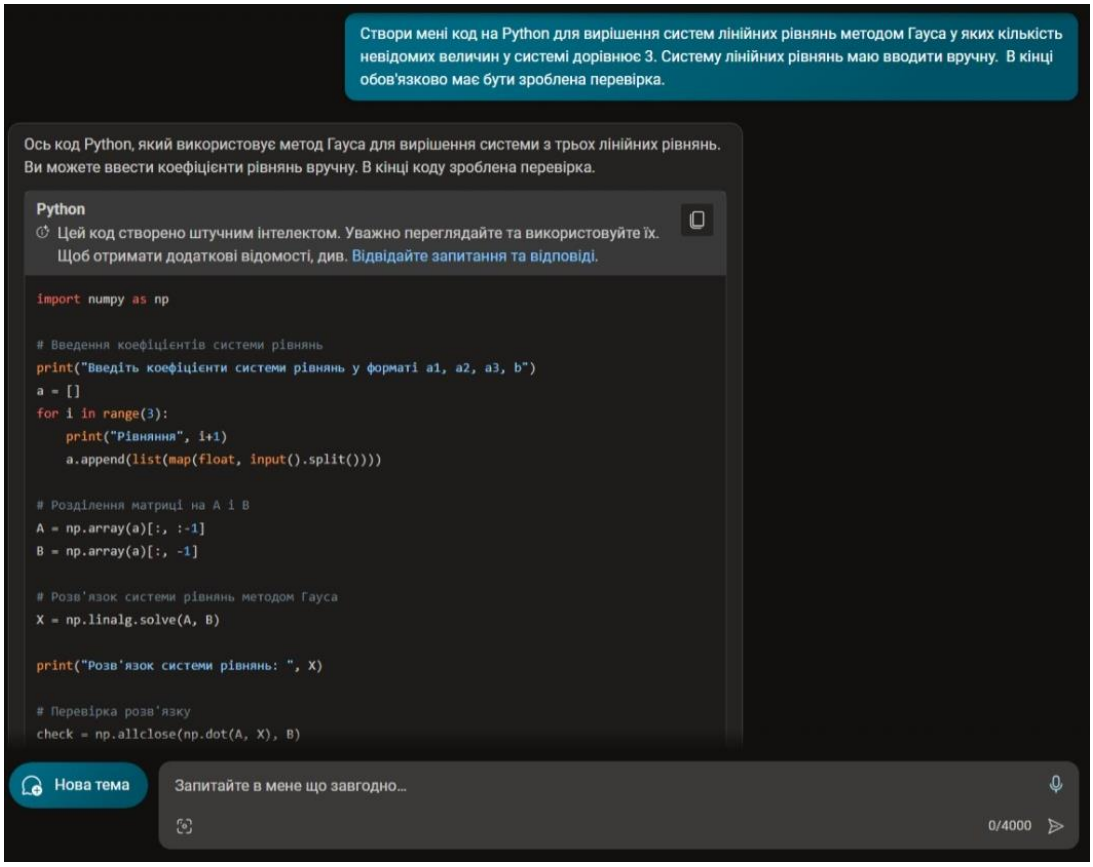


Рис. 3.7 – Результат після запиту в BingAI

Як можна побачити з кода, BingAI пішов іншим шляхом. Для початку він імпортує бібліотеку numpy для роботи з масивами і матричними операціями. Далі запитує у користувача ввести коефіцієнти трьох лінійних рівнянь з трьома невідомими у форматі a1, a2, a3, b, де a1, a2, a3 – коефіцієнти при невідомих, b – вільний член. Створює масив *a*, який містить коефіцієнти усіх рівнянь, і розділяє його на дві матриці: A, яка містить коефіцієнти при невідомих, і B, яка містить вільні члени. Застосовує метод Гауса для розв'язання системи рівнянь, використовуючи функцію np.linalg.solve(A, B). Результат зберігається у масиві X, який містить значення невідомих. Виводить на екран розв'язок системи рівнянь у форматі [x1, x2, x3], де x1, x2, x3 – значення невідомих. Перевіряє правильність розв'язку, використовуючи функцію np.allclose(np.dot(A, X), B). Якщо розв'язок правильний, виводить на екран повідомлення "Перевірка

пройшла успішно!". Якщо неправильний, виводить на екран повідомлення "Щось пішло не так. Перевірте введену систему рівнянь."

Отримавши код від BingAI та скомпілювавши його через Google Collab отримали такі результати:

```
Введіть коефіцієнти системи рівнянь у форматі a1, a2, a3, b
Рівняння 1
1 2 3 5
Рівняння 2
2 3 5 5
Рівняння 3
5 6 7 8
Розв'язок системи рівнянь:  [-4.25  5.75 -0.75]
Перевірка пройшла успішно!
```

Рис 3.8 – Результати коду BingAI

На рис. 3.8 представлені результати коду, у яких ми бачимо, що він запропонував послідовно ввести дані в матрицю. Після цього ми отримали правильно порахований результат. В кінці отримали пройдену перевірку. Тобто, можна зробити висновок, що BingAI повністю впорався з поставленою задачею з восьмої спроби.

Запит в Claude 1 для методу Гауса. Після подання універсального запиту в Claude 1 отримали такий результат – рис. 3.9.

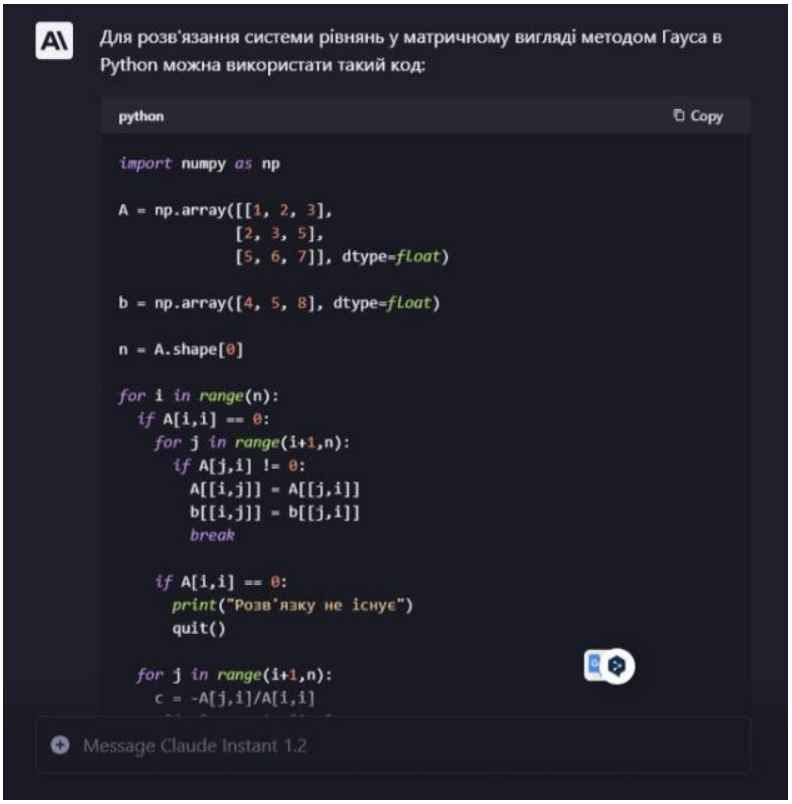


Рис. 3.9 – Результат після запиту в Claude 1

Як бачимо цей код розв'язує систему лінійних рівнянь методом Гауса трохи по-іншому. Для початку, вводиться матриця коефіцієнтів A та вектор вільних членів b. Далі, код перевіряє, чи є на головній діагоналі матриці A нулеві елементи. Якщо так, то шукає ненульовий елемент у стовпці нижче і міняє рядки матриці та вектора вільних членів так, щоб на головну діагональ потрапив ненульовий елемент. Якщо ж ненульового елемента не знайдено, то система рівнянь не має розв'язку. Потім застосовує метод Гауса до матриці A та вектора b, щоб отримати верхню трикутну матрицю. Далі, знаходить розв'язок системи лінійних рівнянь методом зворотного ходу. На виході отримує вектор `x`, який є розв'язком системи лінійних рівнянь.

Отримавши код від Claude 1 та скомпілювавши його через Google Collab отримали такі результати:

```
[-4.25  5.75 -0.75]
```

Рис 3.10 – Результати коду Claude 1

На рис. 3.10 представлені результати коду і як ми бачимо вони не дуже гарні. В результаті ми отримали лише правильно порахований результат, вхідні дані були задані в коді, перевірку ми не маємо. Тобто, можна зробити висновок, що Claude 1не дуже добре, але впорався з поставленою задачею з першої спроби.

Запит в Perplexity для методу Гауса. Після подання універсального запиту в Perplexity отримали такий результат – рис. 3.11.

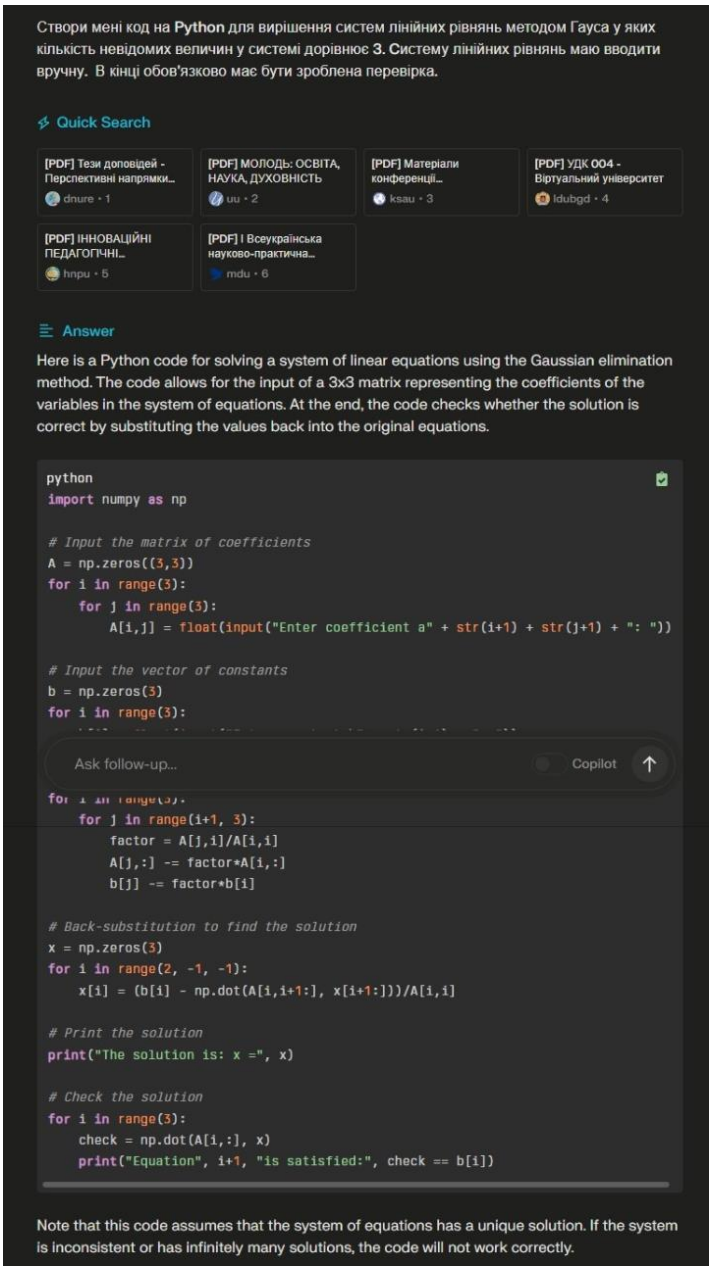


Рис. 3.11 – Результат після запиту в Perplexity

Цей код виконує розв'язання системи лінійних алгебраїчних рівнянь методом Гауса також з використанням бібліотеки NumPy. Для початку йде

створення матриці A розміром 3x3, яка ініціалізується нулями: A = np.zeros((3, 3)). Ця матриця представляє коефіцієнти лінійних рівнянь. Далі, вкладений цикл for i in range(3): і for j in range(3): вводить коефіцієнти рівнянь користувачем через функцію input(). Введені значення зберігаються у відповідних рядках та стовпцях матриці A. b = np.zeros(3) – це створення вектора b розміром 3, який ініціалізується нулями. Цей вектор представляє вектор констант у лінійних рівняннях. Вкладений цикл for i in range(3) вводить константи b користувачем через функцію input(). Введені значення зберігаються в векторі b. Далі виконується метод Гауса для розв'язання системи лінійних рівнянь. У цьому циклі, for i in range(3): обирається поточний рядок, і в циклі for j in range(i+1, 3): обираються рядки, що знаходяться нижче поточного рядка. Для кожного з таких рядків обчислюється фактор, який потрібно відняти від них, щоб зробити нульовим елемент в поточному стовпці матриці A. Відповідно змінюються як сама матриця A, так і вектор b. Після виконання методу Гауса виконується обернена підстановка для знаходження розв'язку. Починаючи з останнього рядка системи, обчислюється кожний x[i] шляхом віднімання від відповідної константи b і скалярного добутку рядка матриці A та вектора x для всіх елементів праворуч від діагоналі. На останньому кроці виводиться розв'язок у вигляді вектора x. У кінці коду виконується перевірка отриманого розв'язку, обчислюючи ліву частину кожного рівняння та порівнюючи її з відповідною константою. Результати перевірки виводяться на екран. Якщо всі рівняння задовольняються, то ви побачите повідомлення "Equation i is satisfied: True" для кожного рівняння.

Отримавши код від Perplexity та скомпілювавши його через Google Collab отримали такі результати:

```
Enter coefficient a11: 1
Enter coefficient a12: 2
Enter coefficient a13: 3
Enter coefficient a21: 2
Enter coefficient a22: 3
Enter coefficient a23: 5
Enter coefficient a31: 5
Enter coefficient a32: 6
Enter coefficient a33: 7
Enter constant b1: 5
Enter constant b2: 5
Enter constant b3: 8
The solution is: x = [-4.25  5.75 -0.75]
Equation 1 is satisfied: True
Equation 2 is satisfied: True
Equation 3 is satisfied: True
```

Рис 3.12 – Результати коду Perplexity

На рис. 3.12 представлені результати коду, у яких ми бачимо, що він запропонував ввести дані в матрицю. Після цього ми отримали правильно порахований результат та в кінці отримали пройдену перевірку. Тобто, можна зробити висновок, що Perplexity повністю впорався з поставленою задачею з другої спроби.

Висновки за методом Гауса. З отриманих результатів можна зробити висновок, що ChatGPT надав класичну реалізацію методу Гауса з прямим і зворотним ходом. Код працює коректно, має введення даних, виведення результату та перевірку. ChatGPT впорався з задачею з першої спроби. BingAI ,в свою ж чергу, використав бібліотеку NumPy для реалізації. Код теж працює вірно, має введення даних, виведення результату та перевірку. BingAI впорався з 8 спроби. Claude 1 надав просту реалізацію без введення даних і перевірки. Код працює, але не оптимальний. Claude 1 впорався з 1 спроби. Perplexity використав NumPy для реалізації з введенням даних, виведенням результату та перевіркою. Код працює вірно. Perplexity впорався з 2 спроби.

Загалом, ChatGPT та Perplexity показали найкращі результати з чіткою структурою коду. BingAI теж добре впорався, але вимагав більше спроб. Claude 1 продемонстрував найгірші результати.

3.1.2 Метод золотого перетину

Метод золотого перетину представляє собою спосіб пошуку екстремуму функції однієї змінної на заданому відрізку. Основна ідея цього методу полягає в розділенні відрізка таким чином, що він ділиться на дві частини у пропорціях, відомих як золотий перетин. Цей метод відомий своєю простотою та ефективністю у вирішенні задач оптимізації та був представлений Джеком Кіфером у 1953 році.

Основний алгоритм включає такі етапи: спочатку відрізок розділяється на дві частини симетрично відносно свого центру, і обчислюються значення функції в цих двох точках. Потім одна з крайніх точок, в якій функція приймає максимальне значення (у випадку пошуку мінімуму, або мінімальне значення для випадку пошуку максимуму), вибирається для подальшого розгляду, і інша точка відкидається. На наступній ітерації, завдяки властивостям золотого перетину, потрібно шукати лише одну нову точку. Ця процедура повторюється до досягнення необхідної точності.

Для ефективного порівняння результатів у ході численних запитів, був розроблений універсальний промт: «Реалізуй метод золотого перерізу для пошуку екстремуму функції на Python». Початкові відрізки для пошуку a = 1.764 та b = 2.236. На виході маємо отримати x = 1.764 та f(x) = 3.03.

Запит в ChatGPT для методу золотого перетину. Після подання універсального запиту в ChatGPT отримали такий результат – рис. 3.13.

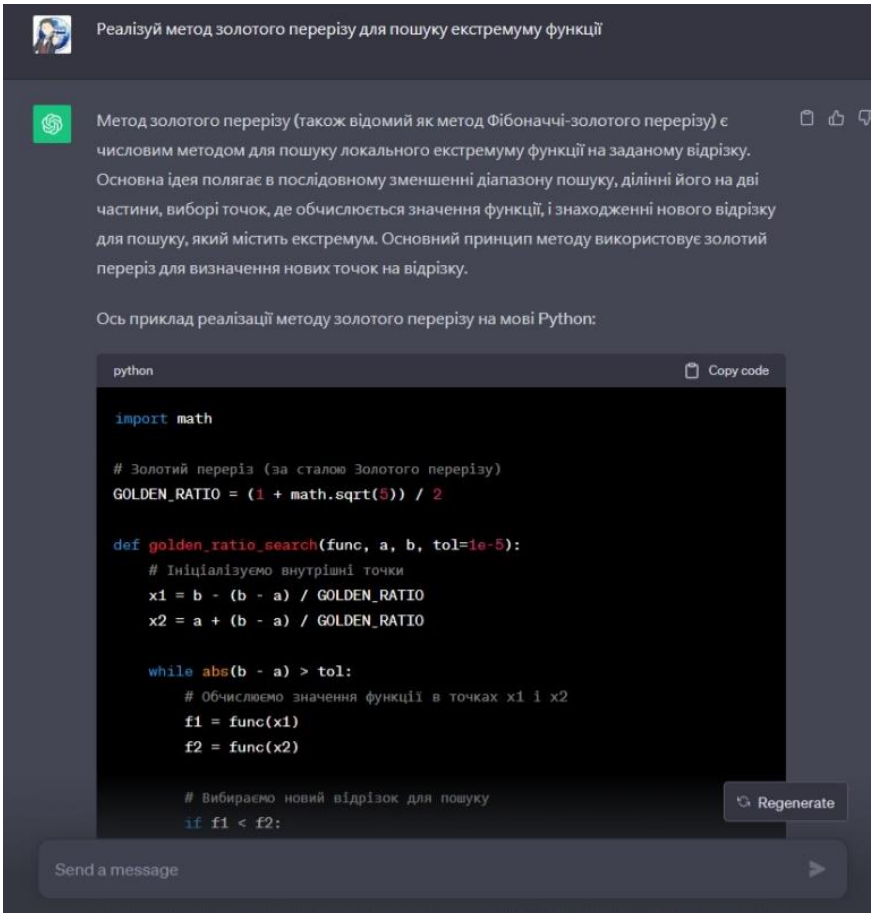


Рис. 3.13 – Результат після запиту в ChatGPT

Працює цей код так:

– GOLDEN_RATIO = (1 + math.sqrt(5)) / 2: визначається константа GOLDEN_RATIO, яка використовується в методі золотого перетину. Золотий перетин – це математична константа, визначена як (1 + √5) / 2, і він грає важливу роль у цьому методі.

– Функція golden_ratio_search(func, a, b, tol=1e-5) приймає функцію func для якої потрібно знайти мінімум, початковий відрізок [a, b], і необов'язковий параметр tol, який встановлює точність пошуку. Функція виконує пошук мінімуму на заданому відрізку.

– Спочатку визначаються дві внутрішні точки x1 та x2 від початкового відрізку [a, b] з використанням золотого перетину.

– У циклі while виконується пошук, поки різниця між b та a більша за задану точність tol.

– На кожній ітерації обчислюються значення функції func в точках x1 і x2 як f1 та f2.

– Порівнюються значення f1 та f2, і вибирається новий відрізок для пошуку:

– якщо f1 менше f2, то новий відрізок обмежується праворуч від x1;

– інакше новий відрізок обмежується ліворуч від x2.

– Після кожної ітерації оновлюються точки x1 та x2 на основі вибору нового відрізку.

– По завершенню циклу повертається середнє значення точок a та b як результат пошуку.

– test_function(x) – ця функція визначає тестову функцію, для якої шукається мінімум. У розглянутому прикладі використовується функція (x - 2) ** 2 + 3.

– Визначення початкового відрізку [a, b] для пошуку мінімуму. У прикладі: a = 0 і b = 4.

– Викликається функція golden_ratio_search з вказаною тестовою функцією та початковим відрізком.

– Виводиться результат пошуку, включаючи значення x, де досягнуто мінімуму, і значення функції в цій точці.

Отримавши код від ChatGPT та скомпілювавши його через Google Collab отримали такі результати:

```
Мінімум функції знаходиться при x = 1.7640036829538528, f(x) = 3.0235134588830253
```

Рис 3.14 – Результати коду ChatGPT

На рис. 3.14 представлені результати коду, у яких ми бачимо, що він правильно знайшов мінімум функції та її значення. Тобто, можна зробити висновок, що ChatGPT повністю впорався з поставленою задачею з першої спроби.

Запит в BingAI для методу золотого перетину. Після подання універсального запиту в BingAI отримали такий результат – рис. 3.15.

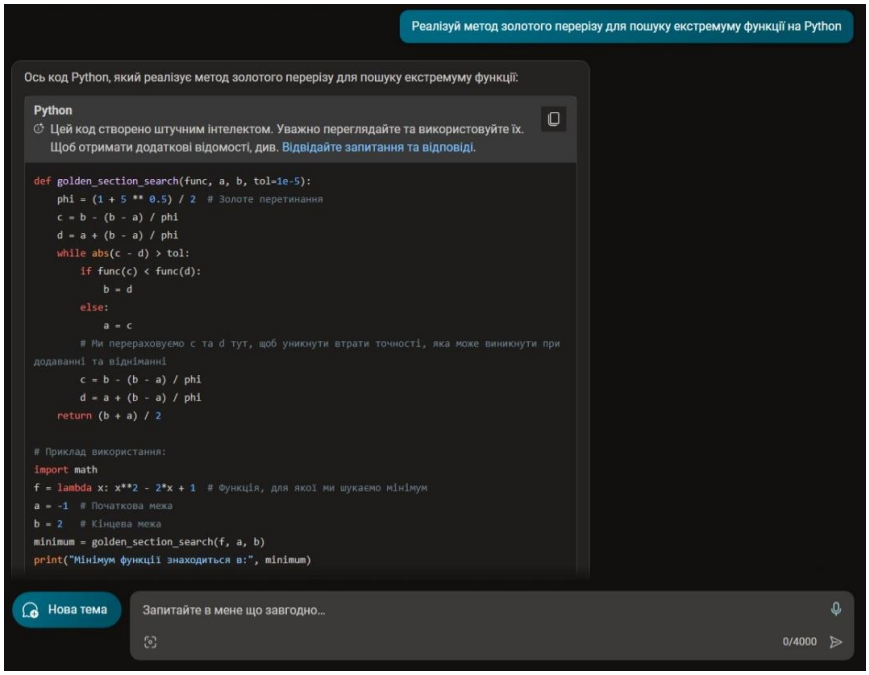


Рис. 3.15 – Результат після запиту в BingAI

Працює цей код так:

– def golden_section_search(func, a, b, tol=1e-5) – це оголошення функції golden_section_search, яка приймає аргументи:

- func - функція, для якої потрібно знайти мінімум;
- a і b – початкова та кінцева межі (відрізок), на якому проводитиметься пошук мінімуму;
- tol – допустима похибка, за замовчуванням дорівнює 1e-5.

– phi = (1 + 5 ** 0.5) / 2: Визначення константи phi, яка представляє Золотий перетин (золоте число), визначене як (1 + √5) / 2.

– Ініціалізація змінних c і d для обмеження відрізка. Значення c розташоване ближче до b, а d розташована ближче до a.

– У циклі while виконується пошук мінімуму з обмеженнями a і b. Цикл продовжується доти, доки різниця між c і d менше за задану точність tol. По суті, цей цикл здійснює ітеративний процес для зменшення відрізка, на якому знаходиться мінімум.

– У циклі обчислюється значення функції func в точках c і d.

– Порівнюються значення func(c) та func(d). Якщо func(c) менше func(d), то змінна b змінюється на d, інакше змінна a змінюється на c. Це змінює відрізок, на якому шукається мінімум, на основі порівнянь функцій.

– Виконується оновлення точок c і d на основі нового відрізку для уникнення втрати точності, яка може виникнути при багаторазовому додаванні та відніманні.

– Після завершення циклу обчислюється середнє значення точок a та b і повертається як результат пошуку.

