Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Інститут інформаційної безпеки, радіоелектроніки та телекомунікацій
Кафедра кібербезпеки та програмного забезпечення

Степаненко Олександр,
студент групи РФ-181

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Розробка комбінованого методу дослідження функцій складної топології

Спеціальність:
122 Комп'ютерні науки

Спеціалізація, освітня програма:
Програмне забезпечення систем захисту інформації

Керівник:
Шаповалов Генадій Віталійович,
к.т.н., ст.викладач

Одеса – 2022

Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Інститут інформаційної безпеки, радіоелектроніки та телекомунікацій
Кафедра кібербезпеки та програмного забезпечення

Рівень вищої освіти перший (бакалаврський)
Спеціальність 122–Комп'ютерні науки
Освітня програма – Програмне забезпечення систем захисту інформації

ЗАТВЕРДЖУЮ
Завідувач кафедри КБПЗ

_____
д.т.н., проф. О.О.Кобозєва
____ _____202_р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Степаненку Олександру

1.Тема роботи: Розробка комбінованого методу дослідження функцій з складною топологією,

керівник роботи Шаповалов Геннадій Вітальєвіч, к. т. н., ст.в.,

затверджені наказом ректора від „____”_____ 20___ нар. №_____.

2.Зміст роботи: аналіз проблемної галузі, постановка задачі, аналіз принципів побудови методів дослідження функцій, розробка комбінованого методу дослідження функцій з складною топологією, створення програмної реалізації розробленого методу, охорона праці.

4. Консультанти розділів роботи

Розділ	Прізвище, ініціалі та посада консультанта	Підпис, дата	
Охорона праці	*доц. Ярова І.А*	Завдання видав	Завдання прийняв

5. Дата видачі завдання "_____"_____2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Рядок виконання	Примітка
1	*Аналіз джерел з теми випускної кваліфікаційної роботи*	*15.11.2021*	*виконано*
2	*Обґрунтування вибору рішення. Збір даних*	*15-12-2021*	*виконано*
3	*Аналіз основних аспектів дослідження функцій*	*11-01-2022*	*виконано*
4	*Аналіз умов використання методів дослідження*	*20-02-2022*	*виконано*
5	*Розроблення власного комбінованого методу дослідження*	*30-03-2022*	*виконано*
6	*Підготовка тексту роботи*	*11-05-2022*	*виконано*
7	*Підготовка презентації та доповіді*	*6-06-2022*	*виконано*
8	*Попередній захист*	*7-06-2022*	*виконано*
9	*Нормоконтроль, рецензування*	*17-06-2022*	*виконано*

Здобувач вищої освіти _____ *Степаненко О.*

Керівник роботи _____ *Шаповалов Г.В.*

ЗАВДАННЯ

на розробку розділу “Охорона праці”

Степаненку Олександру, група РФ-181

Інститут інформаційної безпеки, радіоелектроніки та телекомунікацій

Кафедра кібербезпеки та програмного забезпечення

Тема роботи Розробка комбінованого методу дослідження функції складної топології

Зміст розділу:

1 Аналіз умов праці та вибір основних заходів виробничої безпеки.

2 Аналіз пожежної безпеки. Вибір заходів та засобів пожежної безпеки.

3 Індивідуальне завдання.

Керівник роботи _____ (*Шаповалов Г.В*) «____» _____ 2022р.

Консультант з охорони праці _____ (*доц. Ярова І.А*) «____» _____ 2022р.

АНОТАЦІЯ

Кваліфікаційна робота на тему “Розробка комбінованого методу дослідження функцій зі складною топологією” на здобуття першого (бакалаврського) рівня вищої освіти за спеціальністю 122–Комп'ютерні науки, спеціалізація, освітня програма: Програмне забезпечення систем захисту інформації (або Комп'ютерні науки та інформаційна безпека), містить 6 малюнків, 2 таблиці, 1 додаток, 19 літературних джерел за переліком посилань. Робота виконана на 50 сторінках загального тексту та 43 сторінках основного тексту.

Метою роботи є розробка комбінованого методу дослідження функцій зі складною топологією та програмна реалізація розробленого методу.

У роботі проведено аналіз основних існуючих методів дослідження функції у завданнях умовної оптимізації, що дозволяє створити власний метод.

В результаті виконання кваліфікаційної роботи розроблено метод дослідження функцій зі складною топологією. Проведено порівняння ефективності розробленого методу з існуючими алгоритмами дослідження функцій. Зроблено висновок, що за рахунок впровадження нового методу дослідження функцій вдалося підвищити швидкість дослідження на приблизно 2%, також вдалося зберегти точність розрахунків та запобігти значних втрат пам'яті.

Результати даної роботи можуть бути використані при розробці програмного забезпечення для безпілотної техніки, для проведення досліджень поверхні, для подальших модифікацій та створення більш складних проектів.

ФУКНЦІЇ, ФУНКЦІЇ ЗІ СКЛАДНОЮ ТОПОЛОГІЄЮ, ОПТИМІЗАЦІЯ, ЗАДАЧІ УМОВНОЇ ОПТИМІЗАЦІЇ, ДОСЛІДЖЕННЯ ФУНКЦІЙ, ПОШУК ЕКСТРЕМУМУ.

ANNOTATION

Qualification work on "Development of a combined method of research of functions with complex topology" for the first (bachelor's) level of higher education in the specialty 122 Computer Sciences, specialization, educational program: Software for information security systems (or Computer Science and Information security), contains 6 figures, 2 tables, 1 appendix, 19 references according to the list of references. The work is performed on 50 pages of general text and 43 pages of main text.

The aim of the work is to develop a combined method for studying functions with complex topology and software implementation of the developed method.

The analysis of the basic existing methods of research of function in problems of conditional optimization that allows to create own method is carried out in the work.

As a result of qualification work, a method for studying functions with complex topology was developed. The efficiency of the developed method is compared with the existing algorithms for the study of functions. It is concluded that due to the introduction of a new method of research functions it was possible to increase the speed of research by about 2%, also managed to maintain the accuracy of calculations and prevent significant memory loss.

The results of this work can be used in the development of software for unmanned vehicles, for surface studies, for further modifications and more complex projects.

FUNCTIONS, FUNCTIONS WITH COMPLEX TOPOLOGY, OPTIMIZATION, PROBLEMS OF CONDITIONAL OPTIMIZATION, RESEARCH OF FUNCTIONS, EXTREME SEARCH.

ЗМІСТ

ВСТУП

Функції відіграють важливу роль у існуванні всього людства. Будь-який процес чи явище в тому чи іншому вигляді може бути описаний у вигляді відповідної залежності. У цьому плані функції зі складною топологією є винятком. Саме вони дозволяють найточніше відтворювати та описувати поверхню різних об'єктів, у тому числі й планети Земля, а також окремих її ділянок. На практиці це означає можливість скласти придатну до адаптації для різних завдань модель поведінки у заданих або умовах, що змінюються. На сучасному розвитку технологій це є життєво важливим для функціонування більшості автоматизованих та безпілотних систем.

Актуальність теми дослідження. За довгу годину дослідження проблеми було створено безліч методів дослідження різних функцій, у тому числі і позначених зі складною топологією. Одними з найчастіше застосовуваних є методи математичної оптимізації. Пошук екстремуму функції дозволяє вирішувати безліч проблем, але обмеженість багатьох методів рамками конкретних завдань серйозно знижує можливості пошуку найефективнішого та найменш витратного способу вирішення. Саме суміщення кількох методів дозволить отримати результат з оптимальною витратою ресурсів, з прийнятними витратами за годиною та з найбільшою точністю.

Метою даної є розробка комбінованого методу дослідження функцій зі складною топологією, складання алгоритму та створення його програмної реалізації.

Для досягнення цієї мети в дипломній роботі поставлено такі завдання:

- розглянути існуючі методи дослідження функцій;
- розкрити сильні та слабкі сторони найпоширеніших та якісних методів;
- розглянути види завдань, для вирішення яких застосовують обрані методи;

– сформулювати основні висновки по роботі, а також винести та обґрунтувати пропозиції щодо створення комбінованого методу дослідження, що дозволити з найбільшою ефективністю вирішувати поставлені завдання.

– створити програмну реалізацію розробленого методу

Об'єкт дослідження: математична оптимізація. Предметом дослідження виступили методи математичної оптимізації завдань з обмеженнями.

Новизна дослідження полягає в тому, що у дипломній роботі на основі результатів аналізу існуючих методів оптимізації для завдань з обмеженнями розроблено комбінований метод дослідження функцій зі складною топологією, що дозволяє знаходити оптимальні рішення за прийнятне витрачання ресурсів.

Теоретичну основу дослідження складають положення та висновки, що містяться у працях вчених: Абакаров О. Ш., Сушков Ю. О., Хохлюк В. І., Максимов Ю. А., Філіповська Є. А., Гілл Ф., Мюррей У., Райт М. та ін.

Структура дипломної роботи обумовлена предметом, метою та завданнями дослідження. Робота складається з вступу, чотирьох розділів та укладання.

Во вступі показана актуальність для роботи, визначено мету дослідження та завдання для її досягнення, розкрито теоретичну та практичну значущість проекту.

У першому розділі дається загальна характеристика функцій зі складною топологією, методами математичної оптимізації для вирішення завдань, що описуються подібними функціями. У другому розділі більш детально розглядаються та аналізуються конкретні методи дослідження функцій, їх переваги та недоліки, ефективність застосування для вирішення завдань. Третій розділ присвячений розробці комбінованого методу дослідження функцій, опису дії, оцінці ефективності роботи створеного методу. У четвертому розділі розглядається питання охорони праці: умови роботи, шкідливі та небезпечні фактори, освітлення, мікроклімат, пожежна безпека, способи зниження ризиків під час роботи. Висновок показує результати роботи, визначає прогрес досягнення поставленої мети та загальні висновки з проведеного дослідження.

1 ФУНКЦІЙ ЗІ СКЛАДНОЮ ТОПОЛОГІЄЮ ТА ЇХ ЗАСТОСУВАННЯ

1.1 Функції зі складною топологією

До функцій зі складною топологією відносяться функції 2 і більше змінних. Графіками таких функцій зазвичай виступають будь-які поверхні.