Отримавши код від BingAI та скомпілювавши його через Google Collab отримали такі результати:

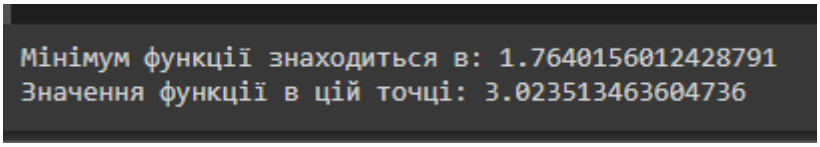


Рис 3.16 – Результати коду BingAI

На рис. 3.16 представлені результати коду, у яких ми бачимо, що він правильно знайшов мінімум функції та її значення. Тобто, можна зробити висновок, що BingAI повністю впорався з поставленою задачею з першої спроби.

Запит в Claude 1 для методу золотого перетину. Після подання універсального запиту в Claude 1 отримали такий результат, показаний на рис. 3.17.

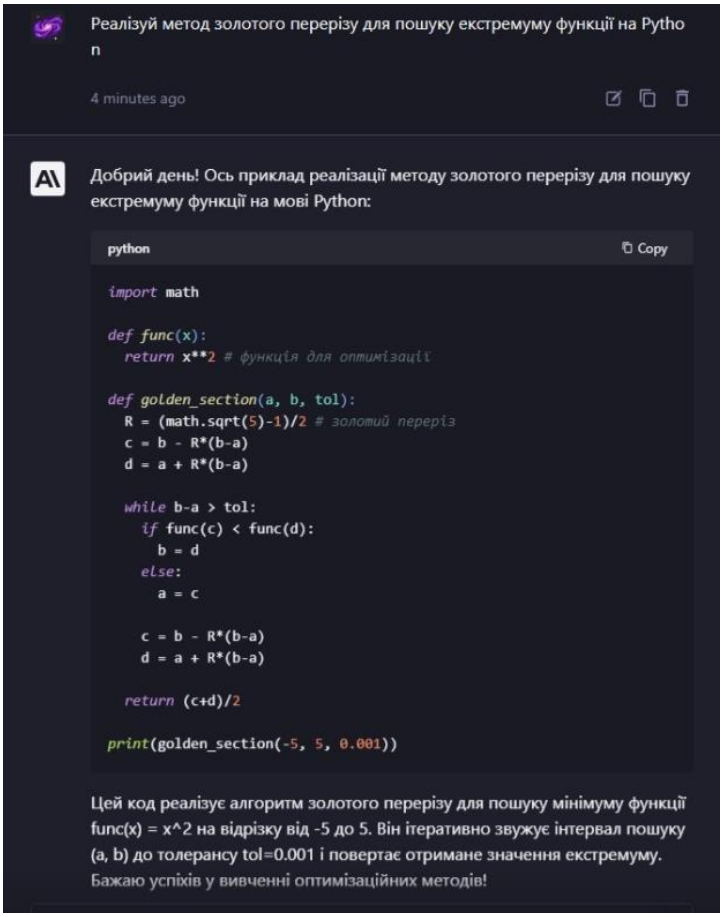


Рис. 3.17 – Результат після запиту в Claude 1

Працює цей код таким чином:

– def func(x) – ця функція приймає один аргумент x і виконує обчислення функції y залежно від значення x. У даному випадку, *y* обчислюється як exp(1/x) + log(2*x). Потім значення y виводиться на екран, і воно повертається як результат.

– def golden_section(a, b, tol) – ця функція реалізує метод золотого перетину для пошуку мінімуму функції на відрізку [a, b] з заданою точністю tol.

– R = (math.sqrt(5)-1)/2. Визначення константи R, яка представляє відношення (1 - Золотий перетин).

– Ініціалізація змінних c і d для обмеження відрізка. c розташована ближче до b, a d розташована ближче до a.

– У циклі while виконується пошук мінімуму на відрізку, доки різниця b-a більша за задану точність tol.

– На кожній ітерації викликається функція func(c) і func(d) для обчислення значень функції в точках c і d.

– Порівнюються значення func(c) та func(d). Якщо func(c) менше за func(d), то відрізок змінюється шляхом оновлення b на d. В іншому випадку, відрізок змінюється шляхом оновлення a на c.

– Точки c і d оновлюються з використанням константи R для уникнення втрати точності, яка може виникнути при багаторазовому додаванні та відніманні.

– У кінці циклу обчислюється значення функції в середній точці (c+d)/2.

– Результат вибору c і d повертається як результат пошуку мінімуму.

– Визначення тестової функції f, для якої ми шукаємо мінімум, а також визначення початкової межі a і кінцевої межі b.

– Виклик функції golden_section з параметрами a, b і tol для пошуку мінімуму функції f на заданому відрізку.

– Виведення результатів пошуку, включаючи точку minimum, де досягнуто мінімум, і значення функції в цій точці f_min.

Отримавши код від Claude 1 та скомпілювавши його через Google Collab отримали такі результати:

```
f(x) = 3.0235157972003677
f(x) = 3.023514570707146
f(x) = 3.0235157972003677
f(x) = 3.023514018190707
f(x) = 3.0235145707071456
f(x) = 3.023514018190707
f(x) = 3.0235145707071456
f(x) = 3.023513755342573
f(x) = 3.023514018190707
f(x) = 3.023513755342573
f(x) = 3.023514018190707
f(x) = 3.0235136862778518
1.764452973379218
```

Рис 3.18 – Результати коду Claude 1

На рис. 3.18 представлені результати коду, у яких бачимо, що він правильно знайшов мінімум функції та її значення, хоч і вивів їх у циклі, про що не було сказано в промті. Тобто, можна зробити висновок, що Claude 1 майже добре впорався з поставленою задачею з першої спроби.

Запит в Perplexity для методу золотого перетину. Після подання універсального запиту в Perplexity отримали такий результат – рис. 3.19.

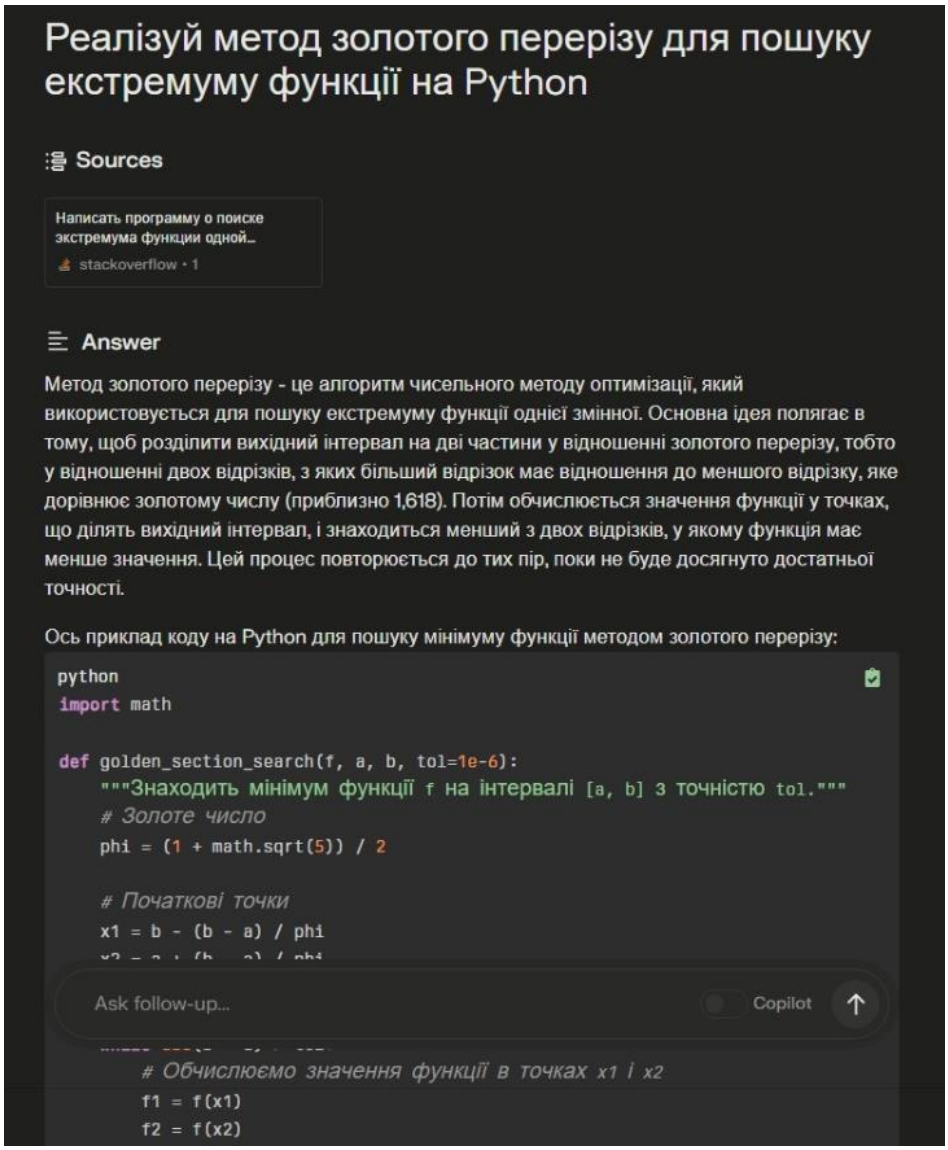


Рис. 3.19 – Результат після запиту в Perplexity

Працює цей код таким чином:

– def golden_section_search(f, a, b, tol=1e-6) – ця функція приймає такі аргументи:

- f – функція f(x), для якої потрібно знайти мінімум;
- a і b – початковий і кінцевий інтервали, на яких проводиться пошук мінімуму;
- tol – допустима похибка (за замовчуванням 1e-6).

– Визначається золоте число phi, яке використовується в методі золотого перетину. Воно обчислюється як (1 + sqrt(5)) / 2.

– Ініціалізуються початкові точки x1 і x2 на інтервалі [a, b] з використанням золотого числа. Точка x1 розташована ближче до b, і точка x2 розташована ближче до a.

– У циклі while виконується пошук мінімуму на інтервалі [a, b] доти, доки різниця між b і a не стане меншою за задану похибку tol.

– У кожній ітерації обчислюються значення функції f в точках x1 і x2 як f1 та f2.

– Порівнюються значення f1 і f2. Якщо f1 менше f2, то мінімум функції знаходиться на відрізку [a, x2]. Тому значення b оновлюється на x2, і точки x1 і x2 оновлюються з використанням золотого числа.

– В іншому випадку мінімум функції знаходиться на відрізку [x1, b]. Тому значення a оновлюється на x1, і точки x1 і x2 оновлюються також з використанням золотого числа.

– Після завершення циклу обчислюється середина відрізка [a, b], і це значення повертається як результат пошуку мінімуму.

– Визначається функція f(x), для якої ми шукаємо мінімум, у цьому випадку це функція, яка залежить від x.

– Визначаються початковий і кінцевий інтервали a і b для пошуку мінімуму.

– Викликається функція golden_section_search(f, a, b) для пошуку мінімуму функції f на вказаному інтервалі.

– Виводяться результати пошуку, включаючи точку x_min, де досягнуто мінімум, і значення функції в цій точці f(x_min).

Отримавши код від Perplexity та скомпілювавши його через Google Collab отримали такі результати:

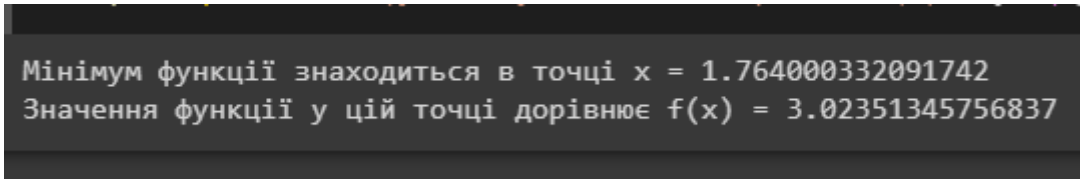


Рис 3.20 – Результати коду Perplexity

На рис. 3.20 представлені результати коду, у яких ми бачимо, що він правильно знайшов мінімум функції та її значення. Тобто, можна зробити висновок, що Perplexity повністю впорався з поставленою задачею з першої спроби.

Висновки за методом золотого перетину. З отриманих результатів можна зробити висновок, що усі 4 моделі штучного інтелекту успішно справилися із завданням написання коду для реалізації методу золотого перетину для пошуку мінімуму функції. Код, згенерований цими моделями, працює коректно і знаходить правильне рішення. Це свідчить про здатність AI генерувати працездатний код за словесним описом для цього методу. Реалізації алгоритму в усіх 4 моделях досить схожі за структурою та принципом роботи. Вони використовують золоте число для ініціалізації точок пошуку, ітераційно звужують інтервал пошуку на основі порівняння значень функції, і повертають середину останнього інтервалу як результат.

Загалом, найбільш "чистий" код з точки зору відповідності завданню був згенерований ChatGPT, BingAI та Perplexity. Claude знову трохи відхилився від постановки, але в цілому теж виконав завдання.

3.1.3 Фрактал Коха

Поняття "фрактал" виникло у кінці 1970-х років і стало ключовим для математиків та комп'ютерних художників. Використання фракталів в комп'ютерній графіці отримало широке розповсюдження завдяки їхній математичній компактності, що дозволяє їх легко відтворювати на екрані

комп'ютера. Термін "фрактал" віддзеркалює його походження, він походить від латинського слова "fractus," що перекладається як "складений з частин". Цей термін був запропонований французько-американським математиком Бенуа Мандельбротом у 1975 році для опису самоподібних структур.

У контексті комп'ютерної графіки, фрактал означає геометричну фігуру, що складається з частин, подібних за формою до всієї фігури, де кожен фрагмент повторюється при зменшенні масштабу. Основною рисою фрактала є його самоподібність відносно масштабу. У будівництві фракталів вирізняють геометричні та алгебраїчні підходи. Геометричні фрактали формуються за допомогою послідовних геометричних операцій, які виконуються над деякою геометричною фігурою. У більшості випадків ці алгоритми мають рекурсивну структуру, що віддзеркалює самоподібність фрактала. Найбільш відомим прикладом геометричного фрактала є крива Коха, побудована Хельге фон Кохом у 1904 році.

Для ефективного порівняння результатів у ході численних запитів, був розроблений універсальний промт: «Побудуй тріадну криву Коха на Python». На виході маємо отримати сніжинку Коха у будь-якому вигляді.

Запит в ChatGPT для фрактала Коха. Після подання універсального запиту в ChatGPT отримали такий результат – рис. 3.21.

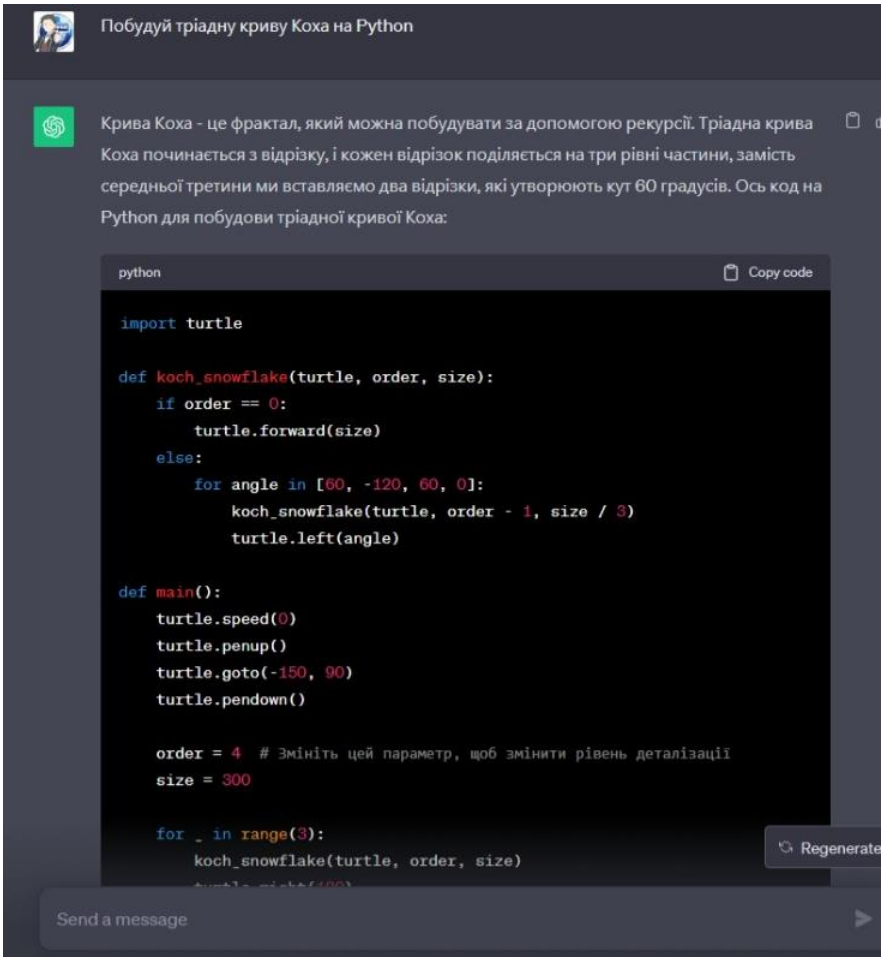


Рис. 3.21 – Результат після запиту в ChatGPT

Працює цей код так:

– import turtle – цей рядок імпортує модуль turtle, який дозволяє створювати графічні малюнки і анімацію.

– def koch_snowflake(turtle, order, size) – ця функція будує сніжинку Коха з вказаним рівнем деталізації (order) і розміром (size). Вона використовує рекурсивний підхід для побудови сніжинки Коха:

- якщо order дорівнює 0, то виконується переміщення черепашки вперед на відстань size;
- в іншому випадку, для кожного кута зі списку [60, -120, 60, 0], функція викликає саму себе зменшуючи order на одиницю і зменшуючи size втрете. Потім черепашка повертається на заданий кут.

– def main(): Ця функція визначає основний код програми.

– Встановлюється максимальна швидкість черепашки (turtle.speed(0)).

– Замість малювання ліній черепашкою використовується підйом пера черепашки (turtle.penup()), і вона пересувається до позиції, де малюватиметься сніжинка.

– Повертається підйом пера (turtle.pendown()).

– Визначаються параметри для побудови сніжинки:

  - order – рівень деталізації сніжинки Коха (за замовчуванням 4);
  - size – розмір сніжинки.

– За допомогою циклу малюється три різних сніжинки Коха, кожна з яких відзначена праворуч на 120 градусів.

– Черепашка приховується (turtle.hideturtle()), і вікно закривається користувачем при кліку на нього.

– if __name__ == "__main__": main() – ця конструкція дозволяє виконувати функцію main тільки тоді, коли цей скрипт виконується напряму, а не імпортується в інший скрипт.

Отримавши код від ChatGPT та скомпілювавши його через PyCharm отримали такі результати:

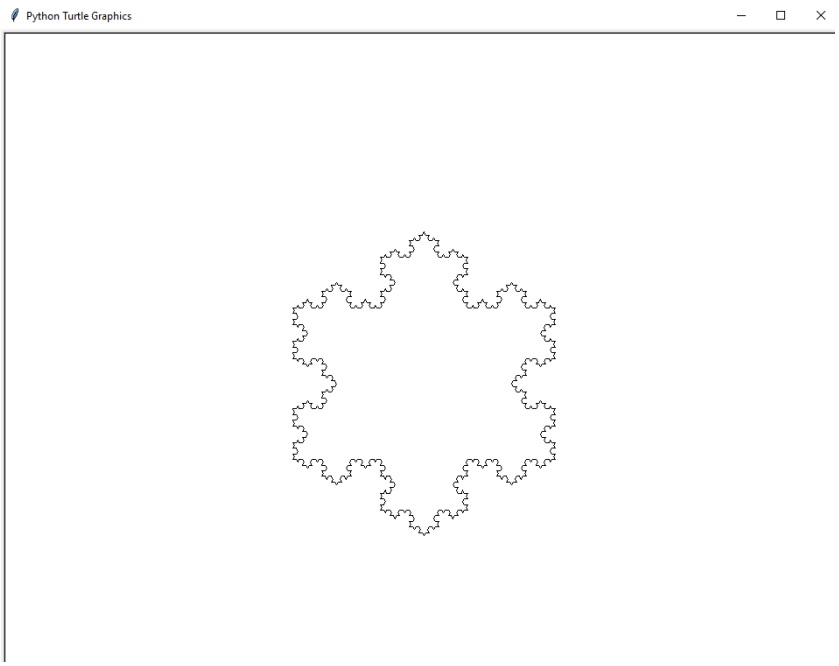


Рис 3.22 – Результати коду ChatGPT

Як бачимо з рис. 3.22 код, який надав ChatGPT повністю впорався з поставленою задачею з першої спроби, тобто він виконав повністю весь цикл та замкнув тріадну криву Коха отримавши ідеальну сніжинку.

Запит в BingAI для фрактала Коха. Після подання універсального запиту в BingAI отримали такий результат – рис. 3.23.

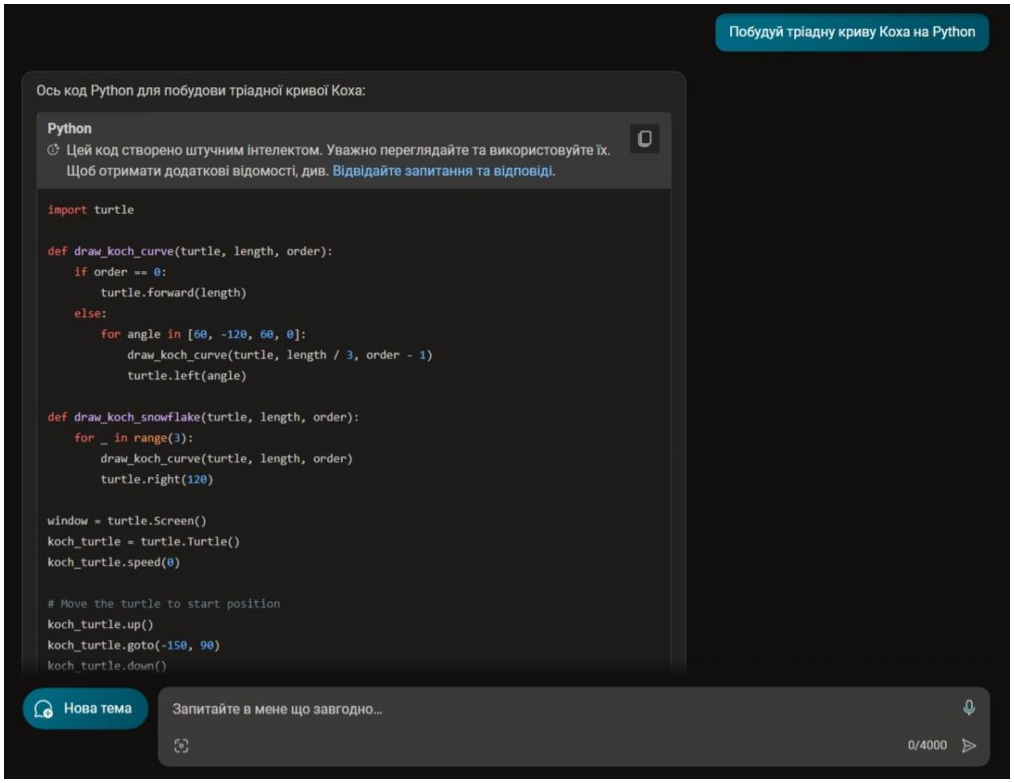


Рис. 3.23 – Результат після запиту в BingAI

Працює цей код таким чином:

– import turtle – цей рядок імпортує модуль turtle, який надає можливість рухати черепашку і рисувати з її допомогою на графічному вікні.

– def draw_koch_curve(turtle, length, order) – ця функція призначена для малювання кривої Коха певної довжини та на певному рівні деталізації:

- якщо order дорівнює 0, то черепашка рухається вперед на вказану довжину length, і це є базовим випадком рекурсії;
- якщо order більше 0, то черепашка рухається через цикл з чотирма ітераціями з кутами [60, -120, 60, 0]. Для кожного кута викликається

draw_koch_curve рекурсивно зменшуючи рівень order на одиницю і зменшуючи довжину на третину.

– def draw_koch_snowflake(turtle, length, order) – ця функція малює сніжинку Коха, яка складається з трьох кривих Коха після однієї. Викликається draw_koch_curve для малювання одного відрізку кривої Коха, потім черепашка повертається на 120 градусів вправо і цей процес повторюється три рази (для кожного з трьох відрізків, які складають сніжинку).

– window = turtle.Screen() – створення графічного вікна через об'єкт turtle.Screen().

– koch_turtle = turtle.Turtle() – створення об'єкта turtle.Turtle(), який представляє черепашку для малювання.

– koch_turtle.speed(0) – встановлення максимальної швидкості для черепашки (швидкість 0 означає максимальну швидкість, без затримок).

– Переміщення черепашки до початкової позиції:
  - відрив пера черепашки (koch_turtle.up());
  - переміщення до позиції (-150, 90) на графічному вікні (koch_turtle.goto(-150, 90));
  - активація пера черепашки для малювання (koch_turtle.down()).

– Виклик функції draw_koch_snowflake для малювання сніжинки Коха. Параметри функції:
  - koch_turtle – об'єкт черепашки, який використовується для малювання;
  - length – довжина початкового відрізку сніжинки;
  - order – рівень деталізації сніжинки Коха (за замовчуванням 3).

– Після завершення малювання черепашка ховається (koch_turtle.hideturtle()), і вікно графічного інтерфейсу черепашки відображається і очікує дій користувача (window.mainloop()).

Отримавши код від BingAI та скомпілювавши його через PyCharm отримали такі результати:

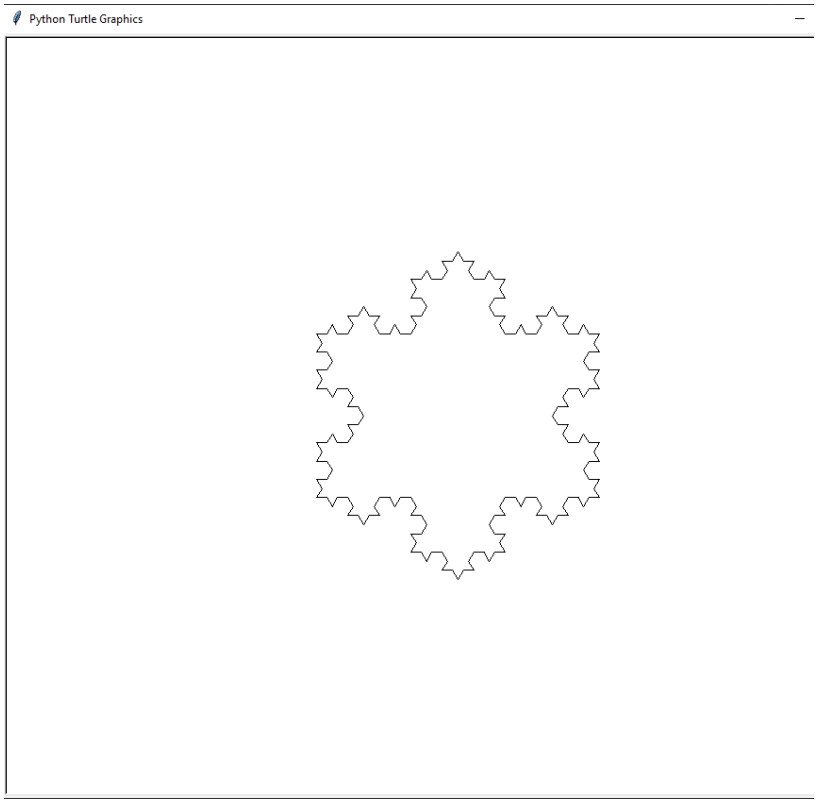


Рис 3.24 – Результати коду BingAI

Як бачимо з рис. 3.24 код, який надав BingAI повністю впорався з поставленою задачею з першої спроби, тобто він виконав повністю весь цикл та замкнув тріадну криву Коха отримавши ідеальну сніжинку.

Запит в Claude 1 для фрактала Коха. Після подання універсального запиту в Claude 1 отримали такий результат – рис. 3.25.

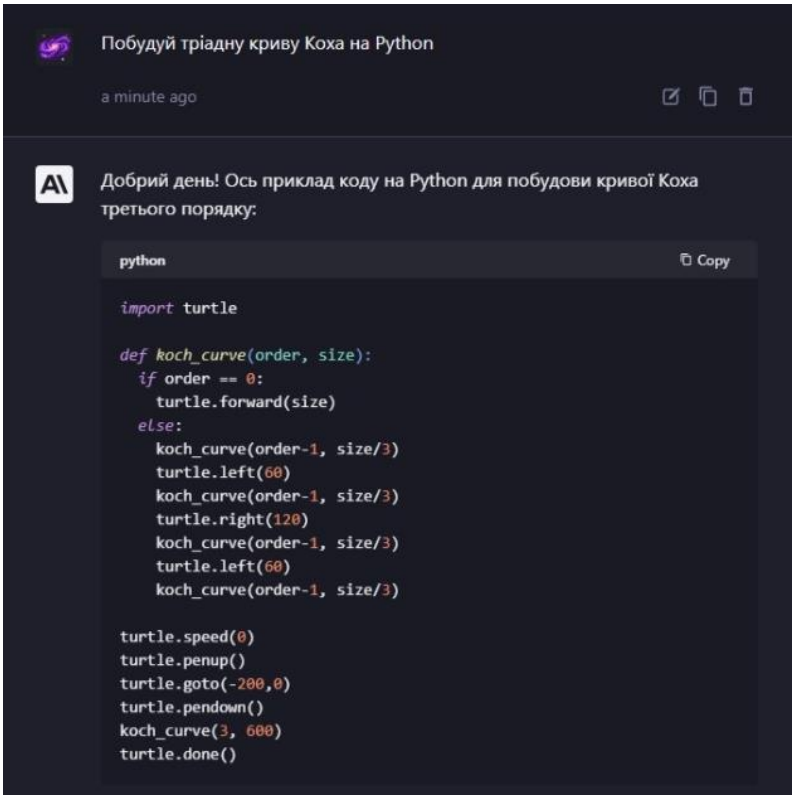


Рис. 3.25 – Результат після запиту в Claude 1

Працює цей код таким чином:

– import turtle – цей рядок імпортує модуль turtle, який дозволяє створювати графічні малюнки за допомогою черепашки.

– def koch_curve(order, size) – ця функція призначена для малювання кривої Коха певного рівня order та зазначеної довжини size:

- якщо order дорівнює 0, то черепашка рухається вперед на відстань size. Це базовий випадок рекурсії;
- у випадку, коли order більше 0, функція рекурсивно викликає себе чотири рази, зменшуючи order на одиницю та зменшуючи size втретє;
- після кожного виклику функції черепашка робить повороти на кути, які потрібні для побудови кривої Коха. Специфічно, це лівий поворот на 60 градусів, потім правий поворот на 120 градусів, лівий поворот на 60 градусів і знову лівий поворот на 60 градусів.

– turtle.speed(0) - встановлюється максимальна швидкість черепашки (швидкість 0 означає максимальну швидкість, без затримок).

– turtle.penup(): черепашка піднімає перо, щоб перемістити її до нової позиції без малювання.

– turtle.goto(-200, 0): черепашка переміщається до початкової позиції (-200, 0).

– turtle.pendown(): перо черепашки опускається, щоб вона могла малювати на екрані.

– koch_curve(3, 600) – виклик функції koch_curve з параметрами order=3 та size=600. Це означає, що буде намальована крива Коха третього рівня деталізації з відрізком довжиною 600 одиниць.

– turtle.done(): черепашка завершує роботу, і вікно графічного інтерфейсу черепашки залишається відкритим для подиву користувача.

Отримавши код від Claude 1 та скомпілювавши його через PyCharm отримали такі результати:

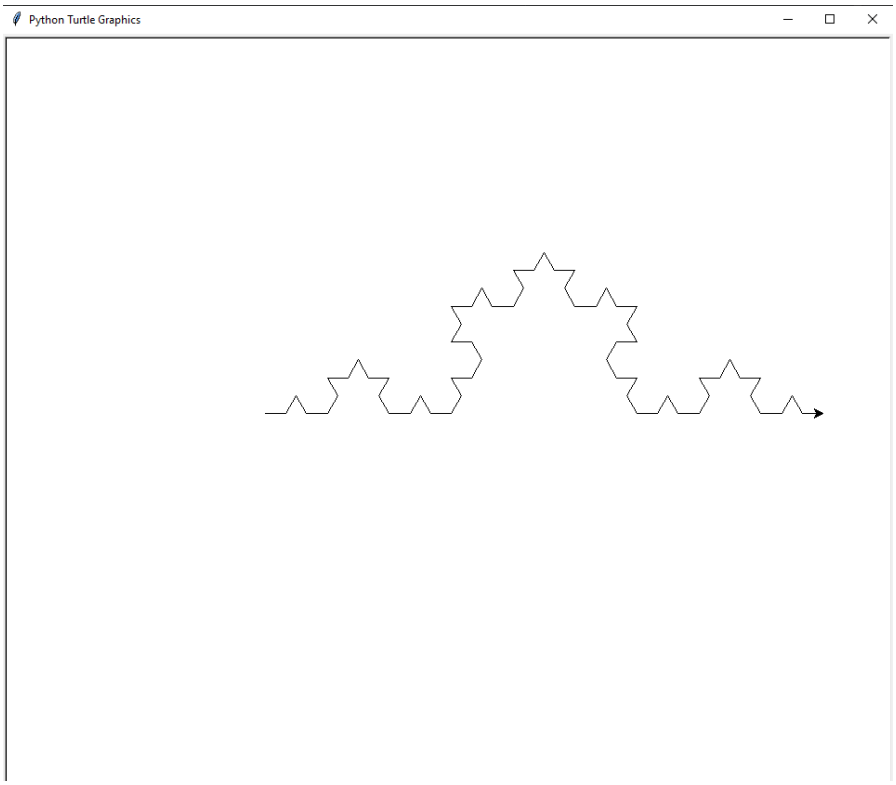


Рис 3.26 – Результати коду Claude 1

Як бачимо з рис. 3.26 Claude 1 трохи не впорався з поставленою задачею, скільки б спроб не було використано, тому що він не дотягує кількість ітерацій, щоб замкнути тріадну криву Коха.

Запит в Perplexity для фрактала Коха. Після подання універсального запиту в Perplexity отримали такий результат – рис. 3.27.

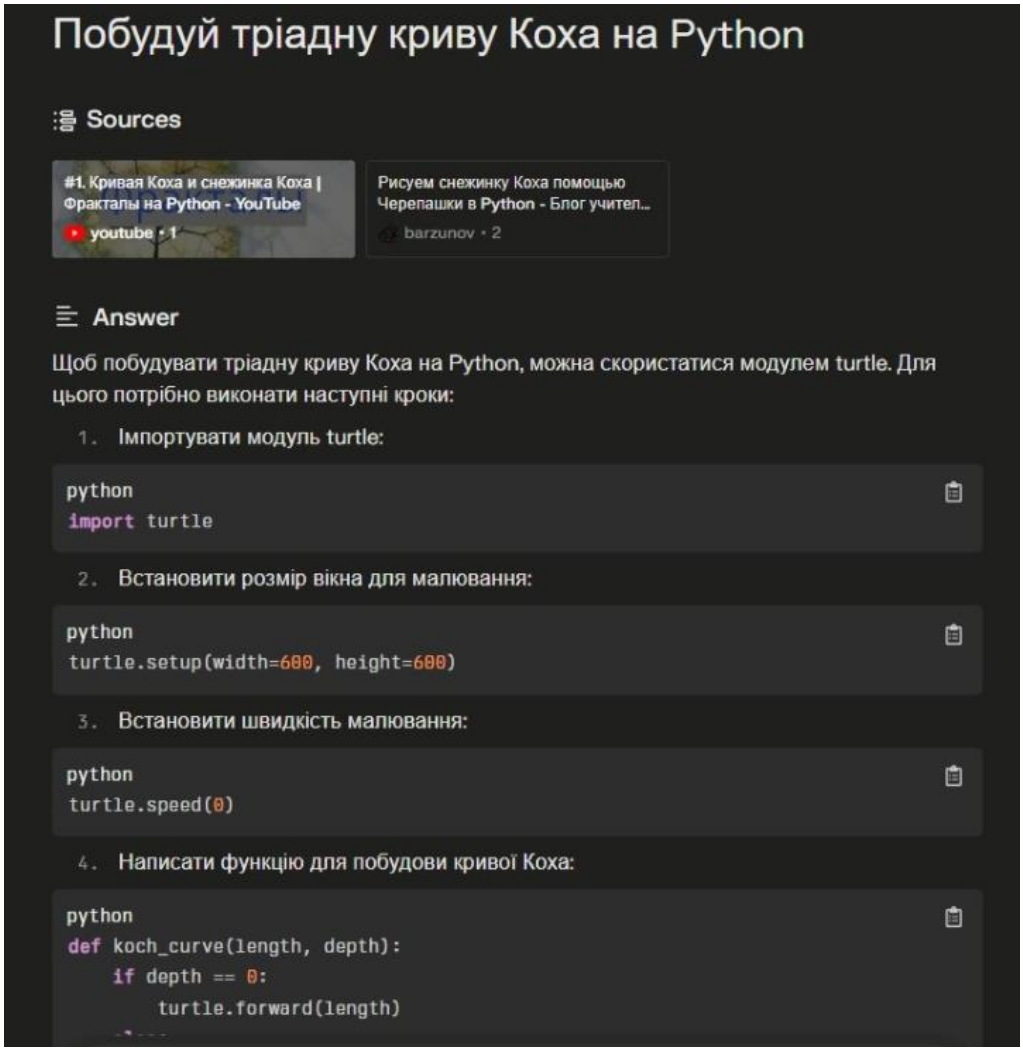


Рис. 3.27 – Результат після запиту в Perplexity

Працює цей код таким чином:

1) import turtle – цей рядок імпортує модуль turtle, який дозволяє створювати графічні малюнки та анімацію за допомогою черепашки.

2) turtle.setup(width=600, height=600) – цей рядок налаштовує вікно для малюнка на графічному екрані. Розмір вікна встановлюється на 600 пікселів у ширину і 600 пікселів у висоту.

3) turtle.speed(0) – встановлюється максимальна швидкість для черепашки. Значення 0 означає найвищу швидкість без затримок.

4) def koch_curve(length, depth) – ця функція призначена для малювання кривої Коха певної довжини length та рівня деталізації (глибини) depth:

- якщо depth дорівнює 0, то черепашка рухається вперед на відстань length. Це базовий випадок рекурсії;
- у випадку, коли depth більше 0, функція рекурсивно викликає себе п'ять разів (по одному разу для кожної частини кривої Коха);
- спочатку черепашка рухається на відстань length / 3, потім викликається ліва поворот на 60 градусів і рекурсивний виклик функції зі зменшеною довжиною і глибиною;
- після цього черепашка повертається на правий поворот на 120 градусів та рекурсивний виклик зі зменшеною довжиною і глибиною;
- далі черепашка рухається на відстань length / 3 та знову викликається ліва поворот на 60 градусів та рекурсивний виклик;
- на останок черепашка робить лівий поворот на 60 градусів та рекурсивний виклик.

5) koch_curve(200, 3) – виклик функції koch_curve для малювання кривої Коха з довжиною 200 та рівнем деталізації 3.

6) turtle.done() – завершення роботи черепашки та вікно графічного інтерфейсу черепашки очікує подальших дій користувача.

Отримавши код від Perplexity та скомпілювавши його через PyCharm отримали такі результати:

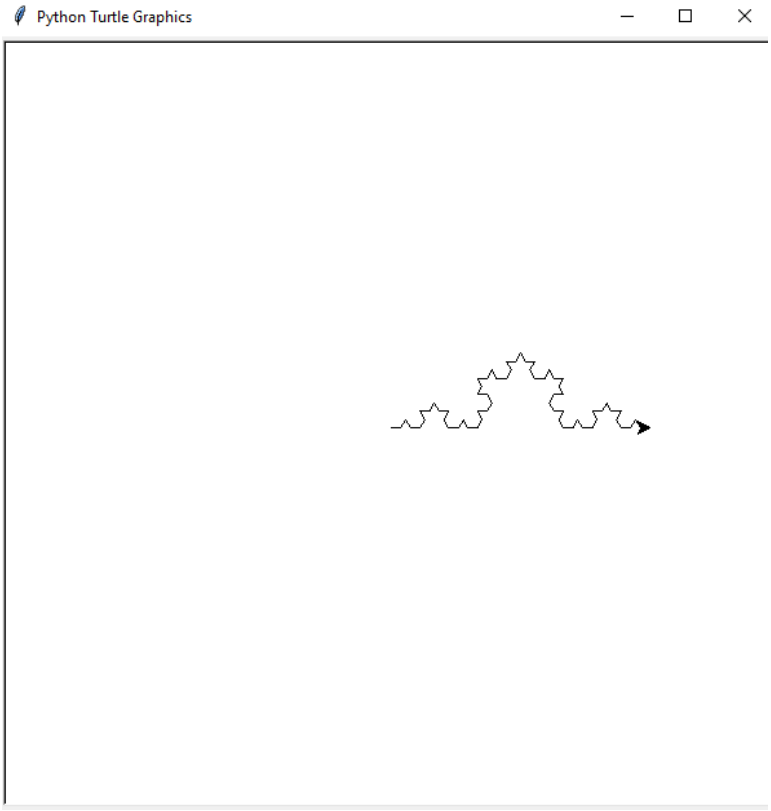


Рис 3.28 – Результати коду Perplexity

Як бачимо з рис. 3.28 Perplexity трохи не впорався з поставленою задачею, скільки б спроб не було використано, тому що він не дотягує кількість ітерацій, щоб замкнути тріадну криву Коха.

Висновки за фракталом Коха. З отриманих результатів можна зробити висновок, що ChatGPT і Bing AI успішно справилися з завданням побудови фрактала Коха. Код, згенерований ними, повністю реалізував алгоритм та вивів очікуваний результат у вигляді замкнутої тріадної кривої. На відміну від моделей Claude 1 та Perplexity, які не змогли повністю реалізувати алгоритм - в їх коді не вистачає ітерацій рекурсії, щоб замкнути тріадну криву Коха. ChatGPT і Bing AI продемонстрували краще розуміння завдання та здатність генерувати працюючий код складних алгоритмів, у порівнянні з Claude 1 та Perplexity. Їх код є добре структурованим, має чіткі коментарі та легко зрозумілий для подальшого використання. В той час, коли Claude 1 та Perplexity потребують додаткового втручання та коригувань від користувача, щоб отримати повноцінне рішення.

3.2 Тестування та вибір ефективнішої нейромережевої моделі

Наступним кроком було проведення ретельного тестування різних варіантів коду, з метою визначення найбільш ефективного відповідно до ряду критеріїв. Це тестування було важливим етапом, оскільки воно дозволило нам отримати об'єктивну інформацію про продуктивність кожного варіанту коду.

Методологія оцінки ефективності роботи нейромережевих моделей включає кілька ключових етапів:

– вимірювання часу виконання коду за допомогою модулів Python timeit та time. Це дозволяє порівняти швидкодію різних моделей;

– профілювання коду за допомогою модулів cProfile та LineProfiler для аналізу в режимі реального часу та виявлення функцій і рядків коду, які займають найбільше ресурсів;

– моніторинг використання пам'яті за допомогою модуля memory_profiler для виявлення можливих витоків пам'яті та проблем з використанням ресурсів;

– аналіз якості отриманих результатів, наприклад повноти побудови графічних зображень для фракталів;

– порівняння показників різних моделей за переліченими критеріями та вибір найкращого варіанту моделі для конкретного завдання.

Отже, оцінка проводиться комплексно з урахуванням швидкодії, ефективності використання ресурсів та якості кінцевих результатів для вибору найкращої моделі.

Перший з основних критеріїв оцінки пов'язаний із вимірюванням часу виконання коду. Для цього використовувались модулі Python, такі як time і timeit. Вимірювання часу надало нам можливість порівняти різні варіанти коду за швидкістю їх виконання.

Другий критерій – профілювання коду. Використовувались модулі cProfile і LineProfiler, щоб отримати інформацію про те, як код виконується в режимі реального часу та визначити, які функції або рядки коду займають найбільше часу в процесі виконання. Це було корисним для ідентифікації можливих місць оптимізації.

Крім того, ще одним важливим аспектом було вивчення використання пам'яті під час виконання коду. Для цього використовувався модуль memory_profiler. Моніторинг пам'яті допоміг нам виявити ефективність роботи кожного варіанту коду та знайти можливі утечки пам'яті або інші проблеми, пов'язані із використанням ресурсів.

Всі ці критерії були враховані під час аналізу результатів тестування. Це дало можливість зробити об'єктивний висновок щодо того варіанту коду, який є найбільш ефективним з точки зору виконання, продуктивності та використання пам'яті.

Результати тестування алгоритму методу Гауса (табл. 3.1) показали, що серед розглянутих чат-ботів ChatGPT має найкращу продуктивність з точки зору вимірювання часу виконання коду. Він виявився найшвидшим, витрачаючи всього 5.6 секунди для завершення обробки тексту. У порівнянні з іншими ботами, такими як Bing AI, Claude 1 і Perplexity, ChatGPT показав значно кращу часову ефективність. Проте, коли мова йде про оптимізацію використання пам'яті, Bing AI виявився менш вимогливим, використовуючи всього 33.1 MiB, що є найменшим показником серед усіх чат-ботів. З іншого боку, ChatGPT і Perplexity використовують приблизно однаковий обсяг пам'яті, а Claude 1, хоча і швидший за часом виконання, вимагає значно більше пам'яті.

Таблиця 3.1

Результати тестування за методом Гауса

Чат-бот	timeit	time	cProfile	LineProfiler	memory_profiler
ChatGPT	5.6s	11.7s	12 function calls	4.5s	20.7 MiB
Bing AI	8.1s	11.3s	32 function calls	3.7s	33.1 MiB
Claude 1	15.3s	0.08s	12 function calls	4.12s	32.7 MiB

Продовження табл. 3.1

Perplexity	14s	16.3s	10 function calls	4.49s	32.6 MiB

Таблиця 3.2

Результати тестування за методом золотого перетину

Чат-бот	timeit	time	cProfile	LineProfiler	memory_profiler
ChatGPT	17.9s	0.1s	166 function calls	9.03s	20.1 MiB
Bing AI	16.4s	0.1s	148 function calls	2.84s	20.1 MiB
Claude 1	10.3s	0.1s	83 function calls	5.8s	20.0 MiB
Perplexity	21.2s	0.1s	202 function calls	3.99s	20.0 MiB

Результати тестування за методом золотого перетину (табл. 3.2) показали, що серед розглянутих чат-ботів Claude 1 має найкращу продуктивність з точки зору швидкості виконання коду. Він виявився найшвидшим, витрачаючи лише 10.3 секунди на завершення завдання. У порівнянні з іншими ботами, такими як ChatGPT, Bing AI і Perplexity, Claude 1 продемонстрував найкращу часову ефективність. Однак, коли мова йде про оптимізацію використання пам'яті, усі чат-боти показали однаковий результат – 20 МіБ. Натомість, за кількістю функціональних викликів Claude 1 виявився найбільш економним – лише 83 виклики. Отже, загалом за швидкістю роботи та ефективним використанням ресурсів Claude 1 продемонстрував найкращі результати в тестуванні.

Таблиця 3.3

Результати тестування фрактала Коха

Чат-бот	timeit	cProfile	memory_profiler	Графічний результат
ChatGPT	35.4s	1280 function calls	374.4 MiB	Повністю побудовано
Bing AI	9.1s	3278 function calls	312.8 MiB	Повністю побудовано
Claude 1	3.7s	814 function calls	216.8 MiB	Не повністю побудовано
Perplexity	5.1s	705 function calls	152.0 MiB	Не повністю побудовано

Результати тестування фракталу Коха (табл. 3.3) показали, що з точки зору побудови графічного зображення фрактала Коха найкраще впоралися ChatGPT та Bing AI, які повністю побудували фрактал. Хоча за часом виконання коду Claude 1 продемонстрував найкращий результат – лише 3.7 секунди, він не зміг повністю побудувати зображення. Так само неповну побудову показав і Perplexity. Щодо оптимізації використання пам'яті, то Perplexity виявився найбільш економним – 152 МіБ. Натомість, ChatGPT витратив найбільше пам'яті – 374.4 МіБ. Отже, враховуючи важливість повної та коректної побудови графічного зображення для даного тесту, ChatGPT та Bing AI продемонстрували найкращі результати. Але найкращий з них буде Bing AI, хоч він і викликав більше функцій в cProfile, але по часу виконання та використання пам'яті – він краще впорався, ніж ChatGPT.

За результатами тестування можна зробити висновок, що для метода Гауса найкраще показав себе код від ChatGPT, для метода золотого перетину – Claude 1 та для фрактала Коха – BingAI. Отже, можна сказати, що кожен з них по своєму, але впорався з задачею створення коду на Python для математичного підрахунку. Що не можна сказати про графічне створення, адже створити павутинну чи

біфуркаційну діаграму ніхто з них не зміг. З фракталом Коха зміг справитися лише ChatGPT та BingAI. Claude 1 та Perplexity не змогли довести до кінця сніжинку.

3.3 Рекомендації щодо написання запитів до чат-ботів

Термін "запит" (або "prompts") у контексті чат-ботів вказує на фрагмент тексту, який подається моделі в якості вхідних даних і надає вказівки для створення відповіді. Запит може бути сформульований у вигляді запитання, твердження або набору інструкцій. Запити використовуються для створення контексту для вирішення завдань або спілкування взагалі. Існує декілька універсальних рекомендацій для написання ефективного запиту:

1) писати запити англійською мовою. Хоча ChatGPT та інші чат-боти розуміють українську мову і можуть надавати відповіді на запити, але важливо пам'ятати, що модель була натренована саме на англійських текстах. Використання англійської мови допомагає отримати більш точні та швидкі відповіді від нейромережі. Коли надходить український запит, моделі потрібен час на переклад, пошук і знову переклад відповіді, що може вплинути на час реакції моделі і призводити до затримок у відповідях. Тому використання англійської мови покращує якість та швидкість спілкування з нейромережею;

2) регулярно оновлювати контекст. ChatGPT та інші чат-боти мають обмежену пам'ять у процесі діалогу. Іноді в процесі розмови мережа може забути інформацію, яка була важлива кілька кроків тому. Щоб уникнути цієї ситуації, потрібно регулярно оновлювати контекст розмови;

3) бути якнайбільш конкретними. Для досягнення найкращих результатів у комунікації з системою, важливо чітко визначити завдання або мету запиту. Потрібно вміти керувати процесом розмови, надаючи чіткий напрямок. Додавати якомога більше інформації до запиту або надавати необхідний контекст, щоб чат-бот міг краще зрозуміти завдання. Використовувати конкретні приклади, уникаючи загальних або абстрактних термінів. Інакше модель може мати складнощі з їх інтерпретацією. Уникати використання надмірно складної мови або технічного жаргону. Спрощена мова полегшить взаєморозуміння.