При розгляді описаних раніше функцій слід пам'ятати про функції 2 і більше змінних. Застосування цих функцій значно розширює коло розв'язуваних завдань, проте породжуючи величезну кількість нових проблем[9]. Для вивчення будь-якої функції необхідно мати про неї уявлення, знати хоч деякі значення, точки, що належать їй, її властивості, на відміну від відомих чи простих представників цього класу математичних понять. У цілому нині функції багатьох змінних досить схожі за описом з функціями 1 змінної[12]. Відмінності однак у кількості невідомих, деяких властивостях та нюансах при проведенні дослідження даних та подібних до них функцій.

Для побудови графіків функцій з багатьма змінними використовують точки з 3 і більше координатами. Сукупність точок, що належать функції, дозволяє побудувати графік. Найчастіше графіком виступає якась поверхня[12]. На функції кількох змінних як і їх побратимів з однієї переносяться такі поняття, як межа, безперервність. Функція, безперервна у кожній точці певної області, називається безперервною у цій області. Межі

$$\lim_{\Delta x \to 0} \frac{\Delta_x z}{\Delta x} = \frac{\partial z}{\partial x} = f'_x(x, y), \qquad \lim_{\Delta y \to 0} \frac{\Delta_y z}{\Delta y} = \frac{\partial z}{\partial y} = f'_y(x, y),$$

якщо вони існують, називаються приватними похідними функції з декількома змінними за цими змінними [9].

Для обчислення приватних похідних функції з декількома змінними необхідно прийняти як константу одну або кілька змінних і вираховувати все як для функції з 1 змінною, що спрощує як розуміння процесу так і роботу з ним[11]. Правила диференціювання функцій однієї змінної так само як і відповідні формули можна використовувати для знаходження приватних похідних функції

декількох змінних. Часткові похідні другого порядку знаходять взяття похідних від приватних похідних першого порядку[11].

Для функцій кількох змінних характерні змішані похідні. Значення змішаних похідних у точках, де вони безперервні, рівні між собою[11].

Для функцій кількох змінних запроваджується поняття градієнта. Градієнтом Функції називають вектор, координати якого є приватними похідними функціями за її змінними[9].

Максимум і мінімум функції називаються її екстремумами, що відповідає функціям з 1 змінною.

Поняття критичних та екстремальних точок для функції з декількома змінними збережені у незмінному вигляді. Тому правила, що стосуються знаходження та існування як критичних точок так і екстремумів однакові як для простих функцій так і для функцій з декількома змінними[12].

1.2 Математична оптимізація: завдання, методи та види

Функції зі складною топологією часто використовуються при описі поверхонь. Такі завдання насамперед вирішуються за допомогою знаходження екстремуму функції (мінімум/максимум).

Математична оптимізація (або просто оптимізація) або математичне програмування - це перебір низки альтернативних рішень для пошуку кращого елемента відповідно до критеріїв пошуку. Завдання оптимізації характерні для всіх галузей науки і життя людства, постійно змінюються, доповнюються, оновлюються, ускладнюються та потребують нових рішень[12].

Оптимізація як розділ математики існує досить давно. Практично з самого зародження людської раси йшло нескінченне прагнення оптимізувати все, що оточувало людей. Це спричинило стрімкий прогрес цивілізації і дозволяло відроджуватися після занепадів. По суті, вся наука працює так чи інакше з оптимізацією, прагнучи покращити існування людей. Сама собою оптимізація є вибором. Вибором, що в тій чи іншій області робить кожна людина щодня і навіть

мить свого життя. Вибір їжі, шляхи до місця призначення, вибір одягу, реплік у діалозі – це оптимізація[13].

Проте всі рішення мають свій рівень важливості. Від одних залежить лише малозначуща дрібниця, інші здатні змінювати долю світу або великих груп людей. Ці рішення є набагато відповідальнішими. З огляду на складності явища наслідки кожного їх менш зрозумілі; для того, щоб уявити ці наслідки, потрібно провести розрахунки. А головне, від цих рішень значно більше залежить[13].

Найскладніше приймати рішення або робити вибір від якого залежить чимало людей. Найчастіше саме неможливість прораховувати довгограючі наслідки заважає робити цей вибір або веде до помилок, неправильних рішень[13]. Тому при цьому необхідно проводити складні розрахунки, намагатися передбачати наслідки вибору та дій, що можуть відігравати важливу роль.

Для більш швидкого та впевненого прийняття важливих рішень потрібна практика[11]. Однак саме вона породжує нові і нові завдання оптимізації, причому їх складність зростає. При цьому необхідно постійно враховувати нові і нові фактори, обставини, несподіванки. Потрібно постійно будувати нові або вдосконалювати наявні математичні моделі та методи, що дозволяють оптимізувати процеси.

Існують завдання оптимізації, для вирішення яких потрібні так звані «важкі» цільові функції: обчислення значення такої функції може вимагати величезних витрат за часом та обчислювальним ресурсам. Як приклад, відмінно підходить обчислення форми крила літака[10]. За таких обставин на прискорення лінійної алгебри в самому алгоритмі оптимізації найчастіше не звертають уваги через недоцільність подібного — прагнуть усіма способами зменшити саму функцію.

Додаткові проблеми можуть виникати у завданнях, у яких цільова функції обчислюється неточно, і з деяким шумом, що вносить невизначеність.

До найпростіших випадків існування задач оптимізації відноситься пошук екстремумів дійсної функції. Так званий процес мінімізації та максимізації. У більш загальному випадку просто потрібно вибрати найкраще з представленого

набору даних. Саме пошук екстремумів функції є сполучною ланкою між функціями зі складною топологією та їх застосуванням на практиці[10]. Саме оптимізація, дозволяє досліджувати описані раніше функції декількох змінних. У прикладній математиці, інженерії, географії, геології, геофізиці, повітроплаванні, у робототехніці та цілій низці інших областей можна застосовувати функції зі складною топологією. Адже насправді такі функції можуть добре описувати поверхню планети та її частин.

Завжди слід пам'ятати, що реальні завдання дуже складні. Як багато експериментів в лабораторіях або на папері не проводилося, завжди потрібно пам'ятати про необхідність практичного підтвердження досліджень. На даний момент існує досить багато методів оптимізації, які можна використовувати для дослідження функцій, але вони не ідеальні. Завдання, що можуть містити тисячі невідомих, потребують застосування методів, які можна насамперед проконтролювати, знизити ризики помилок та можливі похибки[9].

Слід зазначити, що під мінімізацією (максимізацією) функції n змінних f(x)=f(x1,..,xn) на заданій множині U n-мірного векторного простору En розуміється визначення хоча б однієї з точок мінімуму (максимуму) цієї функції на безлічі U, а також, якщо це необхідно, і мінімального (максимального) U значення f(x) [10].

Отже, основним завданням щодо функцій зі складною топологією слід вважати пошук екстремуму. Але при цьому варто також розглянути поняття задач з обмеженнями та без обмежень.

1.3 Умовна та безумовна оптимізація

Оптимізація є дуже широким та розмитим поняттям. Існує кілька видів її. До них відносяться умовна та безумовна оптимізація. Для того щоб краще зрозуміти ці терміни необхідно зрозуміти чим вони відмінні один від одного.

Завдання безумовної оптимізації формально дає можливість використовувати всю область визначення функції для пошуку потрібних значень.

З одного боку, це досить цікава перспектива, що стикається з реальністю при спробах застосувати отримані знання[10]. Подібні завдання цікаві власними силами і допомагають краще зрозуміти процеси дослідження функцій, але за умов реальності вони рідко застосовні. Завдання умовної оптимізації має на відміну від минулої певні обмеження щодо незалежних змінних на множині. Подібні обмеження зазвичай задаються у вигляді кількох нерівностей, що обмежують зону дії щодо функцій[9]. У цієї роботі насамперед розглянуті завдання умовної оптимізації.

Основна мета завдання умовної оптимізації – пошук екстремуму скалярної функції f(x). Для вирішення необхідно застосування лінійної або квадратичної апроксимації для визначення прирощень x1 ... xn на кожній ітерації. Завдяки методу шматково-лінійної апроксимації існує можливість вирішення нелінійних завдань [9].

Використання аналітичних методів для вирішення задач з обмеженнями не призводить до необхідного результату. Тому досить багато чисельних методів було розроблено, протестовано часом і безліччю вирішених завдань[13].

Методи умовної оптимізації, як правило, зводять рішення вихідного завдання до багаторазового вирішення допоміжного завдання безумовної оптимізації. При цьому відбувається пошук максимального (мінімального) значення задач без обмежень [11].

Хоча практично майже всі завдання мають якісь обмеження, завдання безумовної оптимізації, як і раніше, вкрай важливі. Багато алгоритмів розв'язання задачі з обмеженнями передбачають зведення її до послідовності задач безумовної оптимізації. Деякі методи вирішення пов'язані з напрямком мінімізації. Наукове обгрунтування завдань без обмежень можна використовуватиме пояснення і підкріплення завдань умовної оптимізації[11]. Часто це призводить до необхідності розвитку нових методів, просуваючи прогрес. Так при створенні роботи так само були розглянуті та вивчені деякі корисні методи безумовної оптимізації, що є додатковим напрямком для подальших досліджень на тему та можливістю для розвитку системи в

майбутньому.

Наступна оптимізація

$$\min(x-1)^2 \tag{1.1}$$

є безумовною і має рішення x=1, у той час як додавання нерівності, як показано далі

$$\begin{cases}\min(x-1)^2 \\ x \leq 0\end{cases} \tag{1.2}$$

робить оптимізацію умовною та породжує рішення x = 0 [13]

Саме так описуються у загальному вигляді всі завдання як з обмеженнями, так і без них. Однак дослідження безлічі вчених та практичне застосування показали можливість запису обмежень у вигляді нерівностей замість множини. У цьому як збережено сенс записи, а й створено умови легшого вирішення завдань оптимізації з обмеженнями.

$$g_i(x) \leq 0, 1 \leq i \leq m, \tag{1.3}$$
$$h_i(x) = 0, 1 \leq i \leq k.$$

Підбиваючи проміжні підсумки можна сміливо припустити важливість та актуальність обраної теми дослідження. Функції зі складною топологією пов'язані з безліччю практичних напрямів людської діяльності. Завдяки оптимізації можна не тільки проводити прості дослідження функцій, але також створити практичну реалізацію розробленого методу для застосування в будь-якій зі сфер життєдіяльності людини. Так цілком можливе додавання подібної програми як забезпечення для безпілотної техніки, при цьому продукт виконуватиме роль своєрідного навігатора для об'єкта. Також є можливість створення спеціального забезпечення для відстеження змін поверхні землі, картографування, моделювання та цілий ряд областей, що використовують функції, що досліджуються в роботі.