Спрямовувати одне завдання на один запит – це полегшить розв'язання ваших питань. Короткі та лаконічні запити зазвичай краще зрозуміються ШІ.

Для того, аби правильно створити промт потрібно:

1. Визначити роль. Зазначити, ким ви бажаєте, щоб чат-бот представляв себе для даного завдання. Це допоможе чат-боту адаптуватися до необхідного контексту. Наприклад: я хочу, щоб ти діяв як професійний програміст на мові програмування Python.

2. Надати вхідні дані. Перерахувати всі можливі відомості, необхідні для виконання цього завдання. Ця інформація може включати різні дані, такі як опис методу, обсяг коду, присутність коментарів, функції тощо. Наприклад: Опис методу Гауса, необмежений обсяг коду, з двома функціями, додати коментарі.

3. Сформулювати запит. Ретельно визначити те, що плануємо отримати від моделі в результаті. Наприклад: Ти розробиш для мене програму для вирішення методу Гауса з трьома невідомими.

4. Уточнити умови. Конкретизувати можливості чи обмеження бота під час його роботи. Наприклад, можна дозволити йому створювати різні варіанти коду та визначати їх ефективність.

5. Встановити обмеження. На цьому етапі вказати, що чат-бот не повинен використовувати під час виконання завдання. Наприклад, обмежуємо використання технічного жаргону чи інших конкретних елементів.

6. Надати інформацію. Надати всі дані, які вважаємо важливими або які обіцяли надати боту. Наприклад, надаємо опис методу чи надаємо допомогу у вирішенні завдання, якщо бот не справляється.

7. Надати зразок. Якщо завдання допускає, демонструємо приклад кінцевого результату. Наприклад, ось приклад коду, який я очікую отримати в результаті.

Якщо дотримуватися цих правил, то можна отримати більш ефективний запит, який з великою часткою ймовірності дасть потрібну нам відповідь. Враховуючи вказані етапи та рекомендації, покращується спілкування з ChatGPT

та іншими чат-ботами і забезпечується більш точний та продуктивний результат від цих інтелектуальних систем.

Висновок за розділом 3

У цьому розділі було проведено дослідження здатності нейромережевих моделей до генерації коду для розв'язання задач математичного аналізу тексту. Для дослідження були обрані такі алгоритми: метод Гауса, павутинні і біфуркаційні діаграми та побудова фрактала Коха. Були сформовані уніфіковані запити до 4 моделей: ChatGPT, BingAI, Claude та Perplexity. Результати показали, що жодна з моделей не змогла успішно побудувати павутинні та біфуркаційні діаграми. Тому цей алгоритми був замінений на метод золотого перетину. За методом Гауса найкраще впорався ChatGPT, він має найкращу продуктивність з точки зору вимірювання часу виконання коду та виявився найшвидшим, витрачаючи всього 5.6 секунди для завершення обробки тексту. У порівнянні з іншими ботами, такими як Bing AI, Claude 1 і Perplexity, ChatGPT показав значно кращу часову ефективність. За методом золотого перетину найкраще впорався Claude 1 – він має найкращу продуктивність з точки зору швидкості виконання коду та виявився найшвидшим, витрачаючи лише 10.3 секунди на завершення завдання і найбільш економним – лише 83 виклики. За побудовою фрактала Коха найкращими виявились ChatGPT та BingAI. Лише вони змогли повністю замкнути тріадну криву.

Загалом, моделі продемонстрували здатність до генерації працездатного коду за текстовим описом. Але вони мають обмеження у побудові складних візуальних структур в силу недотренованості моделей.

ВИСНОВКИ

У результаті аналізу різних моделей нейронних мереж, які використовуються для розв'язання завдань обробки природньої мови, можна зробити висновок, що нейромережа є математичною моделлю, яка абстрагується від біологічних нейронних мереж і дозволяє вирішувати різноманітні задачі, такі як класифікація, регресія, прогнозування та генерація. Це досягається завдяки взаємодії штучних нейронів, які організовані у вигляді графових структур та обмінюються інформацією через ваги зв'язків. Архітектура нейромережі включає в себе три основних типи шарів: вхідний, прихований та вихідний. Кожен з цих шарів містить групи штучних нейронів, які спільно виконують обробку інформації на різних стадіях. Кожен окремий штучний нейрон представляє собою вузол нейронної мережі, який складається з набору вхідних сигналів та ваг зв'язків, і використовує функцію перетворення та функцію активації.

У світі сучасних технологій існує безліч різних моделей для обробки природної мови (NLP), і вони активно використовуються для аналізу та розуміння мови, якою спілкуються люди. Деякі з найвідоміших моделей включають в себе мовну модель "трансформер", яка зараз використовується в продуктах, таких як ChatGPT, BingAI, Claude 1 та Perplexity, а також інші популярні моделі, такі як рекурентні нейронні мережі мовних моделей (RNNLM), Word2vec, GloVe (Global Vectors) та fastText. Усі ці розробки і дослідження засвідчують постійний розвиток і використання нейромереж у багатьох галузях. Вони виявляються ефективними для обробки текстової інформації, машинного навчання та аналізу даних. Моделі, які базуються на нейронах та тренуються на великих обсягах даних, розширюють можливості застосування штучного інтелекту та сприяють розвитку нових технологій. Нейромережі і надалі залишаються об'єктом інтенсивних досліджень і розвитку, і їхні можливості ще далеко не вичерпані. Зокрема, роки 2022 і 2023 відзначилися значущими подіями в індустрії штучного інтелекту та інтелектуальних

асистентів. Запуск ChatGPT від компанії OpenAI виявився переломним моментом, який відкрив багато нових можливостей у різних галузях, включаючи медицину, освіту, бізнес тощо. Цей інноваційний продукт вирізняється своєю інтелектуальністю та потенціалом покращити спілкування між людьми та штучним інтелектом. Після успішного випуску ChatGPT, Microsoft вивела на ринок свій чат-бот, BingAI і інтегрувала його в браузер Edge та операційну систему Windows 11. Цей крок свідчить про стрімкий розвиток галузі та зростаючий інтерес до інтелектуальних асистентів. Компанія Anthropic зробила свій внесок, випустивши модель для генерації тексту, Claude Instant та чат-бота Claude. А чат-бот Perplexity, який з'явився на ринку, розширює вибір інтелектуальних інтерфейсів та систем штучного інтелекту. Всі ці події свідчать про важливий тренд в індустрії штучного інтелекту - зростаючий інтерес до розробки різних інтелектуальних асистентів та текстових генераторів. Ринок інтелектуальних асистентів активно розвивається і відкриває нові можливості для взаємодії з технологією та інформацією, що відкриває нові перспективи для інновацій і поліпшень в різних сферах життя.

У роботі проведено дослідження щодо здатності моделей штучного інтелекту генерувати код для розв'язання математичних завдань, які описані у текстовій формі. Для дослідження ми обрали такі алгоритми, як метод Гауса, побудову павутинних і біфуркаційних діаграм та фрактала Коха. Моделі, зокрема ChatGPT, BingAI, Claude та Perplexity, були запитані на рішення цих завдань. Результати показали, що жодна з моделей не була здатна успішно побудувати павутинні та біфуркаційні діаграми, і тому ці алгоритми були замінені на метод золотого перетину. За методом Гауса найкраще впорався ChatGPT, він має найкращу продуктивність з точки зору вимірювання часу виконання коду та виявився найшвидшим, витрачаючи всього 5.6 секунди для завершення обробки тексту. У порівнянні з іншими ботами, такими як Bing AI, Claude 1 і Perplexity, ChatGPT показав значно кращу часову ефективність. За методом золотого перетину найкраще впорався Claude 1 - він має найкращу продуктивність з точки зору швидкості виконання коду та виявився

найшвидшим, витрачаючи лише 10.3 секунди на завершення завдання і найбільш економним – лише 83 виклики. За побудовою фрактала Коха найкращими виявились ChatGPT та BingAI. Лише вони змогли повністю замкнути тріадну криву. Загалом, моделі продемонстрували здатність генерувати життєздатний код на основі текстового опису, але вони мають обмеження щодо побудови складних візуальних структур через недостатнє навчання моделей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Рішення системи методом Гауса. OpenAI ChatGPT. URL: https://chat.openai.com/share/488ba6df-9c55-42e6-8179-f8a3a03a12d0. (Дата звернення: 03.10.2023).

2. Bing AI // MicrosoftBing. URL: https://sl.bing.net/kTOsyI4gNQO. (Дата звернення: 03.10.2023).

3. Forefront AI Chat Application. URL: https://www.forefront.ai/app/chat/new. (Дата звернення: 03.10.2023).

4. Perplexity AI Search Results. URL: https://www.perplexity.ai/search/cc7a9501-f46a-4de7-9cca-2c59954e60a2?s=u. (Дата звернення: 03.10.2023).

5. OpenAI. Our Approach to Alignment Research. URL: https://openai.com/blog/our-approach-to-alignment-research. (Дата звернення: 03.10.2023).

6. Yin Zhang, Rong Jin, Zhi-Hua Zhou. 2010. Understanding bag-of-words model: A statistical framework // International Journal of Machine Learning and Cybernetics. 2010. No. 1(1). P. 43 – 52. DOI:10.1007/s13042-010-0001-0.

7. Shahzad Qaiser, Ramsha Ali. Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents // International Journal of Computer Applications. 2018, vol. 181, No. 1. P. 25 – 29. DOI:10.5120/ijca2018917395.

8. Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space // arXiv:1301.3781v3 [cs.CL]. 2013. 12 p. URL: https://arxiv.org/pdf/1301.3781.pdf.

9. Tomas Mikolov, Quoc Le. Distributed Representations of Sentences and Documents // Proceedings of the 31 st International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32. 2014. 9 p. URL: https://arxiv.org/pdf/1405.4053.pdf.

10. Bojanowski P., Grave E., Joulin A., Mikolov T. Enriching Word Vectors with Subword Information // arXiv:1607.04606 [cs.CL]. 2017. 12 p. URL: https://arxiv.org/pdf/1607.04606.pdf.

11. Vaswan A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser L. Attention Is All You Need // 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA. 2017. 15 p. URL: https://arxiv.org/pdf/1706.03762.pdf.

12. Devlin J., Chang M., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding // arXiv:1810.04805 [cs.CL]. 2019. 16 p. URL: https://arxiv.org/pdf/1810.04805.pdf.

13. Глибовець М. М., Олецький О.В. Системи штучного інтелекту. К. : КМ Академія, 2002. 366 с.

14. Russel S., Norvig P. Artificial Intelligence. A modern approach. Pearson Education Limited, 2003. 1170 p.

15. Luger G. F. Artificial Intelligence. Structures and Strategies for Complex Problem Solving. Pearson Education Limited, 2005. 903 p.

16. Haykin S. Neural Networks and Learning Machines. Pearson, 2011. 936 p.

17. Bratko I. Prolog Programming for Artificial Intelligence. Pearson Education, 2012. 560 с.

18. Субботін С.О. Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень. Запоріжжя : ЗНТУ, 2008. 341 с.

19. Ming Zhou. Progress in Neural NLP: Modeling, Learning, and Reasoning / Ming Zhou, Nan Duan, Shujie Liu, Heung-Yeung Shum // Engineering. Vol. 6, Issue 3. 2020. P. 275-290.

20. Nilsson N.J. Artificial Intelligence: A New Synthesis. Elsevier Inc., 1998. 493 p.

21. Руденко О. Г., Бодянський Є.В. Штучні нейронні мережі: навчальний посібник. Харків : ТОВ "Компанія СМІТ", 2006. 404 с.

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) **Магістр**
Галузь знань: **15 – Автоматизація та приладобудування**
Спеціальність: **151 – «Автоматизація та комп'ютерно-інтегровані технології»**

ЗАТВЕРДЖУЮ
Завідувач кафедри теоретичної та прикладної системотехніки
_____**д.т.н., проф. Шматков С. І.**

«08 » грудня 2022 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Захаров Микита Олександрович
(прізвище, ім'я, по батькові студента)

1. Тема роботи «Нейромережеві моделі обробки текстових даних»

керівник роботи Стрілець Вікторія Євгенівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «10» листопада 2023 року № 4101-5/3197

2. Строк подання студентом роботи 28 .11.2023

3. Перелік питань, які потрібно розробити
1. Аналіз нейромережих моделей для обробки текстової інформації.
2. Розробка методити тестування нейромережих моделей для оборобки текстових даних.
3. Формулювання запитів для тестування нейромережевих моделей.
4. Дослідження ефективності роботи нейромережих моделей для оборобки текстових даних.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Проведення літературного огляду з проблематики застосування різних моделей для обробки текстових інформації різного типу	08.12.2022 — 06.03.2023
2	Огляд нейромережих моделей та алгоритми їх застосування для обробки інформації	06.03.2023 — 01.04.2023
3	Розробка методики тестування моделі GPT-3 на різних задачах для обробки текстової інформації	01.04.2023 — 27.06.2023
4	Пошук даних для тестування моделі GPT-3	20.06.2023 — 24.08.2023
5	Аналіз результатів тестування та визначення ефективності використання моделі GPT-3 для обробки текстових даних у задачах	15.08.2023 — 17.09.2023
6	Розробка рекомендацій щодо застосування моделі GPT-3 для обробки текстових даних у задачах	17.08.2023 — 20.09.2023
7	Оформлення пояснювальної записки	01.09.2023 — 27.10.2023
8	Представлення кваліфікаційної роботи керівнику та рецензенту	28.10.2023 - 28.11.2023

5. Дата видачі завдання 08.12.2022 р.

Студент М.О. Захаров _____
ініціали, прізвище підпис

Керівник роботи В.Є. Стрілець _____
ініціали, прізвище підпис

Додаток Б

ІНДИВІДУАЛЬНЕ ТЕХНІЧНЕ ЗАВДАННЯ

Технічне завдання

на розробку програмного виробу

«Методологія оцінювання нейромережевих моделей обробки тексту»

Назва розділу	Назва і зміст підрозділу
1. Введення	1.1. Назва програмного виробу: Методологія оцінювання нейромережевих моделей обробки тексту. 1.2. Галузь застосування: Автоматизація та аналіз станів комп'ютерних систем.
2. Підстава для розробки	2.1. Навчальний план за спеціальністю 151 – «Автоматизація та комп'ютерно-інтегровані технології» 2.2. Завдання на кваліфікаційну роботу магістра затверджено наказом ректора № 4101-5/3197 від 10.11.2023р.
3.Призначення розробки	3.1. Мета розробки програмного виробу: оцінка ефективності використання нейромережевих моделей GPT-3.5 TURBO, GPT-4, Claude 1.2 для обробки текстових даних у задачах різного типу. 3.2. Призначення програмного виробу: використання виробу для оцінкИ ефективності нейромережевих моделей на основі розробленої методології оцінювання. 3.3. Вихідні дані для розробки: результати тестування алгоритмів, оцінка ефективності нейромережевих моделей на різних алгоритмах в табличному виді, рекомендації по написаню запиту.

4. Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: можливість оцінити ефективності використання нейромережевих моделей GPT-3.5 TURBO, GPT-4, Claude 1.2. 4.2. Вимоги до надійності: забезпечення працездатності оцінки ефективності використання нейромережевих моделей. 4.3. Вимоги до умов експлуатації: встановлення мови програмування Python та необхідних бібліотек для розробки тестування. 4.4. Вимоги до складу і параметрів технічних засобів: комп'ютер або ноутбук із 4 ГБ оперативної пам'яті та процесором не нижче Intel Core i3 9-го покоління. 4.5. Вимоги до інформаційної та програмної сумісності: ОС Linux або Windows 8/10, Python 3. 4.6. Вимоги до маркування та упаковки: жорстка упаковка з пластмаси, маркування на українській та англійській мовах. 4.7. Вимоги до транспортування і зберігання: транспортування в упаковці будь-яким способом, температура транспортування/зберігання +5-+20 С. 4.8. Спеціальні вимоги: відсутні.
5. Вимоги до програмної документації.	Програмною документацією до виробу «Методологія оцінювання нейромережевих моделей обробки тексту» вважати: 1) Справжнє Технічне завдання на розробку програмного виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Програму і методику випробувань розробленого програмного виробу (представити у вигляді Додатку В до пояснювальної записки до кваліфікаційної роботи). 3) Опис програмного виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи). 4) Текст програми (представити в Додатку Г, Д до пояснювальної записки до кваліфікаційної роботи).

6. Техніко економічні показники	Визначення ефективності розв’язання математичних задач різного типу існуючими нейромережевими моделями обробки тексту: виконати в розділі 4 пояснювальної записки до кваліфікаційної роботи.	
7. Стадії і етапи розробки	Дата	Назва етапу
	01.09.2023 - 18.09.2023	1. Проведення літературного огляду з проблематики застосування різних моделей для обробки текстових інформації різного типу
	19.09.2023 - 30.09.2023	2. Огляд нейромережих моделей та алгоритми їх застосування для обробки інформації
	01.10.2023 - 25.10.2023	3. Розробка методики тестування нейромережевих моделей на різних задачах для обробки текстової інформації
	26.10.2023 - 16.11.2023	4. Пошук даних для тестування моделі
	17.11.2023 - 20.11.2023	5. Аналіз результатів тестування та визначення ефективності використання нейромережевих моделей для обробки текстових даних у задачах
	20.11.2023 - 31.11.2023	6. Розробка рекомендацій щодо застосування нейромережевих

		моделей для обробки текстових даних у задачах
8. Порядок контролю і приймання	В даному розділі повинні бути вказані загальні вимоги до приймання розробленого програмного виробу наприклад: 1) Перевірка ходу розробки програмного виробу. Керівнику робіт виконувати 1 раз в 3 тижні. 2) Випробування програмного виробу відповідно до Програми і методики випробувань провести на базі комп'ютерного класу або приватного приміщення. 3) Захист розробленого програмного виробу провести на засіданні атестаційної комісії. 4) Пояснювальну записку надати на паперових носіях в одному примірнику, в електронному вигляді.	

Виконавець:

студент групи КУ-61

Захаров М.О.

Замовник:

кандидат технічих наук

Стрілець В.Є.

Додаток В

Програма і методика випробувань програмного виробу

«Методологія оцінювання нейромережевих моделей обробки тексту»

1. Об'єкт випробувань

1. Назва програмного виробу: «Методологія оцінювання нейромережевих моделей обробки тексту».

2. Галузь застосування: Автоматизація та аналіз станів комп’ютерних систем.

3. Перераховані відомості запозичуються з відповідних розділів Технічного завдання.

2. Мета випробувань

Перевірка відповідності функціональності програмної реалізації системи заявленим функціональним можливостям в технічному завданні (Додаток Б до пояснювальної записки до дипломної роботи).

3. Загальні положення

3.1. Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

3.2 Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі комп'ютерного класу кафедри або приватного приміщення в період роботи атестаційної комісії.

3.3 Обсяг випробувань

Приймальні випробування програмного вироби проводяться в обсязі відповідному цієї програми і методики випробувань.

3.4 Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавця та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

Модель повинна задовольняти наступним вимогам:

– працювати на основних операційних системах: Windows, Linux, MacOS;

– вимоги до надійності;

– передбачити захист від некоректних дій користувача;

– сумісність з іншими програмними продуктами;

– бути легко розширюваною;

– елементи програми повинні бути ізольовані одне від одного для зменшення їх впливу на роботи програми під час редагування програмного коду;

– вимоги до складу і параметрів технічних засобів;

– вимоги до маркування та упаковки (не висуваються);

– вимоги до транспортування і зберігання (не висуваються).

– Спеціальні вимоги (не пред’являються).

5. Вимоги до програмної документації

Програмною документацією до виробу «Методологія оцінювання нейромережевих моделей обробки тексту» вважати:

– справжнє технічне завдання на розробку програми (представити як Додаток Б до пояснювальної записки до кваліфікаційної роботи);

– програму і методику випробувань розробленої програми (представити як Додаток В до пояснювальної записки до кваліфікаційної роботи);

– рекомендацій щодо застосування створеної програмної стандартизації у проектах (представити в Розділі 3 пояснювальної записки до кваліфікаційної роботи);

– текст програми (представити як Додаток Г, Д до пояснювальної записки до кваліфікаційної роботи).

6. Засоби і порядок випробувань

6.1. Засоби випробувань

Для проведення випробувань необхідний проект для розробки програмної моделі за допомогою мові програмування Python з використання бібліотек timeit, time, cProfile, line_profiler, memory_profiler та кодом алгоритмів сгенеровних нейромережою.

6.2. Порядок проведення випробувань

Як правило, випробування проводяться в два етапи:

- ознайомчий (1-й етап);

- випробування програмного виробу (2-й етап).

Перелік перевірок, що проводяться на 1 етапі випробувань, включає в себе:

1. Перевірку комплектності програмної документації.

2. Перевірка комплектності складу програмної документації здійснюється за критерієм наявності зазначеної в ТЗ документації.

3. Перевірку комплектності складу технічних і програмних засобів.

4. Методику проведення перевірок на 1 етапі випробувань.

5. Якість програмної документації перевіряється на відповідність вимогам стандартів ЕСПД.

Перелік перевірок, що проводяться на 2 етапі випробувань, включає в себе:

1. Перевірку відповідності технічних характеристик програми вимогам технічного завдання.

2. Перевірку ступеня виконання функціональних вимог до програми.

3. Методику проведення перевірок, що входять до переліку по 2 етапу випробувань.

1. Програма працює відповідно до умов експлуатації операційних систем MS Windows, Linux та MacOS.

2. Для роботи необхідний компілятор мови програмування Python, версії не нижчої ніж 3.0.

3. Порядок проведення випробувань:

3.1. Для початку отримуємо сгенерований код алгоритму від нейромережі;

3.2. Після цього, отриманий код потрібно помістити в окрему ру діректорію для коректної праці memory_profiler.

3.3. Якщо цього потребує алгоритм – ввести вхідні дані.

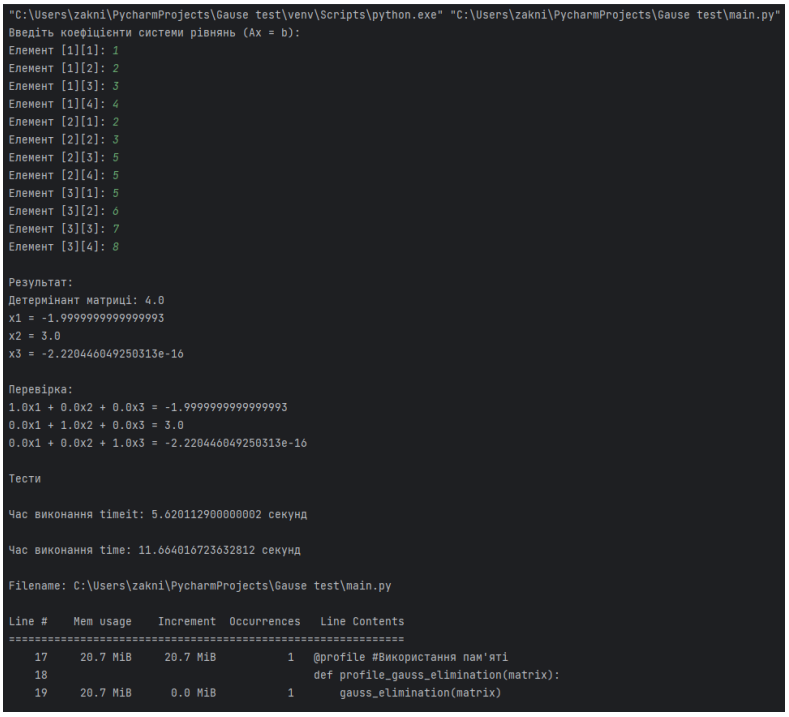
3.4. Після чого необхідно запустити програму тестування за допомогою запуску програми на мові програмування python “python <ім’я файлу>;

3.5. Після натискання на екрані з’явиться результат тестування.

Для проведення випробувань пропонується провести тест 1, тест 2, тест 3.

Тест 1

1. Отримуємо код методу Гауса від ChatGPT.
2. Записуємо в окрему ру діректорію.
3. Вводимо вхідні дані.
4. Запускаємо програму
5. Отримуємо результати тестування



```
"C:\Users\zakni\PycharmProjects\Gause test\venv\Scripts\python.exe" "C:\Users\zakni\PycharmProjects\Gause test\main.py"
Введіть коефіцієнти системи рівнянь (Ax = b):
Елемент [1][1]: 1
Елемент [1][2]: 2
Елемент [1][3]: 3
Елемент [1][4]: 4
Елемент [2][1]: 2
Елемент [2][2]: 3
Елемент [2][3]: 5
Елемент [2][4]: 5
Елемент [3][1]: 5
Елемент [3][2]: 6
Елемент [3][3]: 7
Елемент [3][4]: 8

Результат:
Детермінант матриці: 4.0
x1 = -1.9999999999999993
x2 = 3.0
x3 = -2.220446049250313e-16

Перевірка:
1.0x1 + 0.0x2 + 0.0x3 = -1.9999999999999993
0.0x1 + 1.0x2 + 0.0x3 = 3.0
0.0x1 + 0.0x2 + 1.0x3 = -2.220446049250313e-16

Тести

Час виконання timeit: 5.620112900000002 секунд

Час виконання time: 11.664016723632812 секунд

Filename: C:\Users\zakni\PycharmProjects\Gause test\main.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    17     20.7 MiB     20.7 MiB           1   @profile #Використання пам'яті
    18                                         def profile_gauss_elimination(matrix):
    19     20.7 MiB      0.0 MiB           1       gauss_elimination(matrix)
```

Рис. В.1 Тест 1 результати тестування методу Гауса від ChatGPT

```
Total time: 4.56e-05 s
File: C:\Users\zakni\PycharmProjects\Gause test\GaussChatGPT.py
Function: gauss_elimination at line 1

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     1                                           def gauss_elimination(matrix):
     2         1          7.0      7.0      1.5      n = len(matrix)
     3         1          3.0      3.0      0.7      det = 1.0  # Ініціалізуємо детермінант
     4
     5                                               # Прямий хід
     6         4         15.0      3.8      3.3      for i in range(n):
     7                                                   # Пошук максимального елементу в стовпці під поточним елементом
     8         3          8.0      2.7      1.8          max_row = i
     9         6         26.0      4.3      5.7          for k in range(i+1, n):
    10         3         19.0      6.3      4.2              if abs(matrix[k][i]) > abs(matrix[max_row][i]):
    11                                                           max_row = k
    12         3         14.0      4.7      3.1          matrix[i], matrix[max_row] = matrix[max_row], matrix[i]
    13
    14                                                   # Зробити поточний елемент головним
    15         3          9.0      3.0      2.0          pivot = matrix[i][i]
    16         3         10.0      3.3      2.2          det *= pivot  # Оновлюємо детермінант
    17        12         40.0      3.3      8.8          for j in range(i, n+1):
    18         9         34.0      3.8      7.5              matrix[i][j] /= pivot
    19
    20                                                   # Обнулити інші рядки
    21        12         40.0      3.3      8.8          for k in range(n):
    22         9         26.0      2.9      5.7              if k != i:
    23         6         20.0      3.3      4.4                  factor = matrix[k][i]
    24        24         79.0      3.3     17.3                  for j in range(i, n+1):
    25        18         80.0      4.4     17.5                      matrix[k][j] -= factor * matrix[i][j]
    26
    27                                               # Зворотний хід
    28         1         23.0     23.0      5.0      solutions = [row[-1] for row in matrix]
    29
    30         1          3.0      3.0      0.7      return det, solutions
```

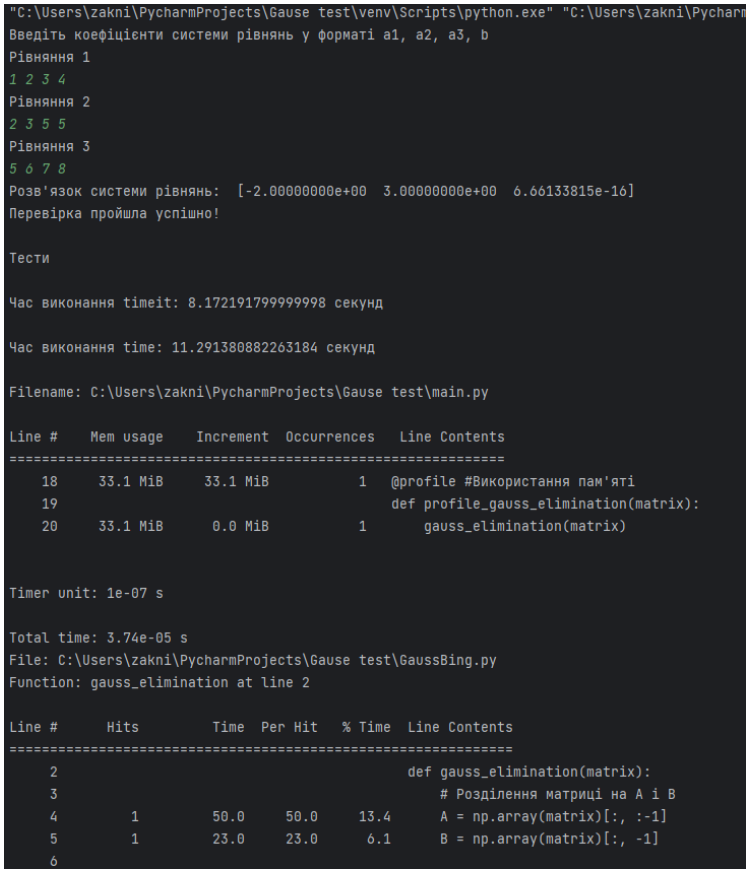
Рис. В.2 Тест 1 результати тестування методу Гауса від ChatGPT

```
Профіль коду cProfile
         12 function calls in 0.000 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 GaussChatGPT.py:1(gauss_elimination)
        1    0.000    0.000    0.000    0.000 GaussChatGPT.py:28(<listcomp>)
        6    0.000    0.000    0.000    0.000 {built-in method builtins.abs}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.exec}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.len}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}



Process finished with exit code 0
```

Рис. В.3 Тест 1 результати тестування методу Гауса від ChatGPT

Тест 2

1. Отримуємо код методу Гауса від Bing.
2. Записуємо в окрему py діректорію.
3. Вводимо вхідні дані.
4. Запускаємо програму
5. Отримуємо результати тестування



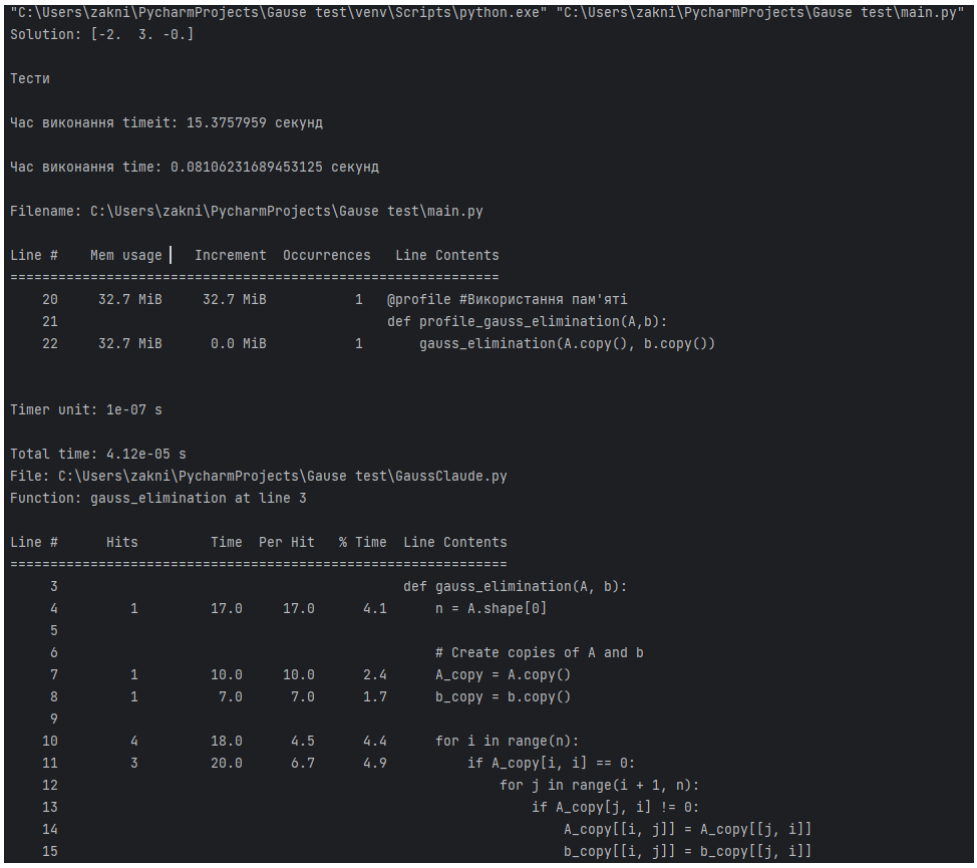
```
"C:\Users\zakni\PycharmProjects\Gause test\venv\Scripts\python.exe" "C:\Users\zakni\Pycharm
Введіть коефіцієнти системи рівнянь у форматі a1, a2, a3, b
Рівняння 1
1 2 3 4
Рівняння 2
2 3 5 5
Рівняння 3
5 6 7 8
Розв'язок системи рівнянь:  [-2.00000000e+00  3.00000000e+00  6.66133815e-16]
Перевірка пройшла успішно!

Тести

Час виконання timeit: 8.172191799999998 секунд

Час виконання time: 11.291380882263184 секунд

Filename: C:\Users\zakni\PycharmProjects\Gause test\main.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    18     33.1 MiB     33.1 MiB           1   @profile #Використання пам'яті
    19                                         def profile_gauss_elimination(matrix):
    20     33.1 MiB      0.0 MiB           1       gauss_elimination(matrix)


Timer unit: 1e-07 s

Total time: 3.74e-05 s
File: C:\Users\zakni\PycharmProjects\Gause test\GaussBing.py
Function: gauss_elimination at line 2

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     2                                           def gauss_elimination(matrix):
     3                                               # Розділення матриці на A і B
     4         1         50.0     50.0     13.4      A = np.array(matrix)[:, :-1]
     5         1         23.0     23.0      6.1      B = np.array(matrix)[:, -1]
     6
     7                                               # Розв'язок системи рівнянь методом Гауса
     8         1        298.0    298.0     79.7      X = np.linalg.solve(A, B)
     9         1          3.0      3.0      0.8      return X, A, B


Профіль коду cProfile
         32 function calls in 0.000 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 GaussBing.py:2(gauss_elimination)
        1    0.000    0.000    0.000    0.000 linalg.py:130(get_linalg_error_extobj)
        2    0.000    0.000    0.000    0.000 linalg.py:135(_makearray)
        3    0.000    0.000    0.000    0.000 linalg.py:140(isComplexType)
        2    0.000    0.000    0.000    0.000 linalg.py:153(_realType)
        1    0.000    0.000    0.000    0.000 linalg.py:159(_commonType)
        1    0.000    0.000    0.000    0.000 linalg.py:203(_assert_stacked_2d)
        1    0.000    0.000    0.000    0.000 linalg.py:209(_assert_stacked_square)
        1    0.000    0.000    0.000    0.000 linalg.py:325(_solve_dispatcher)
        1    0.000    0.000    0.000    0.000 linalg.py:329(solve)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.exec}
        2    0.000    0.000    0.000    0.000 {built-in method builtins.getattr}
        5    0.000    0.000    0.000    0.000 {built-in method builtins.issubclass}
        2    0.000    0.000    0.000    0.000 {built-in method numpy.array}
        2    0.000    0.000    0.000    0.000 {built-in method numpy.asarray}
        1    0.000    0.000    0.000    0.000 {method '__array_prepare__' of 'numpy.ndarray' objects}
        1    0.000    0.000    0.000    0.000 {method 'astype' of 'numpy.ndarray' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
        2    0.000    0.000    0.000    0.000 {method 'get' of 'dict' objects}



Process finished with exit code 0
```

Рис. В.4 Тест 2 результати тестування методу Гауса від Bing

Рис. В.5 Тест 2 результати тестування методу Гауса від Bing

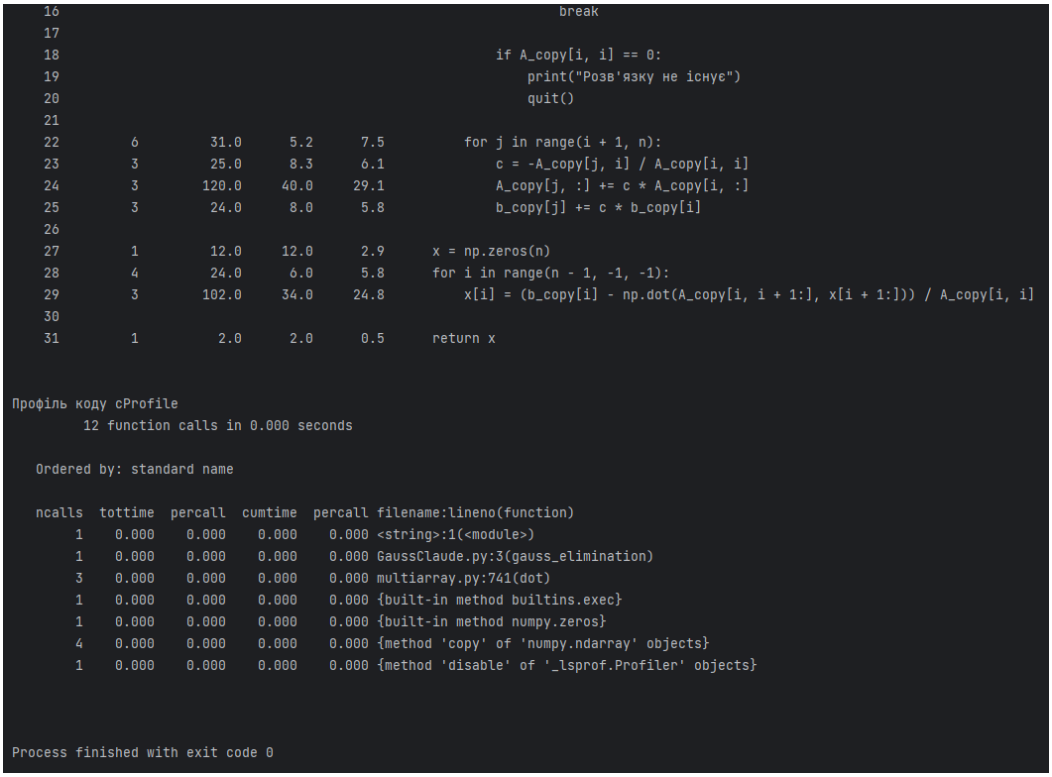
Тест 3

1. Отримуємо код методу Гауса від Claude 1.2.
2. Записуємо в окрему py діректорію.
3. Вводимо вхідні дані.
4. Запускаємо програму
5. Отримуємо результати тестування



```
"C:\Users\zakni\PycharmProjects\Gause test\venv\Scripts\python.exe" "C:\Users\zakni\PycharmProjects\Gause test\main.py"
Solution: [-2.  3. -0.]

Тести

Час виконання timeit: 15.3757959 секунд

Час виконання time: 0.08106231689453125 секунд

Filename: C:\Users\zakni\PycharmProjects\Gause test\main.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    20     32.7 MiB     32.7 MiB           1   @profile #Використання пам'яті
    21                                         def profile_gauss_elimination(A,b):
    22     32.7 MiB      0.0 MiB           1       gauss_elimination(A.copy(), b.copy())


Timer unit: 1e-07 s

Total time: 4.12e-05 s
File: C:\Users\zakni\PycharmProjects\Gause test\GaussClaude.py
Function: gauss_elimination at line 3

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     3                                           def gauss_elimination(A, b):
     4         1         17.0     17.0      4.1      n = A.shape[0]
     5
     6                                               # Create copies of A and b
     7         1         10.0     10.0      2.4      A_copy = A.copy()
     8         1          7.0      7.0      1.7      b_copy = b.copy()
     9
    10         4         18.0      4.5      4.4      for i in range(n):
    11         3         20.0      6.7      4.9          if A_copy[i, i] == 0:
    12                                                       for j in range(i + 1, n):
    13                                                           if A_copy[j, i] != 0:
    14                                                               A_copy[[i, j]] = A_copy[[j, i]]
    15                                                               b_copy[[i, j]] = b_copy[[j, i]]
```

Рис. В.6 Тест 3 результати тестування методу Гауса від Claude 1.2



```
    16                                                  break
    17
    18                                              if A_copy[i, i] == 0:
    19                                                  print("Розв'язку не існує")
    20                                                  quit()
    21
    22         6         31.0      5.2      7.5          for j in range(i + 1, n):
    23         3         25.0      8.3      6.1              c = -A_copy[j, i] / A_copy[i, i]
    24         3        120.0     40.0     29.1              A_copy[j, :] += c * A_copy[i, :]
    25         3         24.0      8.0      5.8              b_copy[j] += c * b_copy[i]
    26
    27         1         12.0     12.0      2.9      x = np.zeros(n)
    28         4         24.0      6.0      5.8      for i in range(n - 1, -1, -1):
    29         3        102.0     34.0     24.8          x[i] = (b_copy[i] - np.dot(A_copy[i, i + 1:], x[i + 1:])) / A_copy[i, i]
    30
    31         1          2.0      2.0      0.5      return x


Профіль коду cProfile
         12 function calls in 0.000 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 GaussClaude.py:3(gauss_elimination)
        3    0.000    0.000    0.000    0.000 multiarray.py:741(dot)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.exec}
        1    0.000    0.000    0.000    0.000 {built-in method numpy.zeros}
        4    0.000    0.000    0.000    0.000 {method 'copy' of 'numpy.ndarray' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}



Process finished with exit code 0
```

Рис. В.7 Тест 2 результати тестування методу Гауса від Claude 1.2

Висновки: випробування пройшло успішно, оскільки кожен з тестів показав очікуванні результати.

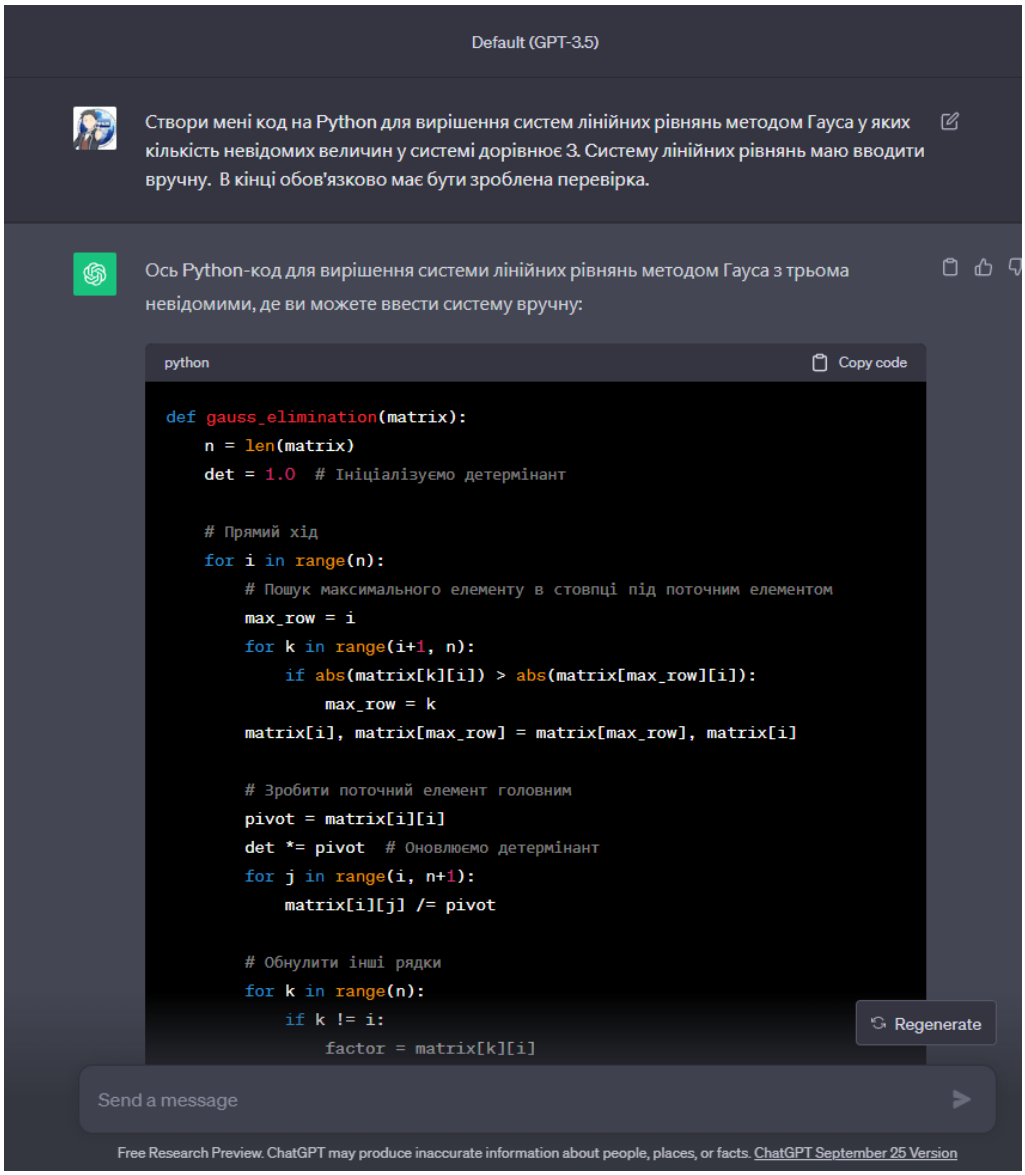
Виконавець:

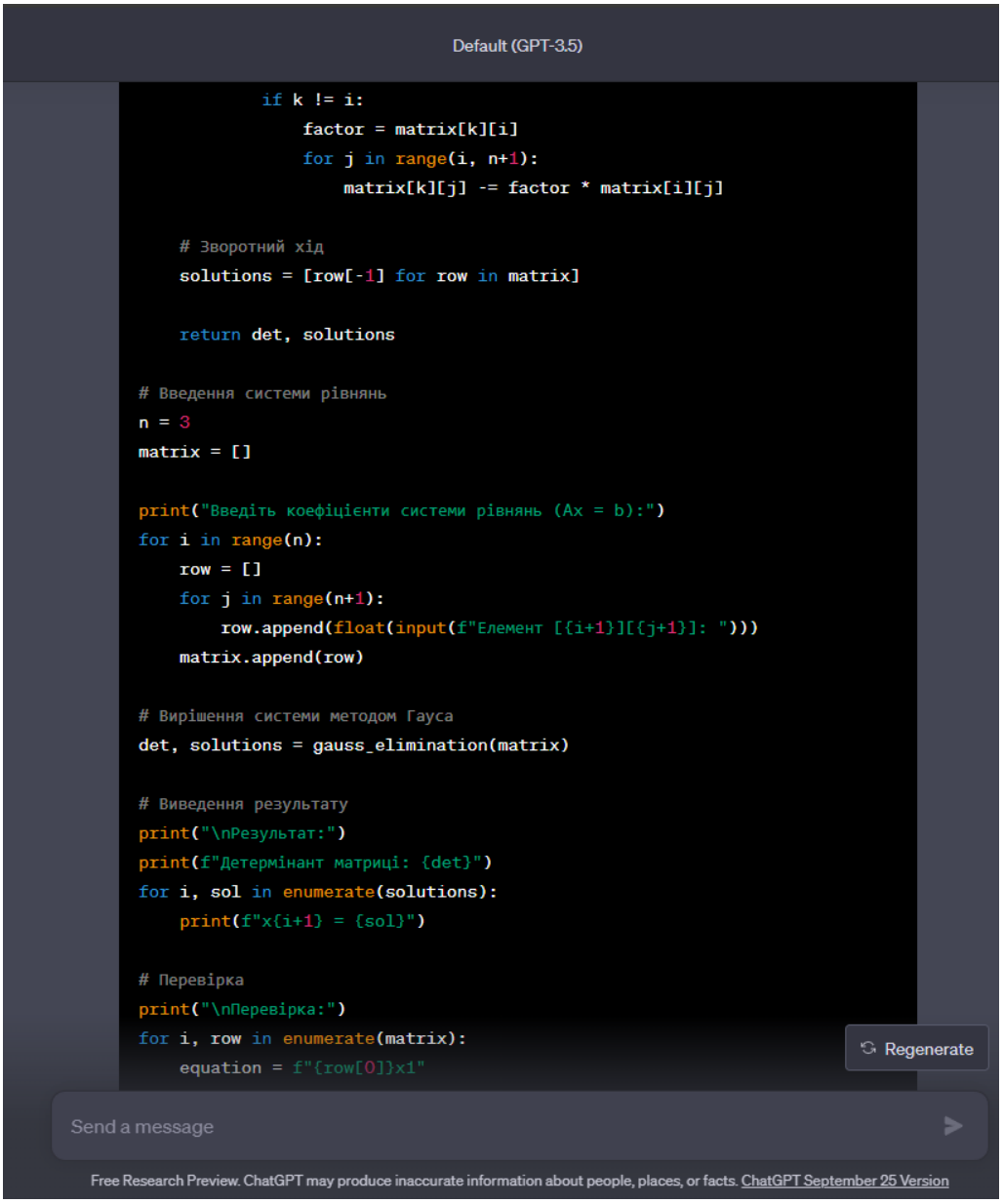
студент групи КУ-61, Захаров М. О.