2. АНАЛІЗ МЕТОДІВ УМОВНОЇ ОПТИМІЗАЦІЇ

2.1 Апроксимація функції кількох змінних: спосіб найменших квадратів.

Окремим випадком оптимізаційних завдань є завдання апроксимації.

На основі отриманої інформації з проведеного аналізу найпопулярніших методів умовної оптимізації було створено комбінований метод дослідження функцій зі складною топологією. Розроблений метод включає отримання та обробку даних в табличному вигляді, побудова функції за допомогою апроксимації методом найменших квадратів, дослідження функції на екстремум за допомогою симплекс-метода і методу якнайшвидшого спуску.

Суть методу найменших квадратів полягає у знаходженні коефіцієнтів лінійної залежності, у яких функція двох змінних набуває найменшого значення. Сума квадратів відхилень експериментальних даних від знайденої прямої має бути найменшою [1].

У випадку системи, що розробляється, вкрай важливо отримати найближчу до реальності поверхню на основі наявних даних. Від цього залежить точність результатів [9]. На практиці ж подібне й зовсім критично. Застосування розробленої системи в безпілотних літальних апаратах дозволить з точністю відтворювати поверхню в зоні дії об'єкта та координувати рух, уникаючи ризику отримати критичні пошкодження або втратити об'єкт остаточно.

Застосування апроксимації шляхом найменших квадратів починається з отримання даних. У випадку практичної реалізації, дані наводяться безпосередньо з датчиків апарату (як приклад - камери, датчики руху, локатори). Отримані дані обробляються і передаються як таблиці, як і потрібно роботи програмного продукту.

Відразу після отримання даних система починає їх перевірку, обробку та передачу для проведення процесу апроксимації методом найменших квадратів. Вибір даного методу насамперед пов'язаний з його простотою та відпрацьованістю на безлічі реальних практичних завдань. Завдяки чому можна

зібрати вичерпні дані, що підтверджують висновки щодо надійності методу. Також варто згадати швидкодію методу, що для сучасної автоматизованої та роботизованої техніки є безперечним і важливим плюсом. Здатність швидко реагувати на навколишнє середовище, що змінюється, мабуть, є однією з головних переваг сучасної та майбутньої техніки.

Застосування апроксимації дозволяє сформувати картину місцевості для об'єкта та, використовуючи отримані дані проводити повноцінні коригування руху, розраховуючи шлях таким чином, щоб уникнути всіх можливих зіткнень та пошкоджень.

Застосування розробленого алгоритму на вирішення реальних завдань є дуже серйозним випробуванням будь-якого існуючого методу оптимізації. Якщо для теоретичних завдань часто досить невеликих масивів даних для проведення експериментів у лабораторних умовах, то вже при зіткненні з реальністю виникає потреба враховувати безліч факторів, що іноді можуть серйозно впливати на реальний об'єкт.

Так для безпілотних систем можна виділити вплив на них повітря, сили тяжіння, залежність від власного завантаження, наявність різних об'єктів на шляху апарата, чи то самі птахи, літальні апарати або інші об'єкти.

Саме ці фактори, здавалося б незначні і опускаються в лабораторних умовах, можуть серйозно вплинути на будь-яку систему, перевантажити її, викликати помилки, призвести до неточностей у роботі або завдати прямих втрат.

Облік таких факторів є однією з найважливіших проблем при вирішенні будь-якого завдання.

Адже чим більше змінних враховується, тим більше громіздкі обчислення потрібно проводити для уникнення проблем[12]. Це веде до серйозного збільшення часу реагування зміну обстановки і відповідно ставить під загрозу весь об'єкт.

Метод найменших квадратів, задіяний у роботі, дозволяє прораховувати поверхню в зоні дії об'єкта з достатньою точністю і швидкістю для уникнення серйозних загроз.

2.2 Симплекс-метод

Серед усіх методів оптимізації симплекс-метод по праву вважається найвідомішим[3]. Створений спеціально для лінійного програмування симплекс метод досягає точного рішення за кінець кількох кроків, проте не варто забувати про необхідність застосування точної арифметики для подібного, що на практиці абсолютно недосяжно[7].

Ідея симплекс-методу складається з двох частин:

Ми отримуємо системи лінійних нерівностей, які задають багатовимірні опуклі багатогранники (так для прикладу в одномірному випадку отримаємо точку, промінь або відрізок, у двовимірному – опуклий багатокутник, у тривимірному – опуклий багатогранник). Мінімізація заданої лінійної функції вимагає від нас знаходження найбільш «дальньої» точки в певному напрямку. Даною точкою є певна вершина в якій усі нерівності звернуться до рівності. Таких точок завжди кінцеве число, хоч їх і може бути дуже багато[7].

Маючи такий набір точок, ми тепер можемо скористатися найпростішим способом перебору всіх значень і знайти результат. Це досить простий, хоч і неефективний, але робочий метод, що вимагає при вирішенні реальних завдань неймовірних витрат обчислювальних ресурсів, пам'яті та часу. Симплекс-метод ж ігноруючи перебір всіх значень, рухається від вершини до вершини по ребрах з кожним кроком, покращуючи значення цільової функції. І коли виявляється, що у вершини немає «сусідів», у яких значення функції кращі, то вона є оптимальною[7].

Симплекс-метод є ітеративним, тобто він послідовно потрохи покращує рішення. Для таких методів потрібно з чогось починати, у загальному випадку це робиться за допомогою вирішення допоміжного завдання, де ми мінімізуємо функцію за певної заданої умови[7].

Симплекс-метод було обрано створення комбінованого методу з кількох причин. По-перше, цей метод є досить швидким і точним, що дозволяє йому

вирішувати безліч простих завдань самостійно і близько до істини, отримуючи достовірні результати. По-друге, цей метод не використовує диференціювання, що дозволяє йому досягати успіху з найменшими серед інших методів витратами пам'яті і часу для обчислень.

По суті симплекс-метод дозволяє задати основу для вирішення будь-якого завдання оптимізації з обмеженнями, яке швидкість і невибагливість дають йому відмінну можливість бути використаним у зв'язці з іншими методами без серйозних навантажень на систему[7]. Хоча аналізований метод все ж таки має ряд проблем у вигляді неточності при вирішенні задач з безліччю змінних та можливості зачіпки за локальний екстремум при дослідженні функції, це цілком компенсується його перевагами.

Використання симплекс-метода дозволяє досить швидко обробляти інформацію, отриману з датчиків та представлену у вигляді описаної поверхні за допомогою методу найменших квадратів. Швидкодія методу дозволяє досить швидко проводити дослідження поверхні та знаходити екстремальні значення, потім порівнювати їх та у результаті знаходити шукане значення. Відмінною можливістю використання даного методу буде, наприклад, пошук посадкових майданчиків безпілотних апаратів, простий обліт території.

2.3 Метод якнайшвидшого спуску

Незважаючи на очевидні переваги симплекс-метода, далеко не завжди вдасться використати його зі 100% точністю визначення потрібного значення. Завжди є ймовірність похибки в обчисленнях. А застосовність раніше розглянутого методу до вирішення простих завдань призводить до труднощів зі складними. Величезна кількість змінних застосовуваних у сучасних практичних завданнях, стрімке ускладнення, необхідність у постійному зростанні точності результатів веде до необхідності шукати та створювати нові рішення у сфері оптимізації.

Метод якнайшвидшого спуску є чудовим прикладом модифікації для

системи, де необхідна швидкість з точністю[1]. У проекті, що розробляється, саме даний метод на основі серії експериментів був визнаний найбільш вдалим варіантом для реалізації. Можливості аналізованого методу достатні для обчислення точних значень функції в екстремальних точках без втрати швидкості обчислень і без використання значних обсягів пам'яті.

З використанням методу якнайшвидшого спуску кожної ітерації величина кроку ak вибирається з умови мінімуму функції f(x) у бік спуску, т. е. f(x[k] – akf'(x[k])) = f(x[k] - af '(x [k])).min[4]

Таким чином, поки значення функції f(x) меншає відбувається рух по антиградієнту. Завдяки чому отримуємо, що з математичної точки зору на кожній ітерації потрібно лише вирішувати завдання одномірної мінімізації a функції j(a) = f(x[k] - af'(x[k])) [4].

Алгоритм методу якнайшвидшого спуску складається з наступних кроків:

1. Спершу необхідно задати координати початкової точки x[0].

2. У точці x[k], де k = 0, 1, 2, ... обчислюємо значення градієнта f'(x[k]).

3. Далі необхідно визначити величину кроку ak, шляхом одномірної мінімізації a функції j(a) = f(x[k] - af'(x[k])).

4. Після чого знаходимо координати точки x[k+1]:

$$xi[k+1] = xi[k] – akf'i(x[k]), i = 1,..., n.$$

5. Нарешті треба перевірити умови зупинки ітераційного процесу. У разі виконання припиняємо всі обчислення. Інакше повертаємось до першого пункту.

У аналізованому методі напрямок руху з точки x[k] стосується лінії рівня в точці x[k+1] (Рисунок2.1). Можна спостерігати зигзагоподібну траєкторію спуску та ортогональність ланок зигзагів один одному. Дійсно, крок ak отримуємо мінімізуючи a функцію f(a) = f(x[k] - af'(x[k])). Необхідна умова мінімуму функції dj(a)/da = 0. Обчисливши похідну складної функції, отримаємо умову ортогональності векторів напрямків спуску в сусідніх точках:

$dj(a)/da = -f'(x[k+1]f'(x[k]) = 0$[1].