Додаток Г

Скріншоти виконання запитів

Результат після запиту в ChatGPT по методу Гауса

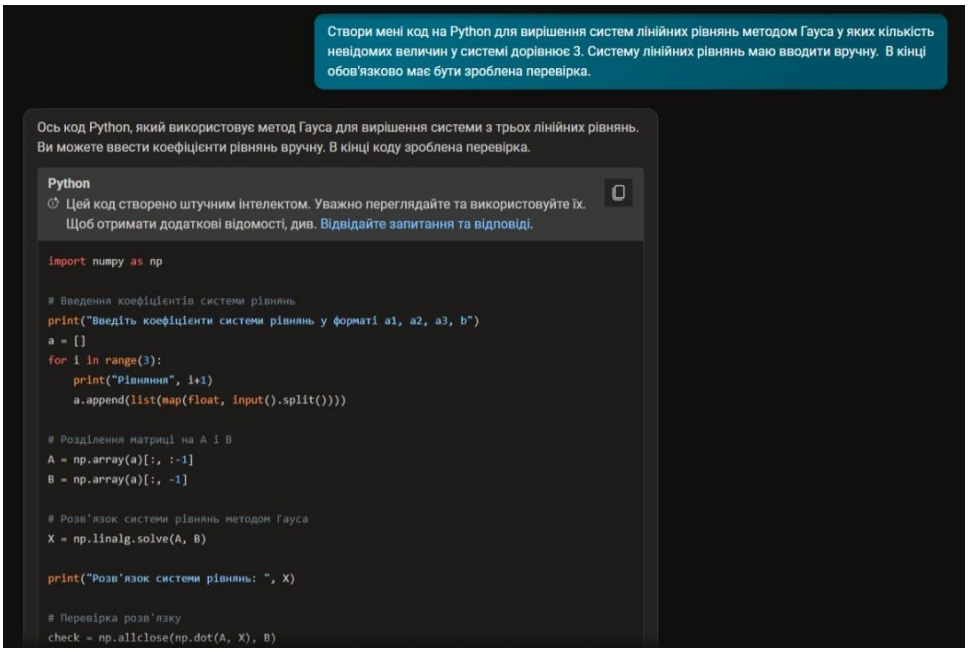


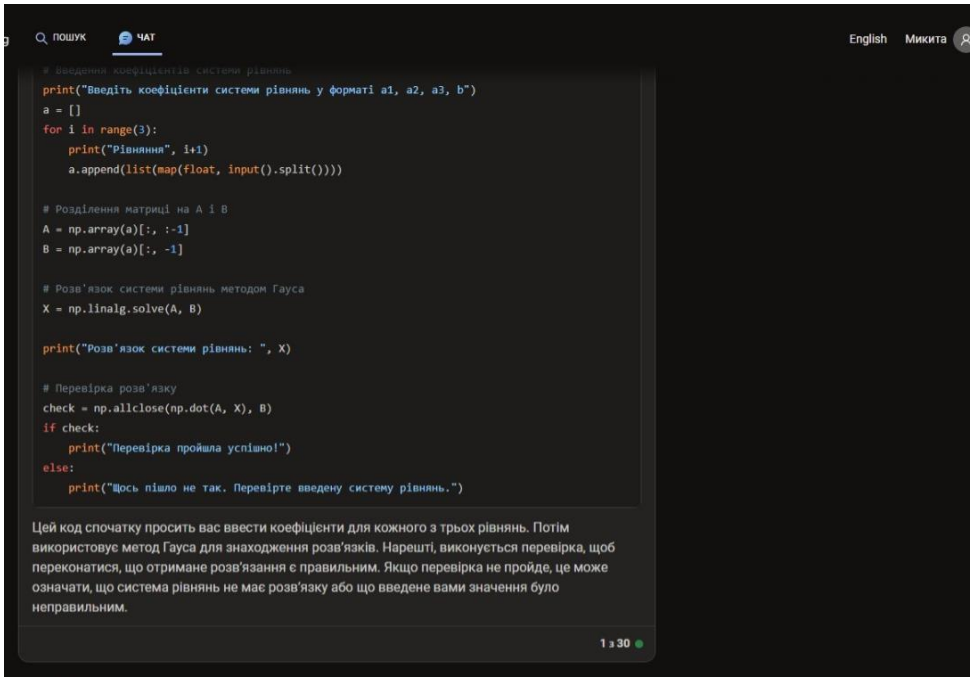



```
    for j in range(1, n):
        equation += f" + {row[j]}x{j+1}"
    equation += f" = {row[-1]}"
    print(equation)
```

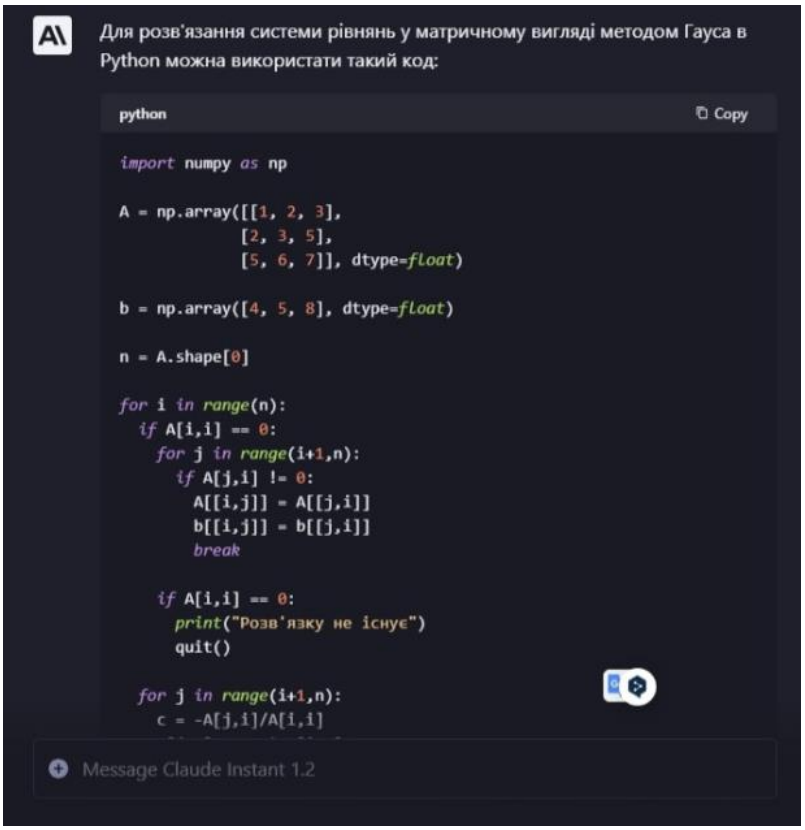
Цей код обчислює детермінант матриці методом Гауса та виводить його разом із рішенням системи рівнянь.

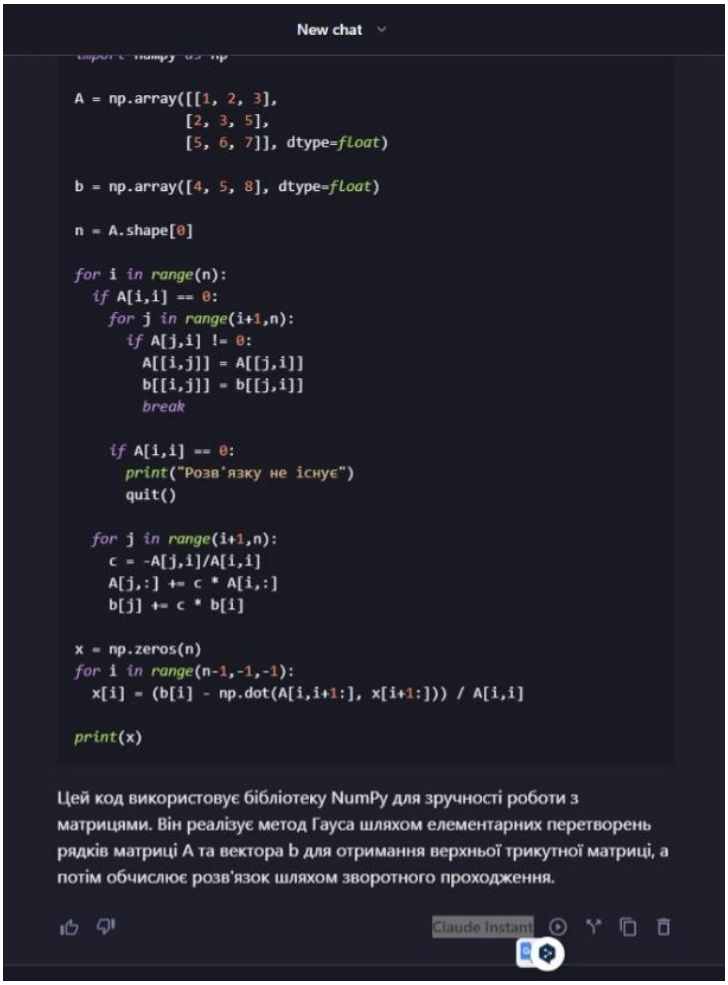
Результат після запиту в BingAI по методу Гауса



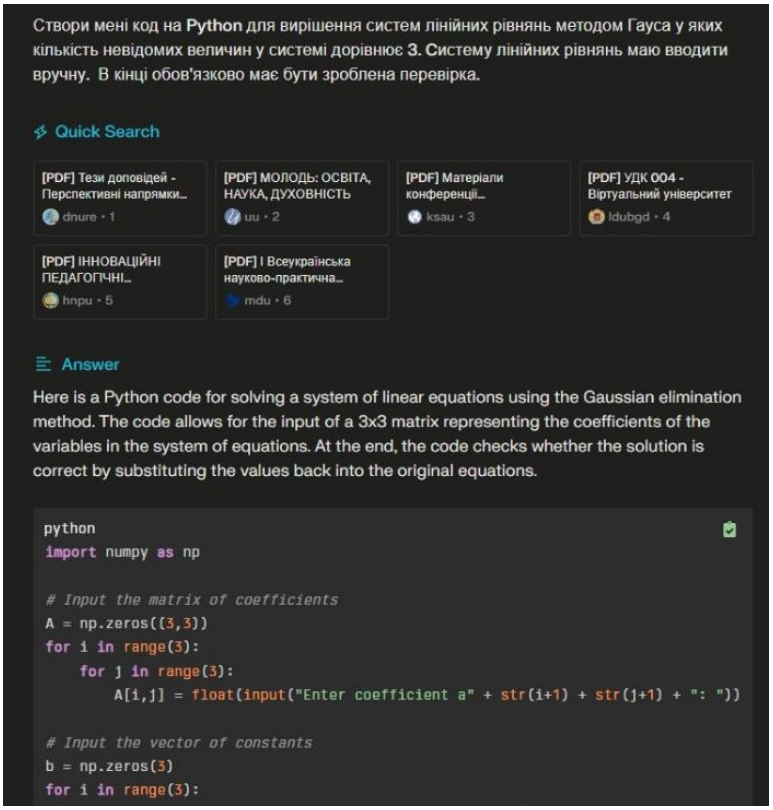


Результат після запиту в Claude 1 по методу Гауса





Результат після запиту в Perplexity по методу Гауса



Створи мені код на **Python** для вирішення систем лінійних рівнянь методом Гауса у яких кількість невідомих величин у системі дорівнює **3**. Систему лінійних рівнянь маю вводити вручну. В кінці обов'язково має бути зроблена перевірка.

Quick Search

[PDF] Тези доповідей - Перспективні напрямки... dnure · 1

[PDF] МОЛОДЬ: ОСВІТА, НАУКА, ДУХОВНІСТЬ uu · 2

[PDF] Матеріали конференції... ksau · 3

[PDF] УДК 004 - Віртуальний університет ldubgd · 4

[PDF] ІННОВАЦІЙНІ ПЕДАГОГІЧНІ... hnpu · 5

[PDF] І Всеукраїнська науково-практична... mdu · 6

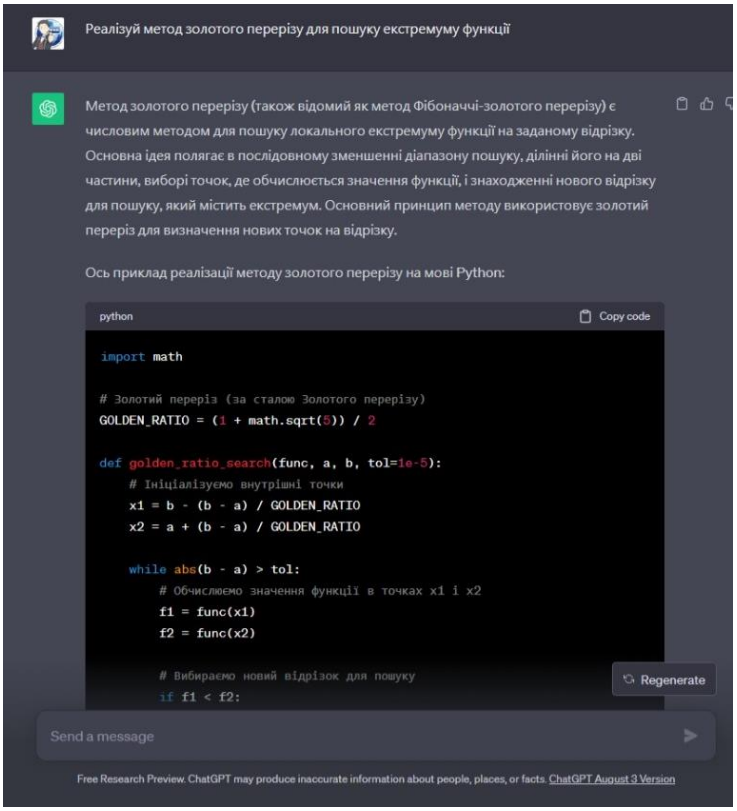
Answer

Here is a Python code for solving a system of linear equations using the Gaussian elimination method. The code allows for the input of a 3x3 matrix representing the coefficients of the variables in the system of equations. At the end, the code checks whether the solution is correct by substituting the values back into the original equations.

```python
import numpy as np

# Input the matrix of coefficients
A = np.zeros((3,3))
for i in range(3):
    for j in range(3):
        A[i,j] = float(input("Enter coefficient a" + str(i+1) + str(j+1) + ": "))

# Input the vector of constants
b = np.zeros(3)
for i in range(3):
    for j in range(3):
        A[i,j] = float(input("Enter coefficient a" + str(i+1) + str(j+1) + ": "))

# Input the vector of constants
b = np.zeros(3)
for i in range(3):
    b[i] = float(input("Enter constant b" + str(i+1) + ": "))

# Perform Gaussian elimination
for i in range(3):
    for j in range(i+1, 3):
        factor = A[j,i]/A[i,i]
        A[j,:] -= factor*A[i,:]
        b[j] -= factor*b[i]

# Back-substitution to find the solution
x = np.zeros(3)
for i in range(2, -1, -1):
    x[i] = (b[i] - np.dot(A[i,i+1:], x[i+1:]))/A[i,i]

# Print the solution
print("The solution is: x =", x)

# Check the solution
for i in range(3):
    check = np.dot(A[i,:], x)
    print("Equation", i+1, "is satisfied:", check == b[i])
```

Note that this code assumes that the system of equations has a unique solution. If the system is inconsistent or has infinitely many solutions, the code will not work correctly.

Результат після запиту в ChatGPT по методу золотого перетину



Реалізуй метод золотого перерізу для пошуку екстремуму функції

Метод золотого перерізу (також відомий як метод Фібоначчі-золотого перерізу) є числовим методом для пошуку локального екстремуму функції на заданому відрізку. Основна ідея полягає в послідовному зменшенні діапазону пошуку, ділінні його на дві частини, виборі точок, де обчислюється значення функції, і знаходженні нового відрізку для пошуку, який містить екстремум. Основний принцип методу використовує золотий переріз для визначення нових точок на відрізку.

Ось приклад реалізації методу золотого перерізу на мові Python:

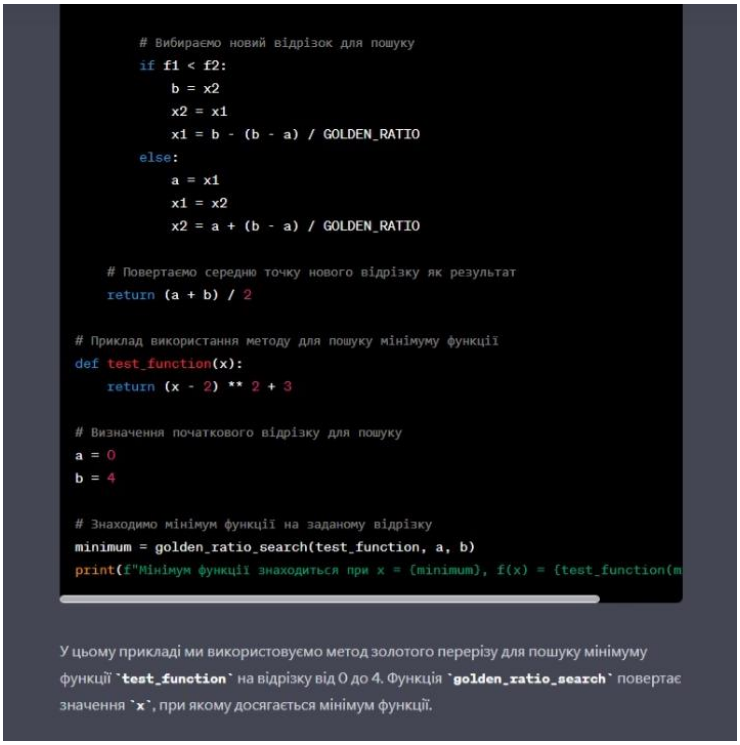
python Copy code

```python
import math

# Золотий переріз (за сталою Золотого перерізу)
GOLDEN_RATIO = (1 + math.sqrt(5)) / 2

def golden_ratio_search(func, a, b, tol=1e-5):
    # Ініціалізуємо внутрішні точки
    x1 = b - (b - a) / GOLDEN_RATIO
    x2 = a + (b - a) / GOLDEN_RATIO

    while abs(b - a) > tol:
        # Обчислюємо значення функції в точках x1 і x2
        f1 = func(x1)
        f2 = func(x2)

        # Вибираємо новий відрізок для пошуку
        if f1 < f2:
```

Regenerate

Send a message

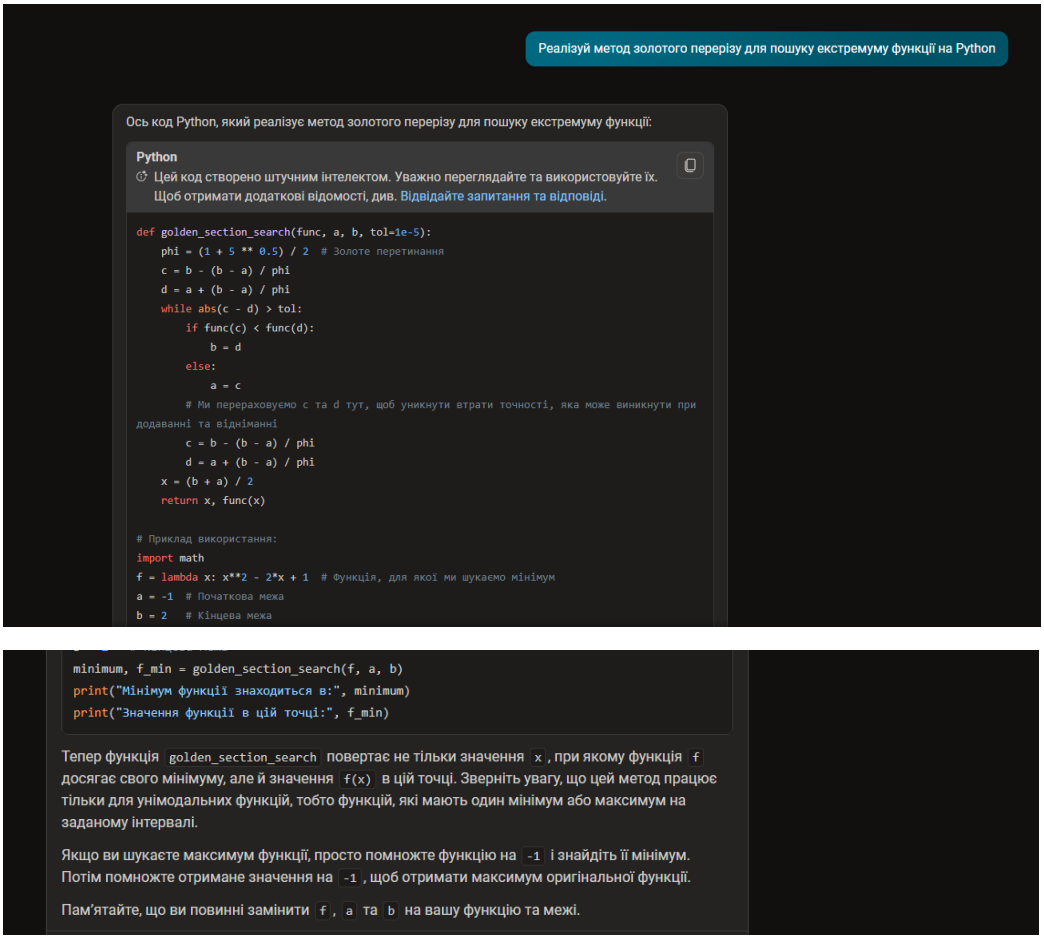
Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT August 3 Version



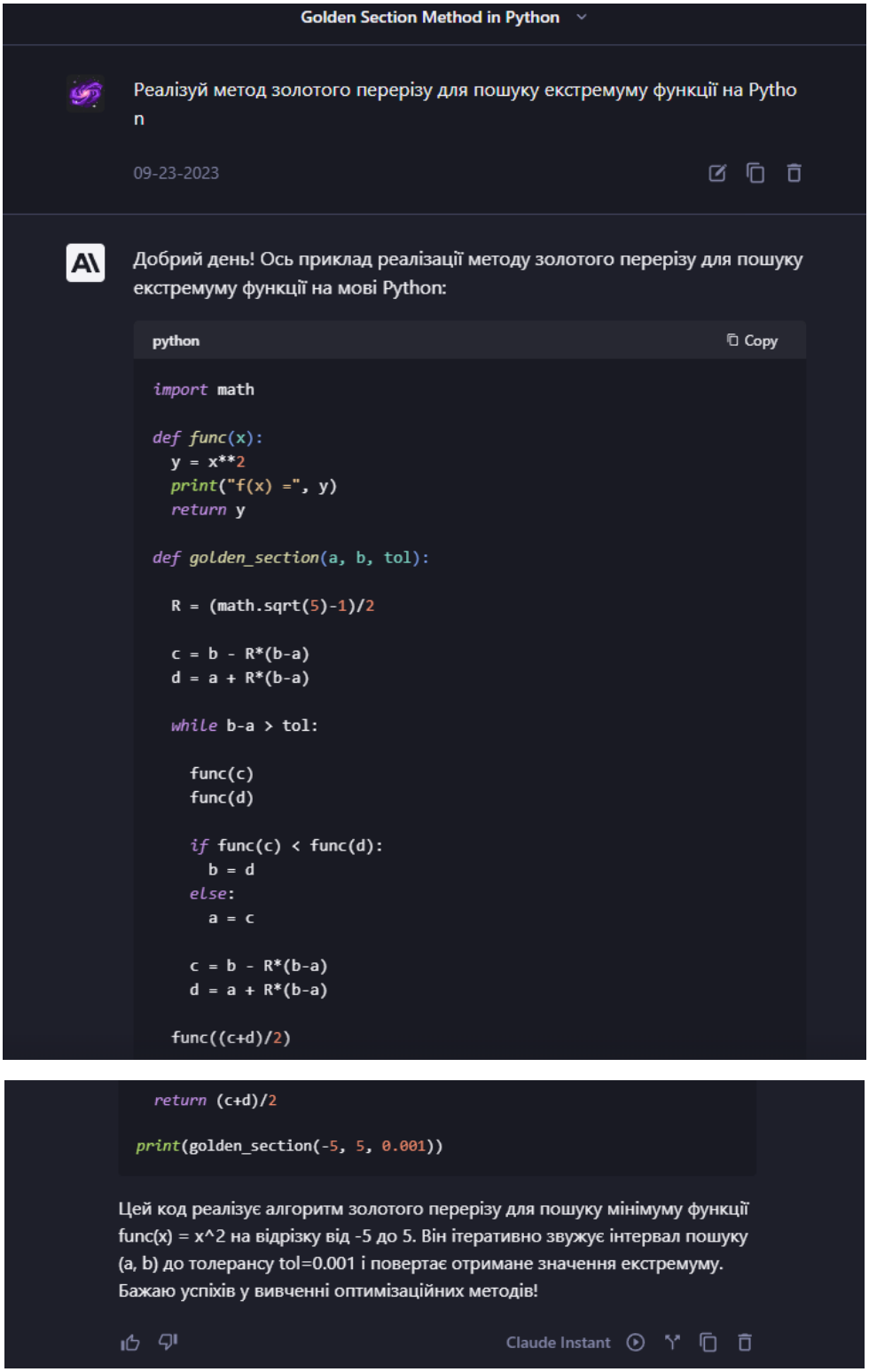
```python
        # Вибираємо новий відрізок для пошуку
        if f1 < f2:
            b = x2
            x2 = x1
            x1 = b - (b - a) / GOLDEN_RATIO
        else:
            a = x1
            x1 = x2
            x2 = a + (b - a) / GOLDEN_RATIO

    # Повертаємо середню точку нового відрізку як результат
    return (a + b) / 2

# Приклад використання методу для пошуку мінімуму функції
def test_function(x):
    return (x - 2) ** 2 + 3

# Визначення початкового відрізку для пошуку
a = 0
b = 4

# Знаходимо мінімум функції на заданому відрізку
minimum = golden_ratio_search(test_function, a, b)
print(f"Мінімум функції знаходиться при x = {minimum}, f(x) = {test_function(m
```

У цьому прикладі ми використовуємо метод золотого перерізу для пошуку мінімуму функції **`test_function`** на відрізку від 0 до 4. Функція **`golden_ratio_search`** повертає значення **`x`**, при якому досягається мінімум функції.

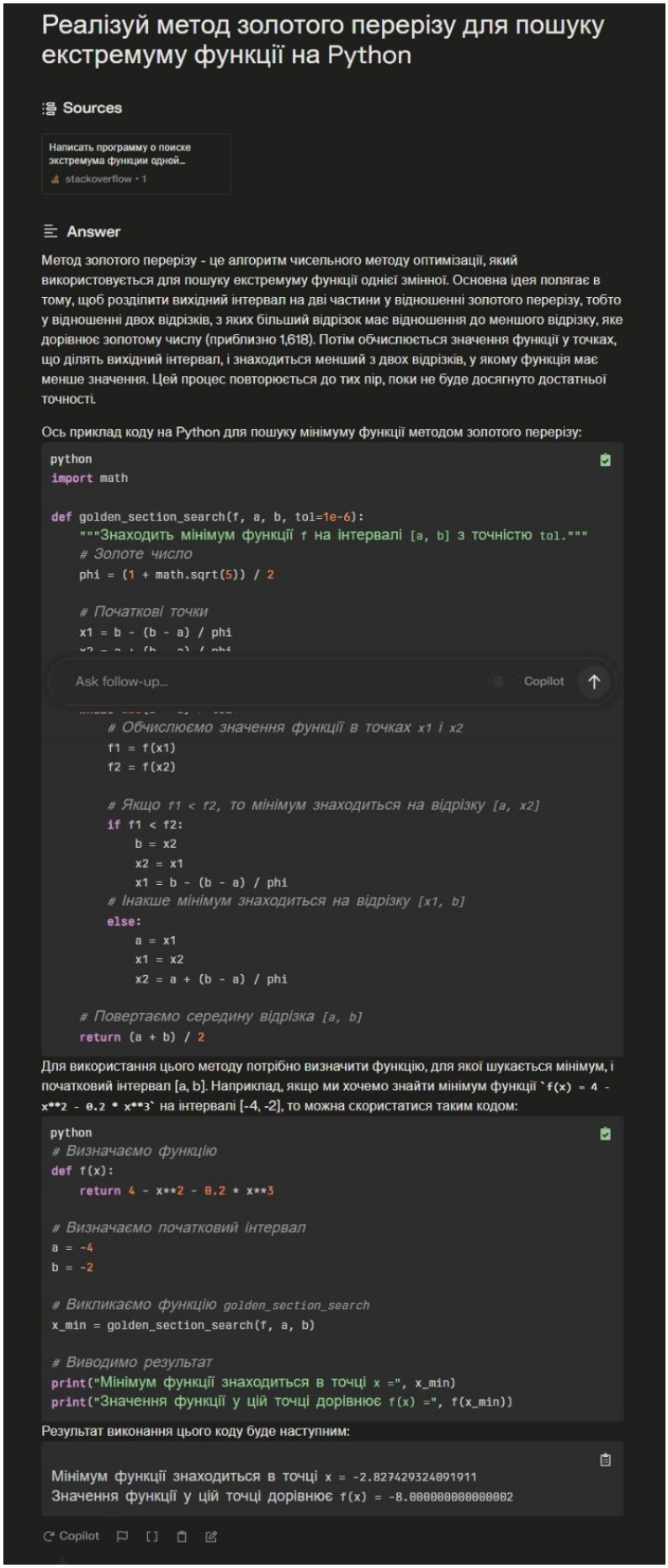
Результат після запиту в BingAI по методу золотого перетину



Результат після запиту в Claude 1 по методу золотого перетину



Результат після запиту в Perplexity по методу золотого перетину



Реалізуй метод золотого перерізу для пошуку екстремуму функції на Python

Sources

Написать программу о поиске экстремума функции одной...
stackoverflow · 1

Answer

Метод золотого перерізу - це алгоритм чисельного методу оптимізації, який використовується для пошуку екстремуму функції однієї змінної. Основна ідея полягає в тому, щоб розділити вихідний інтервал на дві частини у відношенні золотого перерізу, тобто у відношенні двох відрізків, з яких більший відрізок має відношення до меншого відрізку, яке дорівнює золотому числу (приблизно 1,618). Потім обчислюється значення функції у точках, що ділять вихідний інтервал, і знаходиться менший з двох відрізків, у якому функція має менше значення. Цей процес повторюється до тих пір, поки не буде досягнуто достатньої точності.

Ось приклад коду на Python для пошуку мінімуму функції методом золотого перерізу:

```python
import math

def golden_section_search(f, a, b, tol=1e-6):
    """Знаходить мінімум функції f на інтервалі [a, b] з точністю tol."""
    # Золоте число
    phi = (1 + math.sqrt(5)) / 2

    # Початкові точки
    x1 = b - (b - a) / phi
    x2 = a + (b - a) / phi

    while abs(b - a) > tol:
        # Обчислюємо значення функції в точках x1 і x2
        f1 = f(x1)
        f2 = f(x2)

        # Якщо f1 < f2, то мінімум знаходиться на відрізку [a, x2]
        if f1 < f2:
            b = x2
            x2 = x1
            x1 = b - (b - a) / phi
        # Інакше мінімум знаходиться на відрізку [x1, b]
        else:
            a = x1
            x1 = x2
            x2 = a + (b - a) / phi

    # Повертаємо середину відрізка [a, b]
    return (a + b) / 2
```

Ask follow-up... Copilot

Для використання цього методу потрібно визначити функцію, для якої шукається мінімум, і початковий інтервал [a, b]. Наприклад, якщо ми хочемо знайти мінімум функції `f(x) = 4 - x**2 - 0.2 * x**3` на інтервалі [-4, -2], то можна скористатися таким кодом:

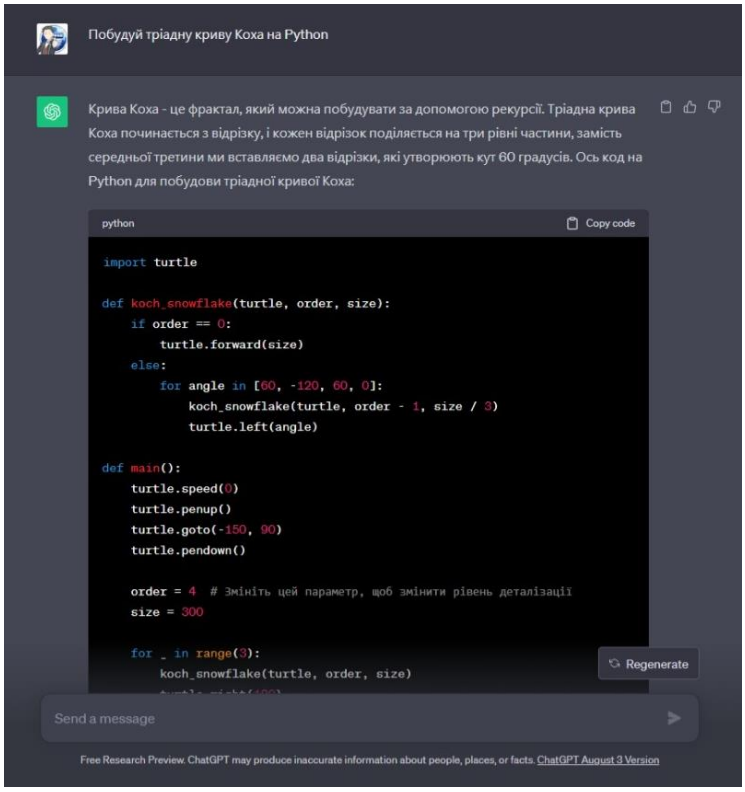
```python
# Визначаємо функцію
def f(x):
    return 4 - x**2 - 0.2 * x**3

# Визначаємо початковий інтервал
a = -4
b = -2

# Викликаємо функцію golden_section_search
x_min = golden_section_search(f, a, b)

# Виводимо результат
print("Мінімум функції знаходиться в точці x =", x_min)
print("Значення функції у цій точці дорівнює f(x) =", f(x_min))
```

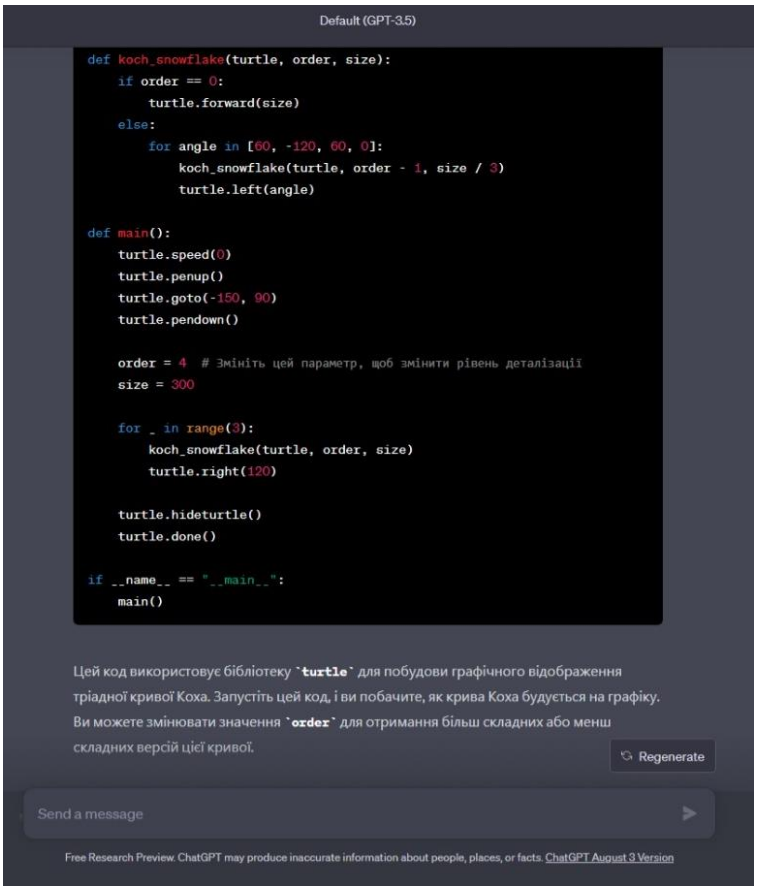
Результат виконання цього коду буде наступним:

```
Мінімум функції знаходиться в точці x = -2.827429324091911
Значення функції у цій точці дорівнює f(x) = -8.000000000000002
```

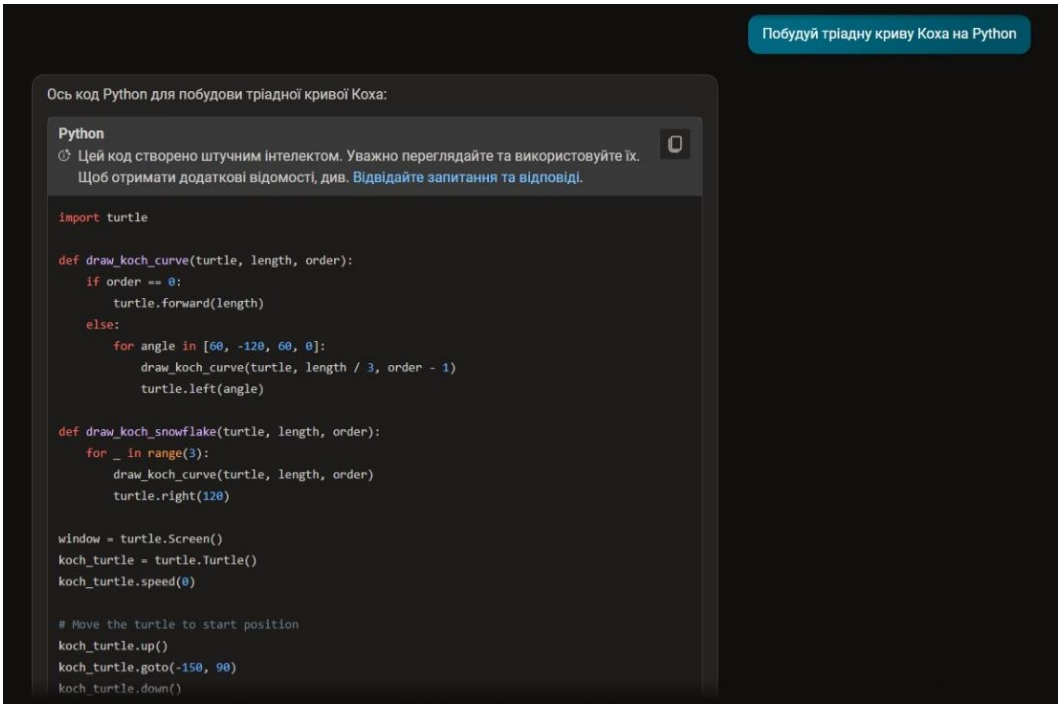
Copilot

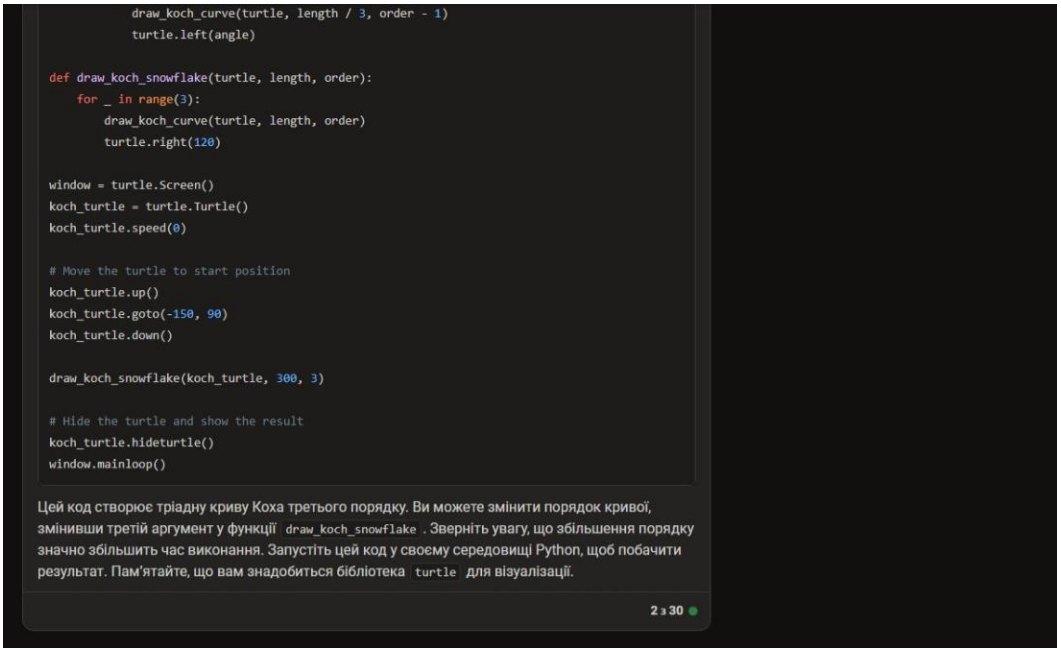
Результат після запиту в ChatGPT по фракталу Коха



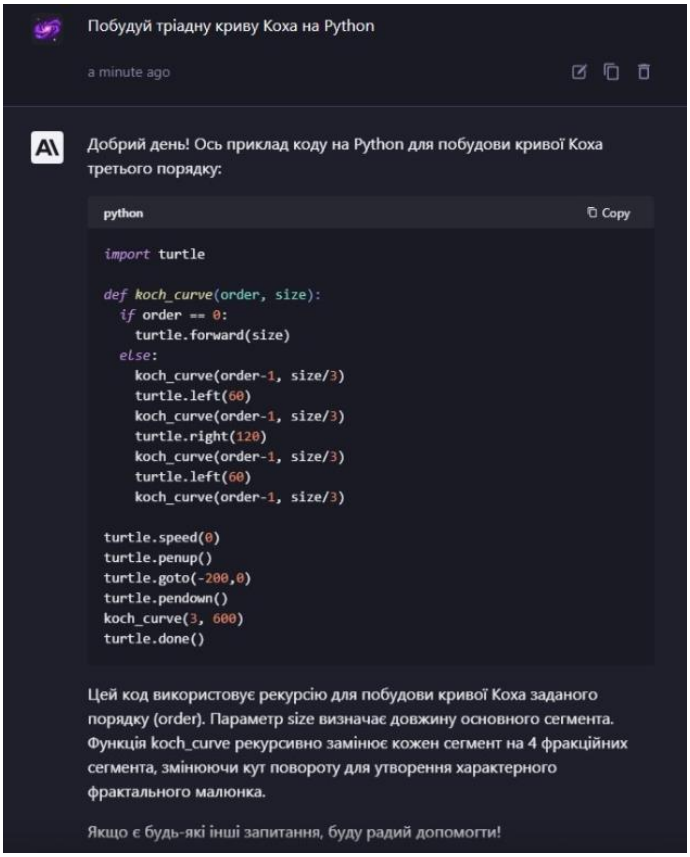


Результат після запиту в BingAI по фракталу Коха

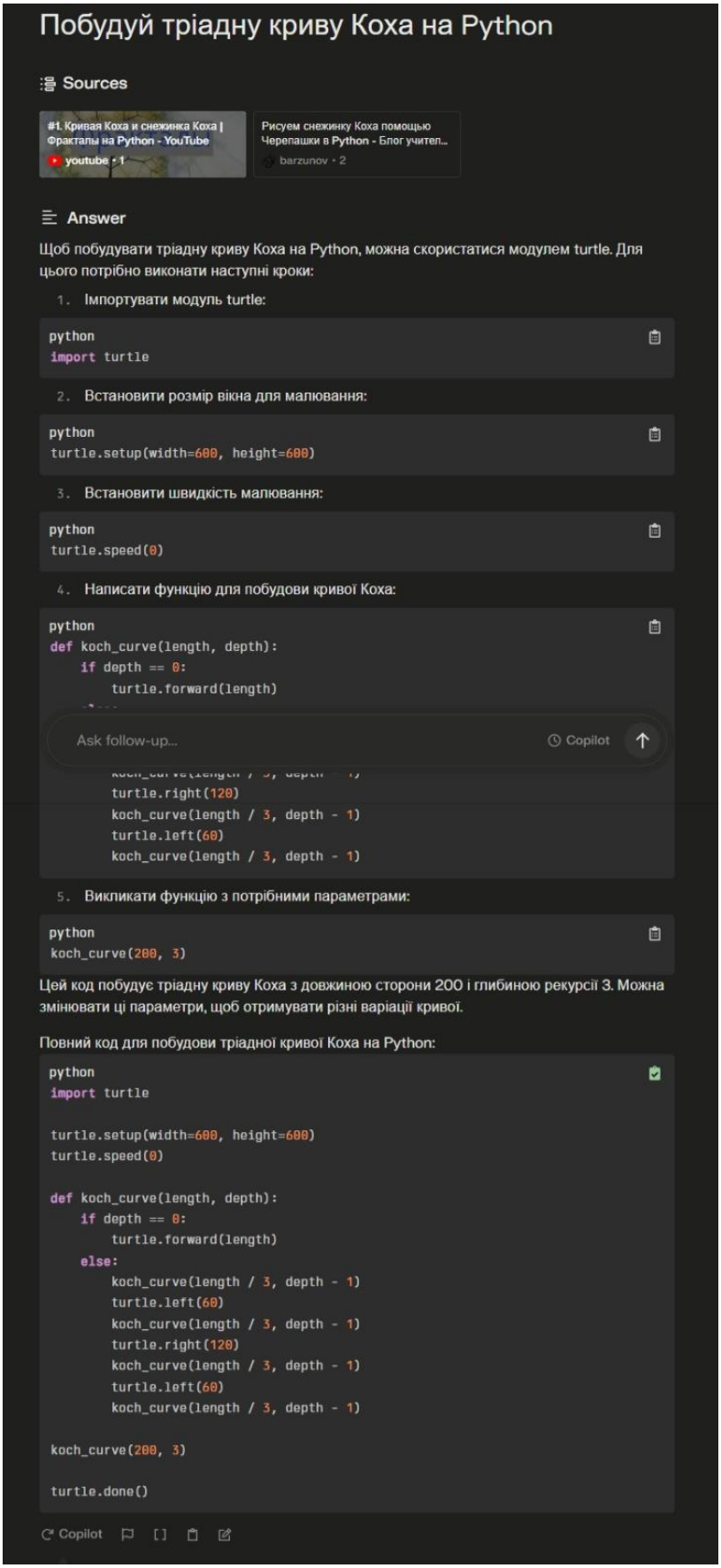




Результат після запиту в Claude 1 по фракталу Коха



Результат після запиту в Perplexity фракталу Коха



Побудуй тріадну криву Коха на Python

Sources

#1 Кривая Коха и снежинка Коха | Фракталы на Python - YouTube
youtube • 1

Рисуем снежинку Коха помощью Черепашки в Python - Блог учител...
barzunov • 2

Answer

Щоб побудувати тріадну криву Коха на Python, можна скористатися модулем turtle. Для цього потрібно виконати наступні кроки:

1. Імпортувати модуль turtle:

```python
import turtle
```

2. Встановити розмір вікна для малювання:

```python
turtle.setup(width=600, height=600)
```

3. Встановити швидкість малювання:

```python
turtle.speed(0)
```

4. Написати функцію для побудови кривої Коха:

```python
def koch_curve(length, depth):
    if depth == 0:
        turtle.forward(length)
```

Ask follow-up... Copilot

```
        turtle.right(120)
        koch_curve(length / 3, depth - 1)
        turtle.left(60)
        koch_curve(length / 3, depth - 1)
```

5. Викликати функцію з потрібними параметрами:

```python
koch_curve(200, 3)
```

Цей код побудує тріадну криву Коха з довжиною сторони 200 і глибиною рекурсії 3. Можна змінювати ці параметри, щоб отримувати різні варіації кривої.