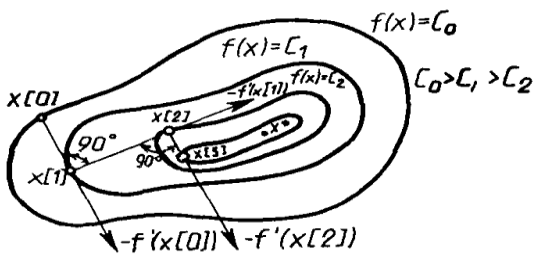

Рисунок 2.1 – Геометрична інтерпретація методу якнайшвидшого спуску

Незважаючи на отримані результати практично все далеко від ідеалу, мінімізовані функції досить часто мають погано зумовлені матриці других похідних (т/М << 1)[4]. Значення вздовж деяких напрямів починаю стрімко зміняться, іноді швидкість подібних змін відрізняється на кілька порядків. Поверхні рівня в подібних обставинах починають змінюватися або витягуватися в найпростіших випадках, або серйозно змінюватися в складних. Саме так з'являються звані яри. Функції, що мають такі властивості, називають яружними. Напрямок антиградієнта цих функцій (див. мал. 2.2) істотно відхиляється від напрямку в точку мінімуму, що призводить до уповільнення швидкості збіжності. І відповідно виникають помилки під час роботи методу, що у певних умовах практично згубно для використовуваного устаткування.

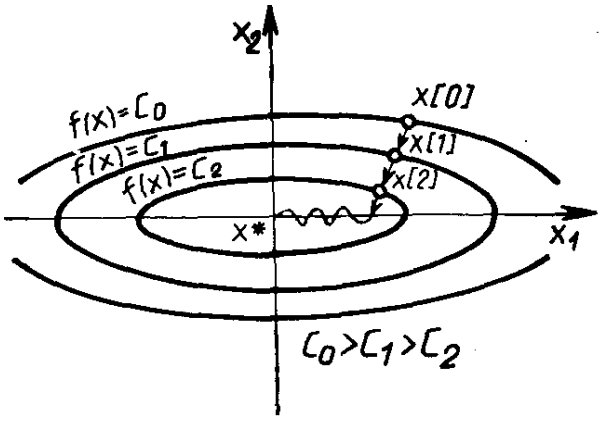

Рисунок2.2 – Чудова функція

Часто при застосуванні методу якнайшвидшого спуску спостерігається залежність швидкості збіжності від точності обчислень градієнта. Втрата точності, а це зазвичай відбувається в околиці точок мінімуму або в яру, і зовсім

порушує збіжність процесу градієнтного спуску. Саме вищезгадані причини вимагають використання методу у комбінації з іншими для гарантованого отримання результату.

На даний момент нами були розглянуті та проаналізовані деякі методи оптимізації функції, що включені до складу алгоритму для дослідження функцій зі складною топологією. Однак не варто забувати і про інші методи оптимізації, що також мають непогані характеристики для вирішення різноманітних завдань. Завжди необхідно пам'ятати про постійний розвиток аналізованої області, що неминуче веде до швидкого старіння вже наявних методів дослідження та потреби нових рішень, що забезпечать необхідний результат отримання результатів.

Також не варто забувати про деякі цікаві методи на кшталт методів штрафних функцій для завдань з обмеженнями. Аналіз подібних методів дозволяє зробити деякі висновки про їх обчислювальні властивості та можливості даних методів. Методи ведуть пошук рішення, не виходячи за межі допустимої області, що характерно як для внутрішніх, так і для зовнішніх функцій[11]. Це дуже важливо в тих випадках, коли цільова функція або обмеження не визначені за межами допустимої множини. Крім цього, є можливість завжди перервати обчислення і отримати при цьому допустиме рішення. Необхідність вирішувати завдання для визначення якоїсь початкової точки, що за складністю цілком суперничає з вихідним завданням нелінійного програмування є серйозним мінусом, що буквально подвоює обсяг роботи. Метод зовнішніх штрафних функцій забезпечує рішення з будь-якої початкової точки, вирішуючи подібним перебігом проблему. Це дозволяє спростити, причому значно програмну реалізацію алгоритмів. Загальним недоліком методів штрафних функцій є складність допоміжної функції F(x, a), яка має яру структуру. При цьому зі зростанням параметра зростає і ступінь яружності. Крім того, при великих значеннях точність обчислень мінімуму F(x, a) сильно зменшується через помилки округлення ЕОМ[10].

Методи внутрішніх штрафних функцій не застосовні для певних завдань нелінійного програмування з обмеженнями рівностями. Для вирішення подібних

питань використовують комбіновані алгоритми, що враховують особливості внутрішніх та зовнішніх штрафних функцій, що дозволяє застосовувати на практиці описані обмеження, необхідні в реальних завданнях і що вкотре доводить переваги комбінованих методів над одиночними. На превеликий жаль застосування методів штрафних функцій, у тому числі і комбінованих, неможливо в роботі, через їх несумісність з раніше розглянутими. Підвищення складності завдання та зниження результативності, прояв яружного ефекту на практиці, зростання витрат часу та обчислювальних потужностей на обчислення призвело до закономірної відмови від використання подібних методів.

Ознайомлення з іншими методами допомогло порівняти за такими ознаками як швидкодія, точність обчислень, витрати обчислювальних ресурсів, складність реалізації. Для подібного аналізу були взяті методи, застосовані в роботі та описані вище, методи штрафних функцій, прямі методи умовної оптимізації.

Їх використання на однаковій задачі дослідження польоту дрону з прокладання оптимального маршруту пройшло відносно успішно, що дозволило скласти думку про можливості розглянутих методів. Найкращий результат був продемонстрований складання методів обраних для реалізації в роботі, далі непоганий результат показали методи прямої оптимізації і на останньому місці виявилися методи штрафних функцій. Незважаючи на відносно невеликий розрив у рамках експерименту, отриманий результат наочно демонструє необхідність використання комбінацій методу для вирішення навіть теоретичних завдань, не кажучи вже про реальні приклади.

Проведення порівняльного аналізу всіх розглянутих методів оптимізації для вирішення задач з обмеженнями виявило також можливі варіанти для модифікації створеного методу дослідження функцій. Подальші дослідження з цієї теми допоможуть створити варіації проекту на вирішення більшого кола оптимізаційних завдань.

На основі проведеного аналізу всіх розглянутих методів для вирішення задач оптимізації з обмеженнями слід висновок про необхідність використання комбінованого методу для вирішення реальних практичних завдань, через ряд

обмежень деяких методів неможливо ефективно застосовувати їх окремо для знаходження екстремуму. Таким чином, лише об'єднання кількох методів дає найбільш точний, повний і відносно швидкий результат. Розглянутий і описаний у роботі варіант об'єднання методів симплексу та якнайшвидшого спуску дозволяє досить швидко та ефективно вирішувати як прості так і більш наближені до реальності завдання, що стоять перед фахівцями. Також створені намітки для модифікації розробленого методу на вирішення складніших і специфічних завдань.

3. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ КОМБІНОВАНОГО МЕТОДУ ДОСЛІДЖЕННЯ ФУНКЦІЙ СКЛАДНОЇ ТОПОЛОГІЇ.

3.1 Середовище програмування

У процесі проведення дослідження, пошуку інформації на тему, аналізу методів оптимізації для вирішення задач з обмеженнями, проектування власного методу дослідження функцій зі складною топологією постало питання про реалізацію власного програмного продукту, що не тільки зможе забезпечити працездатність створеного алгоритму, але й матиме можливість подальшого покращення , з накопиченням нової інформації на тему та усвідомленням наявної.

Для програмної реалізації було вибрано мову програмування Python. Ця мова була взята з цілої низки важливих причин. Однією з основних його багатофункціональність.

Саме в Python було реалізовано багато корисних бібліотек інших мов програмування. Отримавши можливості таких гігантів як C++/C#/C, Java, MatLab, Octave, Maxima, Fortan та інших, Python став справді універсальним засобом створення практично будь-якого проекту від простих калькуляторів до потужних нейронних мереж[2].

Також ця мова залучає досить зрозумілим синтаксисом, якісною документацією за своїми можливостями, вільним доступом для всіх бажаючих.
Робота виконувалася в IDE PyScripter – середовищі розробки мови Python. Версія мови 3.8.

Для створення програмної реалізації комбінованого алгоритму були взяті кілька бібліотек numpy для створення модуля обробки даних та обчислень, matplotlib.pyplot для графічного відображення отриманих результатів, mpl_toolkits.mplot3d для графічного відображення отриманих результатів у тривимірному форматі, scipy.optimize для створення модуля tkinter створення графічного інтерфейсу програмного продукту.

3.2 Реалізація модуля отримання, обробки та перетворення даних

Першою, і, мабуть, не менш важливою, ніж інші, частиною програми є модуль, що може отримати дані від датчиків у табличному форматі, перетворити їх у зручний для мови програмування формат і використовуючи описану раніше апроксимацію методом найменших квадратів побудувати поверхню, яка буде досліджуватися в надалі.

Спеціально для цього використовується бібліотека numpy. Оскільки мова програмування, що розглядається, інтерпретується, на відміну від тих же Сі і Матлаба[2], то існують серйозні обмеження при реалізації проектів. Їхня швидкість роботи занадто сповільнена, та й самі програми виходять досить громіздкими. Однак numpy у свою чергу представляє реалізації обчислювальних алгоритмів (у вигляді функцій та операторів), оптимізованих для обробки багатовимірних масивів. В результаті можемо отримати можливість реалізувати будь-який алгоритм, пов'язаний із роботою над масивами чи матрицями з адекватною швидкістю роботи.

Для створення необхідного в нашій роботі модуля отримання, обробки та перетворення даних необхідно розглянути його можливості окремо.

Так від початку нам необхідно отримати набір даних для подальшої роботи. Для роботи апроксимації методом найменших квадратів потрібно використовувати дані, що надаються в табличному вигляді, тоді програма, отримавши потрібні значення, перетворює їх у масив за допомогою бібліотеки numpy і зможе приступити до наступного кроку, а саме до апроксимації даних методом найменших квадратів.

Отримати дані з таблиці можна багатьма способами. У мові Python існують такі варіанти: отримати дані з файлу, для чого нам потрібно підключити додаткову бібліотеку для роботи з подібними файлами. Також можна налаштувати введення значень з клавіатури, що у разі нерентабельно через специфіку завдання. Є можливість ввести дані безпосередньо в програму, що цілком можливо, але сильно обмежує маневри при проведенні експериментів. І є можливість створити генератор псевдовипадкових чисел для генерації масиву

потрібних нам значень. Цей спосіб є найбільш корисним при проведенні експериментів.

Далі для зручності обчислень використовується варіант із генератором псевдовипадкових послідовностей чисел для експериментів.

Продемонструємо зазначений варіант у вигляді програмної реалізації:

```
X = np.arange(-2,2.2,0.1)
Y = np.arange(-3,3,0.1)
```

У наведеному коді показана ініціалізація 2 масивів з присвоєнням їм групи значень у діапазоні вказаному на місці перших двох значень та з кроком функції 0.1, що дозволить отримати досить значну кількість значень експерименту.

Що ж до реалізації самої апроксимації для отримання потрібної поверхні, то ми наводимо наступний код:

```
def lest_square_met(event):
mx = x.sum()/N
my = y.sum()/N
a2 = np.dot(xT, x)/N
a11 = np.dot(xT, y)/N
kk = (a11 - mx*my)/(a2 - mx**2)
bb = my - kk * mx
ff = np.array([kk*z+bb for z in range(N)])
```

Саме в цій функції відбувається обчислення необхідних значень на основі отриманих даних та за їх допомогою згодом будується поверхня для подальшого дослідження функції. У ході експериментів при проведенні дослідження виникла потреба у використанні інструментів побудови графіків для більш наочного представлення отриманих даних. У підсумковій версії програми ця можливість заблокована за відсутністю необхідності в ній.

3.3 Реалізація модуля пошуку оптимального рішення

Тепер, мабуть, настав час перейти до основного модуля проекту. Саме в даному модулі відбувається дослідження функції зі складною топологією та

пошук мінімального значення.

Для застосування подібної операції як ідеальний інструмент підійде бібліотека scipy.optimize, а саме її модуль minimize, що зберігає в собі цілий ряд реалізованих метод математичної оптимізації для завдань як з обмеженнями так і без обмежень. Також необхідний модуль linprog, що дозволить використовувати інструменти лінійного програмування у нашій програмі.

Модуль scipy був створений як розвиток numpy, розглянутого в попередньому розділі. Цей інструмент дає користувачеві набір можливостей серед розробки SciLab і MatLab. І дозволяє реалізувати більшість проектів, які раніше для мови недоступні[2].

Для нашого завдання необхідно використовувати отримані раніше дані апроксимації методом найменших квадратів. Отримати їх можна кількома способами за допомогою глобальних змінних або створення окремої функції передачі даних. Також можна скористатися інтерфейсом для відображення значень, зчитування та отримання даних, однак цей спосіб вкрай ненадійний і громіздкий. Тому для програми було обрано варіацію з використанням функції.

Далі необхідно розпочати дослідження функції за допомогою симплекс-методу. Для більш наочного показу результату слід використовувати графічне відображення результатів.

Наведемо програмну реалізацію симплекс-методу:

```
def Simplex(event):
x0 = np.array([0.7, 7.7, 4.4, 1.1])
res = minimize(rosenFun, x0, method='nelder-mead',
options={'xtol': 1e-8, 'disp': True})
print(res.x)
```

Оскільки бібліотека вже має цілу низку реалізованих методів оптимізації, то можна використовувати їх для нашого завдання.

Далі наведемо програмну реалізацію виведення результатів у графічному вигляді:

```
fig = plt.figure(figsize=[9, 7])
ax = fig.gca(projection='3d')
```

```
a=e1.get()
a1=e11.get()
b=e2.get()
b1=e22.get()
c=e3.get()
a = int(a)
a1 = int(a1)
b = int(b)
b1 = int(b1)
c = float(c)
# Задаємо кут огляду
ax.view_init(45, 30)
# Створюємо дані для графіка
X = np.arange(a, a1, c)
Y = np.arange(b, b1, c)
X, Y = np.meshgrid(X, Y)
Z = rosenFun(np.array([X,Y]))
ax.scatter(res.x[0], res.x[1], Z[-1], c='blue')
```

У даному фрагменті коду в першу чергу визначається діапазон значень для малювання поверхні. Це досить важливий етап для дослідження, оскільки ці дані отримані за допомогою апроксимації в попередньому розділі. Саме від правильності даних безпосередньо залежить поведінка об'єктів, які використовують у реальних завданнях створений метод дослідження функцій. Будь-яка помилка в побудові може призвести до закономірного кінця у вигляді пошкодження та знищення.

При використанні методу якнайшвидшого спуску необхідно враховувати ряд особливостей.

Наведемо програмну реалізацію методу якнайшвидшого спуску:

```
def descent(best_estimates, is_x):
derivative = derivative_x if is_x else derivative_y
```

```
best_x, best_y = best_estimates
descent_step = step
value = derivative(best_y, best_x)
while abs(value) > global_epsilon:
descent_step *= 0.95
best_y = best_y - descent_step \
if derivative(best_y, best_x) > 0 else best_y +
descent_step
value = derivative(best_y, best_x)
return best_y, best_x
def find_minimum():
return
descent(descent(pick_estimates(calculate_flip_points()
), False), True)
```

У наведеній реалізації показано 2 методи. Перший і найважливіший їх відповідає за реалізацію самого методу якнайшвидшого спуску для вирішення поставленого завдання. Другий вирішує саму задачу, знаходячи потрібне значення екстремуму.

Також для наочної демонстрації отриманих результатів слід використовувати графічне відображення отриманих значень.

Для цього наведемо програмну реалізацію:

```
def get_grid(grid_step):
samples = np.arange(-radius, radius, grid_step)
x, y = np.meshgrid(samples, samples)
return x, y, differentiable_function(x, y)
def draw_chart(point, grid):
point_x, point_y, point_z = point
grid_x, grid_y, grid_z = grid
plot.rcParams.update({
'figure.figsize': (4, 4),
```

```
'figure.dpi': 200,
'xtick.labelsize': 4,
'ytick.labelsize': 4
})
ax = plot.figure().add_subplot(111, projection='3d')
ax.scatter(point_x, point_y, point_z, color='red')
ax.plot_surface(grid_x, grid_y, grid_z, rstride=5,
cstride=5, alpha=0.7)
plot.show()
```

Наведений код розділений для зручності на окремі модулі дозволяє побудувати графічне відображення досліджуваної поверхні та зазначає отримане значення мінімуму на ньому. На малюнку 3.1 показано вигляд подібного відображення:

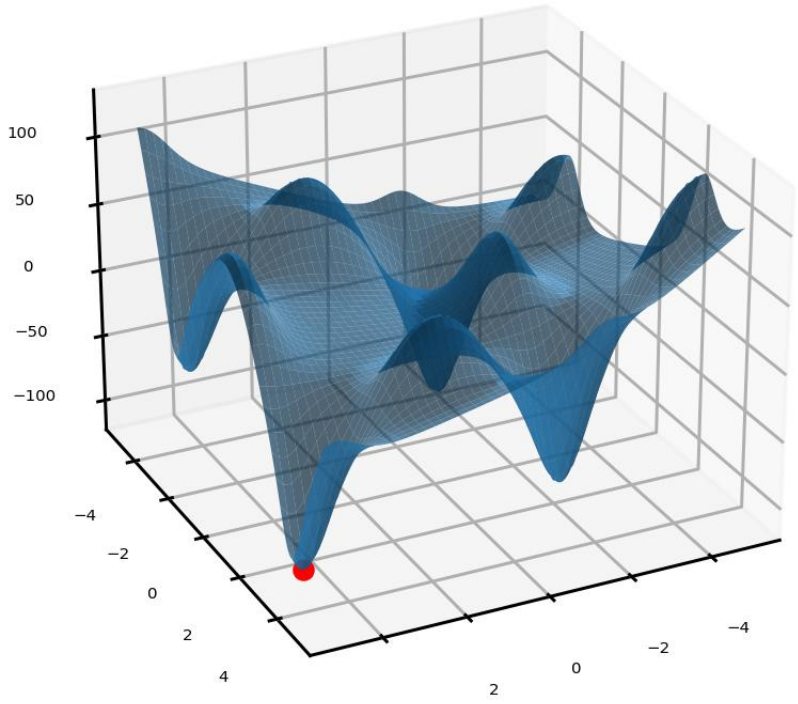

Рисунок 3.1 – Відображення результатів роботи методів якнайшвидшого спуску та симплекс-метода.

Для перевірки ефективності створеної програмі було проведено тестування із використанням декількох функції, що описують поверхню землі.