Повний код для побудови тріадної кривої Коха на Python:

```python
import turtle

turtle.setup(width=600, height=600)
turtle.speed(0)

def koch_curve(length, depth):
    if depth == 0:
        turtle.forward(length)
    else:
        koch_curve(length / 3, depth - 1)
        turtle.left(60)
        koch_curve(length / 3, depth - 1)
        turtle.right(120)
        koch_curve(length / 3, depth - 1)
        turtle.left(60)
        koch_curve(length / 3, depth - 1)

koch_curve(200, 3)

turtle.done()
```

Copilot

Додаток Д

Лістинг програмного коду

Лістинг результата після запиту в ChatGPT по методу Гауса

```
def gauss_elimination(matrix):
    n = len(matrix)
    det = 1.0  # Ініціалізуємо детермінант

    # Прямий хід
    for i in range(n):
        # Пошук максимального елементу в стовпці під поточним елементом
        max_row = i
        for k in range(i+1, n):
            if abs(matrix[k][i]) > abs(matrix[max_row][i]):
                max_row = k
        matrix[i], matrix[max_row] = matrix[max_row], matrix[i]

        # Зробити поточний елемент головним
        pivot = matrix[i][i]
        det *= pivot  # Оновлюємо детермінант
        for j in range(i, n+1):
            matrix[i][j] /= pivot

        # Обнулити інші рядки
        for k in range(n):
            if k != i:
                factor = matrix[k][i]
                for j in range(i, n+1):
                    matrix[k][j] -= factor * matrix[i][j]

    # Зворотний хід
    solutions = [row[-1] for row in matrix]

    return det, solutions

# Введення системи рівнянь
n = 3
matrix = []

print("Введіть коефіцієнти системи рівнянь (Ax = b):")
for i in range(n):
    row = []
    for j in range(n+1):
        row.append(float(input(f'Елемент [{i+1}][{j+1}]: ")))
    matrix.append(row)

# Вирішення системи методом Гауса
det, solutions = gauss_elimination(matrix)

# Виведення результату
```

```
print("\nРезультат:")
print(f'Детермінант матриці: {det}")
for i, sol in enumerate(solutions):
    print(f"x{i+1} = {sol}")

# Перевірка
print("\nПеревірка:")
for i, row in enumerate(matrix):
    equation = f"{row[0]}x1"
    for j in range(1, n):
        equation += f" + {row[j]}x{j+1}"
    equation += f" = {row[-1]}"
    print(equation)
```

Лістинг результата після запиту в BingAI по методу Гауса

```
import numpy as np

# Введення коефіцієнтів системи рівнянь
print("Введіть коефіцієнти системи рівнянь у форматі a1, a2, a3, b")
a = []
for i in range(3):
    print("Рівняння", i+1)
    a.append(list(map(float, input().split())))

# Розділення матриці на A і B
A = np.array(a)[:, :-1]
B = np.array(a)[:, -1]

# Розв'язок системи рівнянь методом Гауса
X = np.linalg.solve(A, B)

print("Розв'язок системи рівнянь: ", X)

# Перевірка розв'язку
check = np.allclose(np.dot(A, X), B)
if check:
    print("Перевірка пройшла успішно!")
else:
    print("Щось пішло не так. Перевірте введену систему рівнянь.")
```

Лістинг результата після запиту в Claude 1 по методу Гауса

```
import numpy as np

A = np.array([[1, 2, 3],
              [2, 3, 5],
              [5, 6, 7]], dtype=float)

b = np.array([4, 5, 8], dtype=float)

n = A.shape[0]
```

```
for i in range(n):
  if A[i,i] == 0:
    for j in range(i+1,n):
      if A[j,i] != 0:
        A[[i,j]] = A[[j,i]]
        b[[i,j]] = b[[j,i]]
        break

    if A[i,i] == 0:
      print("Розв'язку не існує")
      quit()

  for j in range(i+1,n):
    c = -A[j,i]/A[i,i]
    A[j,:] += c * A[i,:]
    b[j] += c * b[i]

x = np.zeros(n)
for i in range(n-1,-1,-1):
  x[i] = (b[i] - np.dot(A[i,i+1:], x[i+1:])) / A[i,i]

print(x)
```

Лістинг результата після запиту в Perplexity по методу Гауса

```
import numpy as np

# Input the matrix of coefficients
A = np.zeros((3,3))
for i in range(3):
    for j in range(3):
        A[i,j] = float(input("Enter coefficient a" + str(i+1) + str(j+1) + ": "))

# Input the vector of constants
b = np.zeros(3)
for i in range(3):
    b[i] = float(input("Enter constant b" + str(i+1) + ": "))

# Perform Gaussian elimination
for i in range(3):
    for j in range(i+1, 3):
        factor = A[j,i]/A[i,i]
        A[j,:] -= factor*A[i,:]
        b[j] -= factor*b[i]

# Back-substitution to find the solution
x = np.zeros(3)
for i in range(2, -1, -1):
    x[i] = (b[i] - np.dot(A[i,i+1:], x[i+1:]))/A[i,i]

# Print the solution
print("The solution is: x =", x)
```

```
# Check the solution
for i in range(3):
    check = np.dot(A[i,:], x)
    print("Equation", i+1, "is satisfied:", check == b[i])
```

Лістинг результата після запиту в ChatGPT по методу золотого перетину

```
import math

# Золотий переріз (за сталою Золотого перерізу)
GOLDEN_RATIO = (1 + math.sqrt(5)) / 2

def golden_ratio_search(func, a, b, tol=1e-5):
    # Ініціалізуємо внутрішні точки
    x1 = b - (b - a) / GOLDEN_RATIO
    x2 = a + (b - a) / GOLDEN_RATIO

    while abs(b - a) > tol:
        # Обчислюємо значення функції в точках x1 і x2
        f1 = func(x1)
        f2 = func(x2)

        # Вибираємо новий відрізок для пошуку
        if f1 < f2:
            b = x2
            x2 = x1
            x1 = b - (b - a) / GOLDEN_RATIO
        else:
            a = x1
            x1 = x2
            x2 = a + (b - a) / GOLDEN_RATIO

    # Повертаємо середню точку нового відрізку як результат
    return (a + b) / 2

# Приклад використання методу для пошуку мінімуму функції
def test_function(x):
    return math.exp(1/x) + math.log(2*x)

# Визначення початкового відрізку для пошуку
a = 1.764
b = 2.236

# Знаходимо мінімум функції на заданому відрізку
minimum = golden_ratio_search(test_function, a, b)
print(f"Мінімум функції знаходиться при x = {minimum}, f(x) =
{test_function(minimum)}")
```

Лістинг результата після запиту в BingAI по методу золотого перетину

```
def golden_section_search(func, a, b, tol=1e-5):
    phi = (1 + 5 ** 0.5) / 2  # Золоте перетинання
    c = b - (b - a) / phi
    d = a + (b - a) / phi
    while abs(c - d) > tol:
        if func(c) < func(d):
            b = d
        else:
            a = c
        # Ми перераховуємо c та d тут, щоб уникнути втрати точності, яка може
виникнути при додаванні та відніманні
        c = b - (b - a) / phi
        d = a + (b - a) / phi
    x = (b + a) / 2
    return x, func(x)

# Приклад використання:
import math
f = lambda x: math.exp(1/x) + math.log(2*x)  # Функція, для якої ми шукаємо мінімум
a = 1.764  # Початкова межа
b = 2.236   # Кінцева межа
minimum, f_min = golden_section_search(f, a, b)
print("Мінімум функції знаходиться в:", minimum)
print("Значення функції в цій точці:", f_min)
```

Лістинг результата після запиту в Claude 1по методу золотого перетину

```
import math

def func(x):
  y = math.exp(1/x) + math.log(2*x)
  print("f(x) =", y)
  return y

def golden_section(a, b, tol):

  R = (math.sqrt(5)-1)/2

  c = b - R*(b-a)
  d = a + R*(b-a)

  while b-a > tol:

    func(c)
    func(d)

    if func(c) < func(d):
      b = d
```

```
    else:
      a = c

    c = b - R*(b-a)
    d = a + R*(b-a)

  func((c+d)/2)

  return (c+d)/2

print(golden_section(1.764, 2.236, 0.001))
```

Лістинг результата після запиту в Perplexity по методу золотого перетину

```
import math

def golden_section_search(f, a, b, tol=1e-6):
    """Знаходить мінімум функції f на інтервалі [a, b] з точністю tol."""
    # Золоте число
    phi = (1 + math.sqrt(5)) / 2

    # Початкові точки
    x1 = b - (b - a) / phi
    x2 = a + (b - a) / phi

    # Поки не досягнуто достатньої точності
    while abs(b - a) > tol:
        # Обчислюємо значення функції в точках x1 і x2
        f1 = f(x1)
        f2 = f(x2)

        # Якщо f1 < f2, то мінімум знаходиться на відрізку [a, x2]
        if f1 < f2:
            b = x2
            x2 = x1
            x1 = b - (b - a) / phi
        # Інакше мінімум знаходиться на відрізку [x1, b]
        else:
            a = x1
            x1 = x2
            x2 = a + (b - a) / phi

    # Повертаємо середину відрізка [a, b]
    return (a + b) / 2

# Визначаємо функцію
def f(x):
    return math.exp(1/x) + math.log(2*x)

# Визначаємо початковий інтервал
a = 1.764
```

```
b = 2.236

# Викликаємо функцію golden_section_search
x_min = golden_section_search(f, a, b)

# Виводимо результат
print("Мінімум функції знаходиться в точці x =", x_min)
print("Значення функції у цій точці дорівнює f(x) =", f(x_min))
```

Лістинг результата після запиту в ChatGPT по фракталу Коха

```
import turtle

def koch_snowflake(turtle, order, size):
    if order == 0:
        turtle.forward(size)
    else:
        for angle in [60, -120, 60, 0]:
            koch_snowflake(turtle, order - 1, size / 3)
            turtle.left(angle)

def main():
    turtle.speed(0)
    turtle.penup()
    turtle.goto(-150, 90)
    turtle.pendown()

    order = 4  # Змініть цей параметр, щоб змінити рівень деталізації
    size = 300

    for _ in range(3):
        koch_snowflake(turtle, order, size)
        turtle.right(120)

    turtle.hideturtle()
    turtle.done()

if __name__ == "__main__":
    main()
```

Лістинг результата після запиту в BingAI по фракталу Коха

```
import turtle

def draw_koch_curve(turtle, length, order):
    if order == 0:
        turtle.forward(length)
    else:
        for angle in [60, -120, 60, 0]:
            draw_koch_curve(turtle, length / 3, order - 1)
            turtle.left(angle)

def draw_koch_snowflake(turtle, length, order):
```

```
    for _ in range(3):
        draw_koch_curve(turtle, length, order)
        turtle.right(120)

window = turtle.Screen()
koch_turtle = turtle.Turtle()
koch_turtle.speed(0)

# Move the turtle to start position
koch_turtle.up()
koch_turtle.goto(-150, 90)
koch_turtle.down()

draw_koch_snowflake(koch_turtle, 300, 3)

# Hide the turtle and show the result
koch_turtle.hideturtle()
window.mainloop()
```

Лістинг результата після запиту в Claude 1по фракталу Коха

```
import turtle

def koch_curve(order, size):
  if order == 0:
    turtle.forward(size)
  else:
    koch_curve(order-1, size/3)
    turtle.left(60)
    koch_curve(order-1, size/3)
    turtle.right(120)
    koch_curve(order-1, size/3)
    turtle.left(60)
    koch_curve(order-1, size/3)

turtle.speed(0)
turtle.penup()
turtle.goto(-200,0)
turtle.pendown()
koch_curve(3, 600)
turtle.done()
```

Лістинг результата після запиту в Perplexity по фракталу Коха

```
import turtle

turtle.setup(width=600, height=600)
turtle.speed(0)

def koch_curve(length, depth):
    if depth == 0:
        turtle.forward(length)
    else:
```

```
        koch_curve(length / 3, depth - 1)
        turtle.left(60)
        koch_curve(length / 3, depth - 1)
        turtle.right(120)
        koch_curve(length / 3, depth - 1)
        turtle.left(60)
        koch_curve(length / 3, depth - 1)

koch_curve(200, 3)

turtle.done()
```

Лістинг коду тесту ChatGPT по методу Гауса

```
import timeit  # Вимірювання часу виконання1
import time  # Вимірювання часу виконання2
import cProfile  # Профілювання коду1
from line_profiler import LineProfiler  # Профілювання коду2
from memory_profiler import profile  # Використання пам'яті
profiler = LineProfiler()

start_time = time.time()
from GaussChatGPT import gauss_elimination, matrix  # ChatGPT
end_time = time.time()

print("\nТести")
execution_timeit = timeit.timeit(lambda: gauss_elimination(matrix))
print(f"\nЧас виконання timeit: {execution_timeit} секунд")
execution_time = end_time - start_time
print(f"\nЧас виконання time: {execution_time} секунд\n")
@profile #Використання пам'яті
def profile_gauss_elimination(matrix):
    gauss_elimination(matrix)
profile_gauss_elimination(matrix)
# Викликати функцію з профілюванням
profiler.add_function(gauss_elimination)
profiler.enable_by_count()
gauss_elimination(matrix)
profiler.disable_by_count()
# Вивести статистику профілювання
profiler.print_stats()
print("\nПрофіль коду cProfile")
cProfile.run('gauss_elimination(matrix)')
```

Лістинг коду тесту BingAI по методу Гауса

```
import timeit  # Вимірювання часу виконання1
import time  # Вимірювання часу виконання2
import cProfile  # Профілювання коду1
from line_profiler import LineProfiler  # Профілювання коду2
from memory_profiler import profile  # Використання пам'яті
profiler = LineProfiler()
```

```
start_time = time.time()
from GaussBing import gauss_elimination, matrix # Bing
end_time = time.time()

print("\nТести")
execution_timeit = timeit.timeit(lambda: gauss_elimination(matrix))
print(f"\nЧас виконання timeit: {execution_timeit} секунд")
execution_time = end_time - start_time
print(f"\nЧас виконання time: {execution_time} секунд\n")
@profile #Використання пам'яті
def profile_gauss_elimination(matrix):
    gauss_elimination(matrix)
profile_gauss_elimination(matrix)
# Викликати функцію з профілюванням
profiler.add_function(gauss_elimination)
profiler.enable_by_count()
gauss_elimination(matrix)
profiler.disable_by_count()
# Вивести статистику профілювання
profiler.print_stats()
print("\nПрофіль коду cProfile")
cProfile.run('gauss_elimination(matrix)')
```

Лістинг коду тесту Claude 1 по методу Гауса

```
import timeit  # Вимірювання часу виконання1
import time  # Вимірювання часу виконання2
import cProfile  # Профілювання коду1
from line_profiler import LineProfiler  # Профілювання коду2
from memory_profiler import profile  # Використання пам'яті
profiler = LineProfiler()

start_time = time.time()
from GaussClaude import gauss_elimination, A,b #Claude 1.2
end_time = time.time()

print("\nТести")
execution_timeit = timeit.timeit(lambda: gauss_elimination(A.copy(), b.copy()))
print(f"\nЧас виконання timeit: {execution_timeit} секунд")
execution_time = end_time - start_time
print(f"\nЧас виконання time: {execution_time} секунд\n")
@profile #Використання пам'яті
def profile_gauss_elimination(A,b):
    gauss_elimination(A.copy(), b.copy())
profile_gauss_elimination(A,b)
# Викликати функцію з профілюванням
profiler.add_function(gauss_elimination)
profiler.enable_by_count()
gauss_elimination(A.copy(), b.copy())
profiler.disable_by_count()
# Вивести статистику профілювання
profiler.print_stats()
print("\nПрофіль коду cProfile")
```

```
cProfile.run('gauss_elimination(A.copy(), b.copy())')
```

Лістинг коду тесту Perplexity по методу Гауса

```
import timeit  # Вимірювання часу виконання1
import time  # Вимірювання часу виконання2
import cProfile  # Профілювання коду1
from line_profiler import LineProfiler  # Профілювання коду2
from memory_profiler import profile  # Використання пам'яті
profiler = LineProfiler()

start_time = time.time()
from GaussPerplexity import gauss_elimination,A,b #Perplexity
end_time = time.time()

print("\nТести")
execution_timeit = timeit.timeit(lambda: gauss_elimination(A.copy(), b.copy()))
print(f"\nЧас виконання timeit: {execution_timeit} секунд")
execution_time = end_time - start_time
print(f"\nЧас виконання time: {execution_time} секунд\n")
@profile #Використання пам'яті
def profile_gauss_elimination(A,b):
    gauss_elimination(A.copy(), b.copy())
profile_gauss_elimination(A,b)
# Викликати функцію з профілюванням
profiler.add_function(gauss_elimination)
profiler.enable_by_count()
gauss_elimination(A.copy(), b.copy())
profiler.disable_by_count()
# Вивести статистику профілювання
profiler.print_stats()
print("\nПрофіль коду cProfile")
cProfile.run('gauss_elimination(A.copy(), b.copy())')
```

Лістинг коду тесту ChatGPT по методу золотого перетину

```
import timeit  # Вимірювання часу виконання1
import time  # Вимірювання часу виконання2
import cProfile  # Профілювання коду1
from line_profiler import LineProfiler  # Профілювання коду2
from memory_profiler import profile  # Використання пам'яті
profiler = LineProfiler()

start_time = time.time()
from GoldenRatioChatGPT import golden_ratio_search, test_function, a, b # Золотий перетин ChatGPT
end_time = time.time()

print("\nТести")
execution_timeit = timeit.timeit(lambda: golden_ratio_search(test_function, a, b))
print(f"\nЧас виконання timeit: {execution_timeit} секунд")
execution_time = end_time - start_time
print(f"\nЧас виконання time: {execution_time} секунд\n")
```

```
@profile #Використання пам'яті
def profile_golden_ratio_search(test_function, a, b):
    golden_ratio_search(test_function, a, b)
profile_golden_ratio_search(test_function, a, b)
# Викликати функцію з профілюванням
profiler.add_function(golden_ratio_search)
profiler.enable_by_count()
golden_ratio_search(test_function, a, b)
profiler.disable_by_count()
# Вивести статистику профілювання
profiler.print_stats()
print("\nПрофіль коду cProfile")
cProfile.run('golden_ratio_search(test_function, a, b)')
```

Лістинг коду тесту BingAI по методу золотого перетину

```
import timeit  # Вимірювання часу виконання1
import time  # Вимірювання часу виконання2
import cProfile  # Профілювання коду1
from line_profiler import LineProfiler  # Профілювання коду2
from memory_profiler import profile  # Використання пам'яті
profiler = LineProfiler()

start_time = time.time()
from GoldenRatioBing import golden_ratio_search, f, a, b # Золотий перетин Bing
end_time = time.time()

print("\nТести")
execution_timeit = timeit.timeit(lambda: golden_ratio_search(f, a, b))
print(f"\nЧас виконання timeit: {execution_timeit} секунд")
execution_time = end_time - start_time
print(f"\nЧас виконання time: {execution_time} секунд\n")
@profile #Використання пам'яті
def profile_golden_ratio_search(f, a, b):
    golden_ratio_search(f, a, b)
profile_golden_ratio_search(f, a, b)
# Викликати функцію з профілюванням
profiler.add_function(f)
profiler.enable_by_count()
golden_ratio_search(f, a, b)
profiler.disable_by_count()
# Вивести статистику профілювання
profiler.print_stats()
print("\nПрофіль коду cProfile")
cProfile.run('golden_ratio_search(f, a, b)')
```

Лістинг коду тесту Claude 1 по методу золотого перетину

```
import timeit  # Вимірювання часу виконання1
import time  # Вимірювання часу виконання2
import cProfile  # Профілювання коду1
from line_profiler import LineProfiler  # Профілювання коду2
from memory_profiler import profile  # Використання пам'яті
```

```
profiler = LineProfiler()

start_time = time.time()
from GoldenRatioClaude import golden_ratio_search, a, b, tol # Золотий перетин Claude
1.2
end_time = time.time()

print("\nТести")
execution_timeit = timeit.timeit(lambda: golden_ratio_search(a, b, tol))
print(f"\nЧас виконання timeit: {execution_timeit} секунд")
execution_time = end_time - start_time
print(f"\nЧас виконання time: {execution_time} секунд\n")
@profile #Використання пам'яті
def profile_golden_ratio_search(a, b, tol):
    golden_ratio_search(a, b, tol)
profile_golden_ratio_search(a, b, tol)
# Викликати функцію з профілюванням
profiler.add_function(golden_ratio_search)
profiler.enable_by_count()
golden_ratio_search(a, b, tol)
profiler.disable_by_count()
# Вивести статистику профілювання
profiler.print_stats()
print("\nПрофіль коду cProfile")
cProfile.run('golden_ratio_search(a, b, tol)')
```

Лістинг коду тесту Perplexity по методу золотого перетину

```
import timeit  # Вимірювання часу виконання1
import time  # Вимірювання часу виконання2
import cProfile  # Профілювання коду1
from line_profiler import LineProfiler  # Профілювання коду2
from memory_profiler import profile  # Використання пам'яті
profiler = LineProfiler()

start_time = time.time()
from GoldenRatioPerplexity import golden_ratio_search, f, a, b # Золотий перетин
Perplexity
end_time = time.time()

print("\nТести")
execution_timeit = timeit.timeit(lambda: golden_ratio_search(f, a, b))
print(f"\nЧас виконання timeit: {execution_timeit} секунд")
execution_time = end_time - start_time
print(f"\nЧас виконання time: {execution_time} секунд\n")
@profile #Використання пам'яті
def profile_golden_ratio_search(f, a, b):
    golden_ratio_search(f, a, b)
profile_golden_ratio_search(f, a, b)
# Викликати функцію з профілюванням
profiler.add_function(f)
profiler.enable_by_count()
golden_ratio_search(f, a, b)
```

```
profiler.disable_by_count()
# Вивести статистику профілювання
profiler.print_stats()
print("\nПрофіль коду cProfile")
cProfile.run('golden_ratio_search(f, a, b)')
```

Лістинг коду тесту ChatGPT по фракталу Коха

```
import turtle
import time
import cProfile
from line_profiler import LineProfiler
from memory_profiler import profile

# Функція для вимірювання часу виконання
def measure_time(func):
    start_time = time.time()
    func()
    end_time = time.time()
    print(f"Час виконання: {end_time - start_time} секунд")

# Функція для малювання фрактала Коха
def koch_snowflake(turtle, order, size):
    if order == 0:
        turtle.forward(size)
    else:
        for angle in [60, -120, 60, 0]:
            koch_snowflake(turtle, order - 1, size / 3)
            turtle.left(angle)

# Головна функція
@profile
def main():
    turtle.speed(0)
    turtle.penup()
    turtle.goto(-150, 90)
    turtle.pendown()

    order = 4  # Змініть цей параметр, щоб змінити рівень деталізації
    size = 300

    for _ in range(3):
        koch_snowflake(turtle, order, size)
        turtle.right(120)

    turtle.hideturtle()
    turtle.done()

if __name__ == "__main__":
    # Вимірюємо час виконання
    print("Вимірювання часу виконання:")
    measure_time(main)
```

```
    # Використовуємо cProfile
    print("\nВикористання cProfile:")
    cProfile.run("main()")

    # Використовуємо LineProfiler
    print("\nВикористання LineProfiler:")
    lp = LineProfiler()
    lp.add_function(koch_snowflake)
    lp_wrapper = lp(main)
    lp_wrapper()

    # Використовуємо memory_profiler
    print("\nВикористання memory_profiler:")
    main()
```

Лістинг коду тесту BingAI по фракталу Коха

```
import turtle
import time
import cProfile
from line_profiler import LineProfiler
from memory_profiler import profile

# Функція для вимірювання часу виконання
def measure_time(func):
    start_time = time.time()
    func()
    end_time = time.time()
    print(f"Час виконання: {end_time - start_time} секунд")

# Функція для малювання фрактала Коха
ef draw_koch_curve(turtle, length, order):
    if order == 0:
        turtle.forward(length)
    else:
        for angle in [60, -120, 60, 0]:
            draw_koch_curve(turtle, length / 3, order - 1)
            turtle.left(angle)

def draw_koch_snowflake(turtle, length, order):
    for _ in range(3):
        draw_koch_curve(turtle, length, order)
        turtle.right(120)

# Головна функція
@profile
window = turtle.Screen()
koch_turtle = turtle.Turtle()
koch_turtle.speed(0)

# Move the turtle to start position
koch_turtle.up()
koch_turtle.goto(-150, 90)
```

```
koch_turtle.down()

draw_koch_snowflake(koch_turtle, 300, 3)

# Hide the turtle and show the result
koch_turtle.hideturtle()
window.mainloop()

if __name__ == "__main__":
    # Вимірюємо час виконання
    print("Вимірювання часу виконання:")
    measure_time(main)

    # Використовуємо cProfile
    print("\nВикористання cProfile:")
    cProfile.run("main()")

    # Використовуємо LineProfiler
    print("\nВикористання LineProfiler:")
    lp = LineProfiler()
    lp.add_function(koch_curve)
    lp_wrapper = lp(main)
    lp_wrapper()

    # Використовуємо memory_profiler
    print("\nВикористання memory_profiler:")
    main()
```

Лістинг коду тесту Claude 1 по фракталу Коха

```
import turtle
import time
import cProfile
from line_profiler import LineProfiler
from memory_profiler import profile

# Функція для вимірювання часу виконання
def measure_time(func):
    start_time = time.time()
    func()
    end_time = time.time()
    print(f"Час виконання: {end_time - start_time} секунд")

# Функція для малювання фрактала Коха
def koch_curve(order, size):
  if order == 0:
    turtle.forward(size)
  else:
    koch_curve(order-1, size/3)
    turtle.left(60)
    koch_curve(order-1, size/3)
    turtle.right(120)
    koch_curve(order-1, size/3)
```

```
    turtle.left(60)
    koch_curve(order-1, size/3)

# Головна функція
@profile
def main():
    turtle.speed(0)
    turtle.penup()
    turtle.goto(-200, 0)
    turtle.pendown()
    koch_curve(3, 600)
    turtle.done()
if __name__ == "__main__":
    # Вимірюємо час виконання
    print("Вимірювання часу виконання:")
    measure_time(main)

    # Використовуємо cProfile
    print("\nВикористання cProfile:")
    cProfile.run("main()")

    # Використовуємо LineProfiler
    print("\nВикористання LineProfiler:")
    lp = LineProfiler()
    lp.add_function(koch_curve)
    lp_wrapper = lp(main)
    lp_wrapper()

    # Використовуємо memory_profiler
    print("\nВикористання memory_profiler:")
    main()
```

Лістинг коду тесту Perplexity по фракталу Коха

```
import turtle
import time
import cProfile
from line_profiler import LineProfiler
from memory_profiler import profile

# Функція для вимірювання часу виконання
def measure_time(func):
    start_time = time.time()
    func()
    end_time = time.time()
    print(f"Час виконання: {end_time - start_time} секунд")

# Функція для малювання фрактала Коха
def koch_curve(length, depth):
    if depth == 0:
        turtle.forward(length)
    else:
        koch_curve(length / 3, depth - 1)
```

```
        turtle.left(60)
        koch_curve(length / 3, depth - 1)
        turtle.right(120)
        koch_curve(length / 3, depth - 1)
        turtle.left(60)
        koch_curve(length / 3, depth - 1)

# Головна функція
@profile
def main():
    turtle.setup(width=600, height=600)
    turtle.speed(0)
    koch_curve(200, 3)
    turtle.done()

if __name__ == "__main__":
    # Вимірюємо час виконання
    print("Вимірювання часу виконання:")
    measure_time(main)

    # Використовуємо cProfile
    print("\nВикористання cProfile:")
    cProfile.run("main()")

    # Використовуємо LineProfiler
    print("\nВикористання LineProfiler:")
    lp = LineProfiler()
    lp.add_function(koch_curve)
    lp_wrapper = lp(main)
    lp_wrapper()

    # Використовуємо memory_profiler
    print("\nВикористання memory_profiler:")
    main()
```