Так були взяти наступні функції:

```
def differentiable_function(x, y):
    return np.sin(x) + np.exp((1 - np.cos(y)) ** 2) + \
           np.cos(y) + np.exp((1 - np.sin(x)) ** 2) + (x -
y) ** 2
```

така функція описує гористу поверхню

```
def differentiable_function(x, y):
    np.sin(x) * np.exp((1 - np.cos(y)) ** 2) + \
           np.cos(y) * np.exp((1 - np.sin(x)) ** 2) + (x -
y) ** 2
```

така функція описує холмисту поверхню

після проведення дослідження були отримані наступні результати:

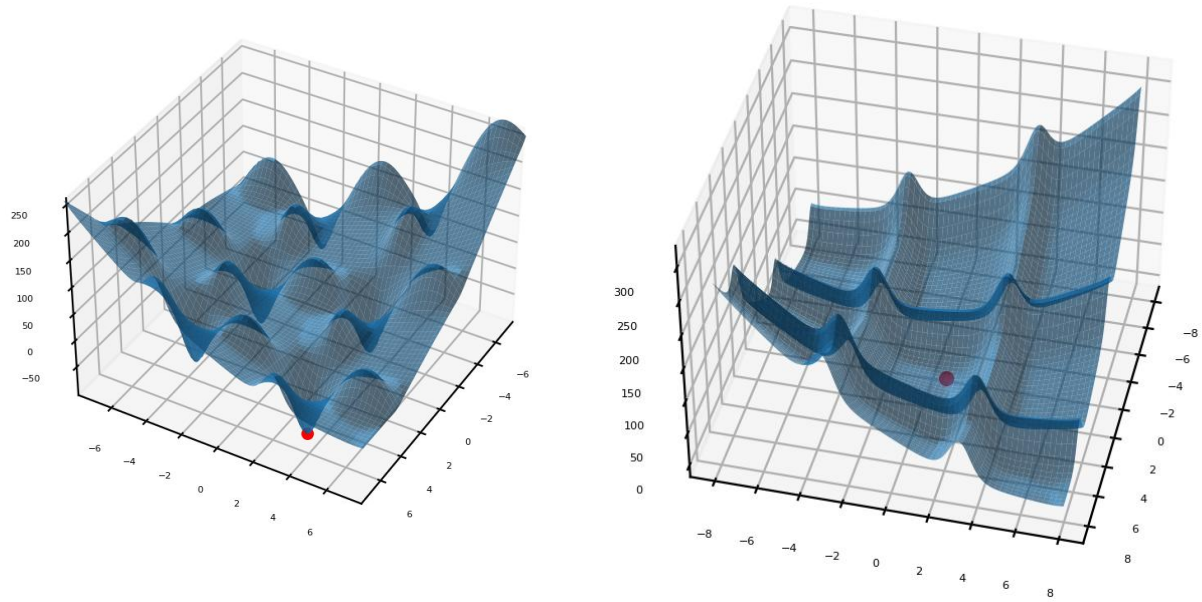

Рисунок3.2 Обрані для тестування функції, що описують поверхню

Для порівняння результатів були проведені аналогічні дослідження одиночними методами(симплекс та найскорішого спуску). Наприкінці отримані результати дали змогу скоротити кількість ітерацій для дослідження(387 та 296 для комбінованого методу, більш 500 для одного симплекс-методу в обох випадках, 466 та 390 для методу якнайскорішого спуску). Результати тестування показують перевагу розробленого комбінованого методу над одиночними.

Саме таким чином побудована програма вирішує поставлені завдання із

наочною демонстрацією результату.

3.4 Інтерфейс системи

Незважаючи на здавалося б повне виконання поставлених раніше завдань, необхідно пам'ятати, що продукт, що розробляється, в першу чергу повинен створюватися з орієнтуванням на інших користувачів, а не на одного лише програміста-розробника. Для подібних цілей часто створюється інтерфейс користувача, що дає можливість кожному, хто використовує створений продукт, зрозуміти його призначення і успішно застосовувати для вирішення різних завдань.

У Python для реалізації інтерфейсу можна використовувати модуль tkinter. Саме він містить цілу низку корисних інструментів для створення та налаштування власного інтерфейсу для програми.

Так для розробленої системи було створено наступний інтерфейс:

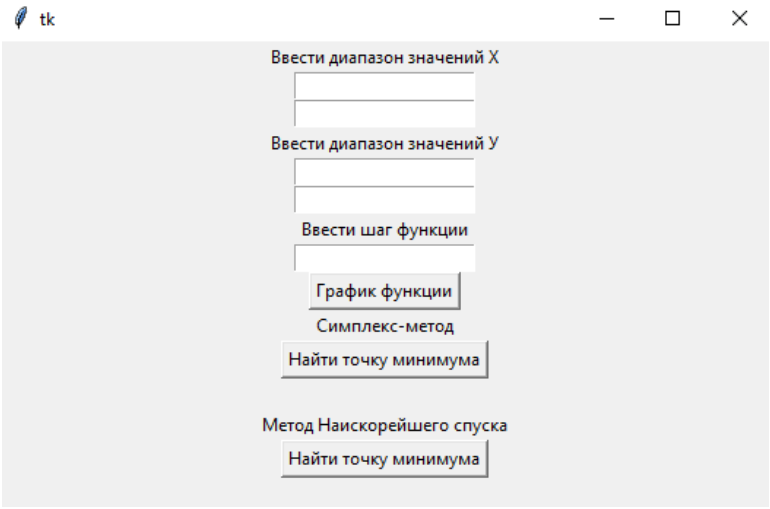

Рисунок3.3 Інтерфейс програми користувача

Насамперед слід зазначити необхідність простоти інтерфейсу для швидшого розуміння його різними користувачами. Також обов'язковими умовами є зрозумілість і часто важливу роль відіграє дизайн інтерфейсу. У створеному продукті інтерфейс досить простий та зручний. Перша з позначених кнопок дозволяє отримати дані в табличному форматі, перетворити їх на масиви та використовувати апроксимацію методом найменших квадратів для побудови досліджуваної поверхні. Наступні 2 кнопки пов'язані з модулем оптимізації та

дозволяють вивести результати роботи методів у вигляді графіків.

Наведемо програмну реалізацію інтерфейсу користувача:

```
from tkinter import *
root = Tk ()
l1= Label(root, width=75, text ='Ввести діапазон
значень X')
e1 = Entry()
e11 = Entry()
l2= Label(root, width=50, text ='Ввести діапазон
значень У')
e2 = Entry()
e22 = Entry()
l3= Label(root, width=50, text ='Ввести крок функції')
e3 = Entry()
b1 = Button (root, text = 'Графік функції')
l4= Label(root, width=50, text ='Симплекс-метод')
b2= Button(root, text='Знайти точку мінімуму')
l5 = Label (root)
l6= Label(root, width=50, text ='Метод Найшвидшого
спуску')
b3= Button(root, text='Знайти точку мінімуму')
l7 = Label (root)
l1.pack()
e1.pack()
e11.pack()
l2.pack()
e2.pack()
e22.pack()
l3.pack()
e3.pack()
```

```
b1.pack()
l4.pack()
b2.pack()
l5.pack()
l6.pack()
b3.pack()
l7.pack()
root.mainloop()
```

Як результат варто відзначити що на основі виконаної раніше дослідницької, аналітичної та практичної роботи здатність створити метод дослідження функцій зі складною топологією, що дозволяє проводити дослідження функцій, знаходити екстремуми, виводити значення у зручному для сприйняття графічному форматі. Створений інтерфейс користувача досить зручний і простий у використанні. Таким чином, можна вважати виконаними всі завдання, поставлені на початку дослідження.

4. ОХОРОНА ПРАЦІ

Аналіз умов праці та вибір основних заходів виробничої безпеки.

При виконанні будь-якої діяльності людина повинна враховувати важливість вирішення питань з охорони праці.

Адекватна оцінка всіх небезпечних та шкідливих факторів, що можуть впливати на людину та погіршувати стан здоров'я дуже важлива для підвищення якості умов праці та збереження комфортного життя.

В даному розділі дипломної роботи розглядується місце Python-розробника, що створює програмний продукт, який реалізує комбінований метод дослідження функції із складною топологією.

Для аналізу умов праці необхідно провести огляд робочого місця програмісту та описати вже існуючи умови. Після чого потрібно розробити рекомендації для покращення стану.

Приміщення, де розташоване робоче місце розробника знаходиться у ново побудованому будинку близько до центру міста. Приблизно в 1 кварталі знаходяться зупинка міського електротранспорту, у самого офісу є власний паркінг для працівників. Наявність описаних об'єктів є важливою для забезпечення комфортних засобів прибуття на роботу працівників. Приміщення, де знаходиться робоче місце розробника, розташовано на другому поверсі будівлі. Будівля має сучасне планування, що відповідає всім нормам забезпечення комфорту та захисту працівників. Площа приміщення з робочим місцем розробника - 20 м^2, висота – 3 м, об'єм – 60 м^3. Відповідно до нормативних документів площа одного робочого місця програмісту повинна бути не менше ніж за 6м^2, а об'єм не менший за 20м^3 [14], в приміщенні постійно працюють 3 програміста, що відповідає нормам.

Кожне робоче місце має стіл з площею поверхні – 1.6 м^2, крісло Атлетик Пластик-М Neapol 20, на столі знаходиться персональний комп'ютер, що має монітор Samsung S22C450MW 22'', системний блок Acer Veriton X2631G SFF, клавіатуру RZTK KB 310 Backlit USB, миш Logitech B100 USB Black. В

приміщенні також знаходяться принтер Canon Pixma MG2550S, кондиціонер Ergo AC-0708CH.

У приміщенні є 2 вікна розміром 1000х1600 мм. У кожного вікна є відкидна половина, що може бути в 3 режимах: відкрито, закрито, відкрито для провітрювання. Джерелами штучного освітлення стали світлодіодні лампи LED FLF-92 36W NW. 6 розеток з напругою 220 В, кабель Інтернет-з'єднання, кабель локальної мережі підприємства, маршрутизатор TP-Link TL-WR841N.

Під час виконання проектів на розробників можуть впливати деякі шкідливі небезпечні фізичні та психофізичні фактори згідно до :

- підвищена або знижена температура повітря робочої зони,
- підвищена чи знижена вологість повітря,
- недостатня освітленість робочого місця,
- підвищений рівень шуму на робочому місці,
- фізичні перевантаження (одноманітна поза викликає статичну втому),
- нервово-психічні перевантаження(розумове перенапруга, перенапруга аналізаторів).

Для запобігання або максимального зниження впливу вказаних факторів рекомендовано нормувати робочий день таким чим, щоб були можливі перерви для простих фізичних вправ, також потрібно проводити контроль якості меблів у приміщенні.

Для створення комфортних умов праці необхідно обов'язково враховувати стан мікроклімату у приміщенні. Використовуючи наведені нормативні документи та враховуючи енерговитрати, визначаємо приналежність роботи програміста до категорії легких робіт Іа, Іб. Згідно з ДСН 3.3.6.042-99 повинні бути встановлені наступні умови: для холодного періоду року оптимальна температура повітря – 22-24 ° С (21-23 ° С), оптимальна відносна вологість – 40-60%, швидкість руху повітря - 0,1 м/с; для теплого періоду року оптимальна температура повітря – 23-25 ° С (22-24 ° С), оптимальна відносна вологість – 40-60%, швидкість руху повітря - 0,1 м/с (0,2 м/с).

Порушення мікрокліматичних умов може привести до захворювання

працівників, падіння ефективності виконання проектів. Тому постійно потрібно підтримувати комфортні умови в приміщенні, для чого використовують водяне опалення у холодний період року та кондиціонер у теплий період. Регулювання параметрів мікроклімату виконується з використанням кондиціонеру, який підтримує як додаток до опалення у холодний період року та охолоджує у теплий.

Освітлення робочого місця є одним з найважливіших показників. Робота програміста напряму пов’язана з напругою зору. Тому при відхиленнях освітлення від норми з’являється недостача або надлишковість світу у приміщенні, що веде до проблем з зором та до зниження ефективності працівника.

Коефіцієнт природного освітлення (КПО) згідно є нормованим параметром для природного освітлення. КПО залежить від розряду зорових робіт, що виконуються працівником. Роботу розробника відносять до середньої точності (IV розряд зорових робіт, мінімальний розмір об'єкту розрізнення складає 0,5-1,0мм), при цьому КПО має бути 1,5% при використанні бокового освітлення. Для штучного освітлення нормованим параметром виступає $E_{мін}$ – мінімальний рівень освітленості, та $К_п$ – коефіцієнт пульсації світлового потоку, який не повинен бути більшим ніж 20%. Мінімальна освітленість встановлюється залежно від розряду виконуваних зорових робіт. Для IV розряду зорових робіт вона становить 300-500 лк.

При проведенні дослідження умов праці знайдено, що всі встановлені в приміщенні комп’ютери(3 одиниці) задають шум, що впливає на стан людини у приміщенні. Кожен з пристроїв має декілька елементів, що відповідають за створення шумів (на приклад вентилятори, жорсткі диски). Підвищувати рівень шуму також можуть периферійні пристрої, до яких відносять принтери, сканери. Допустимий еквівалентний рівень шуму для робочого місця програміста складає 50 дБА, але наявність багатьох джерел утворювання шуму приводить до перевищення нормованого значення, що погіршує умови праці.

Для захисту працівників від впливу шуму на стан здоров’я потрібно встановлювати додаткову ізоляцію(перегородки) та забезпечити всіх працівників персональними навушниками. Також є рекомендація змінити обладнання в

приміщенні, що не створює додатковий шум.

Приміщення, якому належить робоче місце розробника - сухе, без пилу, з нормальною температурою повітря, ізольованими підлогами та малим числом заземлених приладів, тому за небезпекою ураження електричним струмом воно належить до 1 класу.

До металевих деталей відносяться корпуси системних блоків комп’ютерів, деякі частини стільців, засоби нагріву приміщення.

Основні причини ураження людини електричним струмом на робочому місці: доторкання до металевих елементів (корпусу, периферії комп'ютера), що можуть накопичувати заряд та випускати його внаслідок пошкодження ізоляції або інших обставин; нерегламентоване використання електричних приладів; відсутність інструктажу співробітників з правил електробезпеки.

Пожежна безпека

Згідно з НАПБ Б.03.002-2007 визначивши призначення будівлі, кількість поверхів можна встановити ступінь вогнестійкості будинків, категорії по вибухопожежній та пожежній небезпеці.

Будівля із робочим місцем програмісту належить до категорії В. Це забезпечує наявність як легкозаймистих так і важкогорючих речовин. Так різний папір, меблі, документація можуть горіти без вибуху. Те саме стосується обладнання у поміщенні[17].

За конструктивними характеристиками будівлю можна віднести до будинків з несучими та огороджуючими конструкціями із природних або штучних кам'яних матеріалів, бетону або залізобетону, де для перекриття допускається використання дерев'яних конструкцій, захищених штукатуркою або важкогорючими листовими, а також плитними матеріалами.

Пожежа у робочому приміщенні, де робить Python-розробник може привести до втрати матеріальних та інформаційних ресурсів, також ніколи не вдасться виключити ймовірність гибелі людини внаслідок полум’я або диму. Тому необхідно визначити всі можливі причини виникнення пожежі для розробки методів протидії.

Причинами виникнення пожежі можуть бути: несправності електропроводки, розеток та вимикачів які можуть призвести до короткого замикання або пробою ізоляції; використання пошкоджених (несправних) електроприладів; використання у приміщенні електронагрівальних приладів з відкритими нагрівальними елементами; виникнення пожежі внаслідок влучення блискавки в будинок; загоряння будинку внаслідок зовнішніх впливів; неакуратне поводження з вогнем і недотримання мір пожежної безпеки.

Для запобігання пожежі у будинку потрібно правильно визначити всі можливі причини виникнення пожежі та розробити засоби боротьби з ними. В ідеальному стані необхідно повністю виключити всі можливі джерела виникнення пожежі або максимально забезпечити безпеку працівників.

У приміщенні з робочим місцем Python-розробника є декілька основних джерел виникнення пожежі. Найбільш ймовірними є проблеми з електроустаткуванням. Так при пошкодженні розташованих в офісі кабелів, при поломках комп'ютерів та периферійних пристроїв, при недостатньої ізоляції можуть з'являтися іскри електричного струму, що з великою імовірністю приводять до пожежі. Запобігти цього можливо при постійних перевірках обладнання у приміщенні, при проведенні інструктажів з техніки безпеки, при своєчасної заміні старих пристроїв на більш безпечні та нові.

До виникнення пожежі може привести використання різних електричних пристроїв не призначених для офісу. Наприклад, електричні чайники, мікрохвильові печі, різне додаткове обладнання для взаємодій з комп'ютерами. Підключення зазначених предметів приводить до перенавантаження електромережі, що підвищує ймовірність займання. Для запобігання цього потрібно виділити окреме приміщення для всіх подібних пристроїв, забороняти використання близько до робочого місця та перевіряти всі приміщення на наявність їх. Також необхідно проводити періодичні зустрічі, на яких до працівників доносять причини неможливості використання.

Слід також зазначити в якості ймовірних чинників пожежі несправне обладнання для нагрівання та короткі замикання. Знизити ризики таких явищ

можливо при постійному інспектуванні всього обладнання у офісі, при своєчасної зміні старих або пошкоджених елементів.

Блискавки можуть також привести к виникненню пожежі при ударі по будівлі, але використання блискавковідводу допоможе запобігти цього.

Наприкінці, слід нагадати, що більшість всіх пожеж виникає внаслідок недотримання людиною техніки безпеки, щодо використання легкозаймистих матеріалів та речовин або паління у приміщенні. Постійне нагадування про небезпеку такої поведінки, заборона паління у будівлі допоможуть запобігти важких наслідків.

В заданої будівлі можуть відбуватися пожежі **категорій А** (присутність у приміщенні паперу, документації, меблів з дерева), **В** (синтетичні матеріали так чи інакше присутні у багатьох приміщеннях), також можуть відбутися пожежі категорій **Е** (все обладнання у приміщенні використовує електричний струм для роботи, що закономірно підіймає ймовірність виникнення пожежі).

Тип вогнегасника потрібно вибирати, виходячи з особливостей конкретного об’єкта. Якщо на об’єкті можливі осередки пожеж різних класів, то слід вибирати вогнегасники окремо для кожного класу пожежі або віддавати перевагу більш універсальному вогнегаснику щодо області застосування.

Відстань між місцями розташування вогнегасників не повинна перевищувати: 15 м – для приміщень категорій А, Б, В; 20 м – для приміщень категорій В, Г, а також для громадських будівель та споруд[20].

На основі наведених даних визначимо тип та необхідну кількість вогнегасників:

Площа приміщення – 20; Категорія пожеж: А, В, Е; Тому можемо зробити висновки про необхідність використання порошкових вогнегасників з масою заряду вогнегасної речовини 5 кг і більше у кількості 1 одиниця. Всього виходить 1 порошковий вогнегасник з масою заряду вогнегасної речовини 3 кг.

Порошкові вогнегасники є найбільш універсальними для гасіння пожеж різних категорій, вони мають досить широкий діапазон температур експлуатації, принаймні для боротьби із пожежами всіх категорій. Для приміщення з робочим

місцем Python-розробника слід використовувати порошкові вогнегасники ВП-3, ВП-4 в 1 екземплярі.

ВИСНОВКИ

На закінчення слід зазначити низку моментів. На самому початку проведеного дослідження була мета - створити комбінований метод дослідження функцій зі складною топологією і створити його програмну реалізацію, яка наочно повинна була продемонструвати ефективність даного методу над вже існуючими аналогами.

У процесі роботи над темою проекту було вивчено численні матеріали, що стосуються досліджуваної та суміжних із нею тем. Розглянуто різні завдання оптимізації як з обмеженнями так і без, розглянуто схожості та відмінності, методи вирішення таких завдань. Досліджено та проаналізовано ефективність методів дослідження функцій. Проведено порівняння, на основі якого визначено найбільш оптимальні методи вирішення задач. З таких варто відзначити симплекс-метод, спосіб якнайшвидшого спуску, способи прямої оптимізації, способи штрафних функцій.

На основі отриманих даних розроблений і описаний комбінований метод дослідження функцій, що включає апроксимацію даних, одержуваних з датчиків, методом найменших квадратів, подальше дослідження за допомогою досить швидкого симплекс-метода і більш точне рішення за допомогою методу якнайшвидшого спуску. Проведене порівняння роботи методу дозволило підвищити точність одержуваного результату приблизно на 2%, а також знизити ризики утворення яру та наступного за нею провалу в пошуку рішень.

Створення програмної реалізації розробленого методу стало фінальним штрихом у виконання роботи. Отриманий інтерфейс дозволяє користувачам із зручністю використовувати можливості програмного продукту, порівнювати ефективність роботи методів і отримувати результати в наочному графічному вигляді. Проведене тестування з використанням функцій, що описують поверхню

дає змогу оцінити переваги розробленого методу над вже існуючими, що використовують роздільно. Також при використанні реальних прикладів слід зазначити підвищення точності приблизно на 2 % відносно одинарних методів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Мальковський Н. Огляд градієнтних методів у задачах математичної оптимізації URL:https://habr.com/ru/post/413853/
2. Метод BFGS або один із найефективніших методів оптимізації. URL:https://habr.com/ru/post/333356/
3. Лабінцев А. SciPy, оптимізація URL:https://habr.com/ru/post/439288/
4. Універсальний градієнтний спуск URL: http://docplayer.com/69224601-Universalnyy-gradientnyy-spusk.html
5. Shewchuk Jr. An Introduction to Conjugate Gradient Method Without the Agonizing Pain URL:https://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf
6. Optimization Methods. URL:https://mech.iitm.ac.in/nspch52.pdf
7. Nelder J.A, Mead R.A Simplex method for function minimization. URL:https://people.duke.edu/~hpgavin/cee201/Nelder+Mead-ComputerJournal-1965.pdf
8. Нові методи збереження optimization і comparison with other methods URL:https://watermark.silverchair.com/8-1-42.pdf?token=AQECAHi208BE49Ooan9kkhW_Ercy7Dm3ZL_9Cf3qfKAc485ysgAAAr8wggK7BgkqhkiG9w0BBwagggKsMIICqAIBADCCAqEGCSqGSIb3DQEHATAeBglghkgBZQMEAS4wEQQMPZy3Xfyh3ymdf6f9AgEQgIICchQSFf_Obe0Sa-p8WZ
9. Математична оптимізація URL: https://hmong.ru/wiki/Optimization_(mathematics)
10. Математична оптимізація. Методи оптимізації (вступний курс) URL:http://staff.ulsu.ru/semushin/

11. Трифонов А.Г. Постановка задачі оптимізації та чисельні методи її вирішенняhttps://hub.exponenta.ru/post/postanovka-zadachi-optimizatsii-i-chislennye-metody-ee-resheniya356#2_2_1
12. Мальківський Н. Огляд основних методів математичної оптимізації для завдань з обмеженнями URL:https://habr.com/ru/post/428794/
13. Лозовський А. Огляд методів чисельної оптимізації. Безумовна оптимізація: метод ліній URL:https://habr.com/ru/post/561128/
14. НПАОП 0.00-6.23-92. Порядок проведення атестації робочих місць за умовами праці
15. ГН 3.3.5-8-6.6.1 2002. Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу
16. НПАОП 0.00-1.28-10. Правила охорони праці під час експлуатації електронно-обчислювальних машин.
17. НАПБ Б.03.002-2007. Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою.
18. НАПБ Б.03.001-2004. Типові норми належності вогнегасників.
19. НАПБ А.01.001-2004. Правила пожежної безпеки в Україні.

Додаток А. Лістинг програмного продукту

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
з matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np
from scipy.optimize import minimize
from tkinter import *

root = Tk ()
l1= Label(root, width=75, text ='Ввести діапазон значень X')
e1 = Entry()
e11 = Entry()
l2= Label(root, width=50, text ='Ввести діапазон значень У')
e2 = Entry()
e22 = Entry()
l3= Label(root, width=50, text ='Ввести крок функції')
e3 = Entry()
b1 = Button (root, text = 'Графік функції')
l4= Label(root, width=50, text ='Симплекс-метод')
b2= Button(root, text='Знайти точку мінімуму')
l5 = Label (root)
l6= Label(root, width=50, text ='Метод Найшвидшого спуску')
b3= Button(root, text='Знайти точку мінімуму')
l7 = Label (root)
l8= Label(root, width=50, text ='Алгоритм Бройдена-Флетчера-
Голдфарба-Шанно')
b4= Button(root, text='Знайти точку мінімуму')
l9 = Label (root)

def graphFun(event):
# Налаштовуємо 3D графік
fig = plt.figure(figsize=[9, 7])
```

```
ax = fig.gca(projection='3d')
a=e1.get()
a1=e11.get()
b=e2.get()
b1=e22.get()
c=e3.get()
a = int(a)
a1 = int(a1)
b = int(b)
b1 = int(b1)
c = float(c)
# Задаємо кут огляду
ax.view_init(45, 30)

# Створюємо дані для графіка
X = np.arange(a, a1, c)
Y = np.arange(b, b1, c)
X, Y = np.meshgrid(X, Y)
Z = rosenFun(np.array([X,Y]))
# Малюємо поверхню
ax.scatter(X, Y, Z, c='blue')
#surf = ax.plot_surface(X, Y, Z, cmap=cm.Reds)
plt.show()

b1.bind('<Button-1>', graphFun)

##Симплекс метод Нелдера Міда
def SimplexNM(event):
x0 = np.array([0.7, 7.7, 4.4, 1.1])
res = minimize(rosenFun, x0, method='nelder-mead',
options={'xtol': 1e-8, 'disp': True})
print(res.x)
l5['text']=res.x

fig = plt.figure(figsize=[9, 7])
```

```
ax = fig.gca(projection='3d')
a=e1.get()
a1=e11.get()
b=e2.get()
b1=e22.get()
c=e3.get()
a = int(a)
a1 = int(a1)
b = int(b)
b1 = int(b1)
c = float(c)
# Задаємо кут огляду
ax.view_init(45, 30)

# Створюємо дані для графіка
X = np.arange(a, a1, c)
Y = np.arange(b, b1, c)
X, Y = np.meshgrid(X, Y)
Z = rosenFun(np.array([X,Y]))
ax.scatter(res.x[0], res.x[1], Z[-1], c='blue')

ax.scatter(x0[0], x0[1], x0[2], c='violet')
ax.scatter(res.x[0], res.x[1], res.x[2], c='red')
# Малюємо поверхню
surf = ax.plot_surface(X, Y, Z, cmap=cm.Reds)
plt.show()

b2.bind('<Button-1>', SimplexNM)

##метод Пауела
def powellMethod(event):
x0 = np.array([0.7, 7.7, 2.4, 1.1])
res = minimize(rosenFun, x0, method='powell',
options={'xtol': 1e-8, 'disp': True})
print(res.x)
```

```
l7['text']=res.x

fig = plt.figure(figsize=[9, 7])
ax = fig.gca(projection='3d')
a=e1.get()
a1=e11.get()
b=e2.get()
b1=e22.get()
c=e3.get()
a = int(a)
a1 = int(a1)
b = int(b)
b1 = int(b1)
c = float(c)
# Задаємо кут огляду
ax.view_init(45, 30)

# Створюємо дані для графіка
X = np.arange(a, a1, c)
Y = np.arange(b, b1, c)
X, Y = np.meshgrid(X, Y)
Z = rosenFun(np.array([X,Y]))
ax.scatter(x0[0], x0[1], x0[2], c='violet')
ax.scatter(res.x[0], res.x[1], res.x[2], c='red')
# Малюємо поверхню
#ax.scatter(X, Y, Z, c='blue')

surf = ax.plot_surface(X, Y, Z, cmap=cm.Greys)
plt.show()

b3.bind('<Button-1>', powellMethod)

def differentiable_function(x, y):
return np.sin(x) * np.exp((1 - np.cos(y)) ** 2) + \
np.cos(y) * np.exp((1 - np.sin(x)) ** 2) + (x - y) ** 2
```

```
def rotate_vector(length, a):
return length * np.cos(a), length * np.sin(a)
def derivative_x(epsilon, arg):
return (differentiable_function(global_epsilon + epsilon, arg) -
differentiable_function(epsilon, arg)) / global_epsilon
def derivative_y(epsilon, arg):
return (differentiable_function(arg, epsilon + global_epsilon) -
differentiable_function(arg, epsilon)) / global_epsilon
def calculate_flip_points():
flip_points = np.array([0, 0])
points = np.zeros((360, arr_shape), dtype=bool)
cx, cy = центр
for i in range(arr_shape):
for alpha in range(360):
x, y = rotate_vector(step, alpha)
x = x * i + cx
y = y * i + cy
points[alpha][i] = derivative_x(x, y) + derivative_y(y, x) > 0
if not points[alpha][i - 1] and points[alpha][i]:
flip_points = np.vstack((flip_points, np.array([alpha, i - 1])))
return flip_points
def pick_estimates(positions):
vx, vy = rotate_vector(step, positions[1][0])
cx, cy = центр
best_x, best_y = cx + vx * positions[1][1], cy + vy *
positions[1][1]
for index in range(2, len(positions)):
vx, vy = rotate_vector(step, positions[index][0])
x, y = cx + vx * positions [index] [1], cy + vy * positions [index]
[1]
if differentiable_function(best_x, best_y) >
differentiable_function(x, y):
best_x = x
best_y = y
for index in range(360):
```

```
vx, vy = rotate_vector(step, index)
x, y = cx + vx * (arr_shape - 1), cy + vy * (arr_shape - 1)
if differentiable_function(best_x, best_y) >
differentiable_function(x, y):
best_x = x
best_y = y
return best_x, best_y
def gradient_descent(best_estimates, is_x):
derivative = derivative_x if is_x else derivative_y
best_x, best_y = best_estimates
descent_step = step
value = derivative(best_y, best_x)
while abs(value) > global_epsilon:
descent_step *= 0.95
best_y = best_y - descent_step \
if derivative(best_y, best_x) > 0 else best_y + descent_step
value = derivative(best_y, best_x)
return best_y, best_x
def find_minimum():
return
gradient_descent(gradient_descent(pick_estimates(calculate_flip_poin
ts()), False), True)
def get_grid(grid_step):
samples = np.arange(-radius, radius, grid_step)
x, y = np.meshgrid(samples, samples)
return x, y, differentiable_function(x, y)
def draw_chart(point, grid):
point_x, point_y, point_z = point
grid_x, grid_y, grid_z = grid
plot.rcParams.update({
'figure.figsize': (4, 4),
'figure.dpi': 200,
'xtick.labelsize': 4,
'ytick.labelsize': 4
})
```

```
ax = plot.figure().add_subplot(111, projection='3d')
ax.scatter(point_x, point_y, point_z, color='red')
ax.plot_surface(grid_x, grid_y, grid_z, rstride=5, cstride=5,
alpha=0.7)
plot.show()
l1.pack()
e1.pack()
e11.pack()
l2.pack()
e2.pack()
e22.pack()
l3.pack()
e3.pack()
b1.pack()
l4.pack()
b2.pack()
l5.pack()
l6.pack()
b3.pack()
l7.pack()
##l8.pack()
##b4.pack()
##l9.pack()
root.mainloop()
```