МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

ФАКУЛЬТЕТ МЕХАТРОНІКИ ТА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Експериментальне дослідження методів недиференційної оптимізації

Рівень вищої освіти другий (магістерський)
Спеціальність 122 комп'ютерні науки
Освітня програма комп'ютерні науки

Виконав: студент групи МгІТ-1-23

Плотніков О . О

Науковий керівник к.т.н., доц. Корогод Г. О.

Рецензент д.т.н., проф. Чупринка В. І.

Київ 2024

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет мехатроніки та комп'ютерних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти другий (магістерський)
Спеціальність 122 Комп'ютерні науки
Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувачка кафедри КН
_____ Н.В.Чупринка

«_____» _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Плотнікову Олександру Олександровичу

1. Тема роботи Експериментальне дослідження методів недиференційної оптимізації.
Науковий керівник роботи Корогод Ганна Олександрівна, к. т .н., доц. затверджений наказом вищого навчального закладу 3. 09.2024 року. № 188-уч.
2. Вихідні дані до кваліфікаційної роботи Розробка кафедри комп'ютерних наук. Математичні методи дослідження керуючих рішень. Задачі керування інформаційними ресурсами. Принципи технології Domain-driven design проектування та реалізації програмних засобів.
3. Зміст дипломної роботи (перелік питань, які потрібно розробити) : РОЗДІЛ 1. Задача та методи дослідження; РОЗДІЛ 2. Проєктування та обґрунтування необхідних моделей і методів; РОЗДІЛ 3. Обґрунтування застосованих програмних технологій розробки; Висновки; Джерела використаної літератури; Додатки.
4. Дата видачі завдання 10.9.2024 р

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної магістерської роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	5.10.2023	
2	Розділ 1. Задача та методи дослідження	5.10.2024	
3	Розділ 2. Проєктування та обґрунтування необхідних моделей і методів	5.10.2024	
4	Розділ 3. Обґрунтування застосованих програмних технологій розробки.	10.10.2024	
5	Висновки	25.10.2024	
6	Оформлення дипломної магістерської роботи (чистовий варіант)	30.10.2024	
7	Подача кваліфікаційної роботи (проекту) науковому керівнику для відгуку (за 14 днів до захисту)	4.11.2024	
8	Здача дипломної магістерської роботи на кафедру для рецензування (за 14 днів до захисту)	6.11.2024	
9	Перевірка дипломної магістерської роботи на наявність ознак плагіату (за 10 днів до захисту)	8.11.2024	
10	Подання дипломної магістерської роботи на затвердження завідувачу кафедри (з 7 днів до захисту)	14.11.2024.	

З завданням ознайомлений:

Студент _____ **Олександр Плотніков**
(підпис)

Науковий керівник роботи _____ **Ганна Корогод**
(підпис)

АНОТАЦІЯ

Плотніков О. О. Експериментальне дослідження методів недиференційної оптимізації.

Дипломна магістерська робота за спеціальністю 122 комп'ютерні науки та інформаційні технології – Київський національний університет технологій та дизайну, Київ, 2024 рік.

В дипломній роботі описані теоретичні аспекти та питання практичної реалізації програмних засобів для демонстрації роботи та обґрунтованого вибору алгоритмів, що можуть бути застосовані для задач недиференційної оптимізації. Продемонстрована можливість застосування нерелаксаційних алгоритмів. Нерелаксаційні алгоритми є єдиним можливим методом дослідження задач, що сформулюються у вигляді задач нелінійного програмування з допомогою функцій, що не мають похідних в будь яких точках. Можливості нерелаксаційних алгоритмів демонструються з допомогою візуалізації програмних компонентів, що дозволяють спів ставити ліній рівня функцій та графічне відображення ітерації алгоритму. Програмне забезпечення дозволяє зрозуміло демонструвати та досліджувати роботу нерелаксаційних алгоритмів оптимізації. Програмний засіб може бути використаний в якості дидактичного інструменту. Можливості графічного порівняння особливостей роботи нерелаксаційних алгоритмів на тлі ліній рівня типових функцій робить аналіз зручним та наочним.

Ключові слова: Методи пошуку, алгоритми безумовної оптимізації першого порядку, субградієнтні методи.

ANNOTATION

Plotnikov O.O. Experimental study of methods of non -differential optimization.

Master's Degree in the specialty 122 Computer Science and Information Technologies - Kyiv National University of Technology and Design, Kyiv, 2024.

The thesis describes the theoretical aspects and issues of practical implementation of software for demonstration of work and a sound selection of algorithms that can be applied to non -differential optimization problems. The possibility of use of non -reactive algorithms is demonstrated. Non -reactive algorithms are the only possible method of studying problems that are formulated in the form of nonlinear programming problems using functions that do not have derivatives at any points. The possibilities of non -reactive algorithms are demonstrated by visualizing software components that allow you to sing lines of function levels and graphically display the iteration of the algorithm. Software allows you to clearly demonstrate and explore the work of non -reactive optimization algorithms. The software can be used as a didactic tool. The possibilities of graphical comparison of the features of non -reactive algorithms on the background of the lines of standard functions make the analysis convenient and visual.

Keywords: search methods, algorithms of unconditional first -order optimization, subgradient methods.

ЗМІСТ

Вступ

Актуальність теми. Важливі задачі дослідження операцій сформулюються у вигляді задач нелінійного програмування з допомогою функцій, що не мають похідних в будь яких точках[1-2]. Для аналізу таких задач використовують пошуку або субградієнтний спуск. Субградієнтний спуск — це метод оптимізації, який використовується для знаходження екстремумів невизначених або негладких функцій. Він є розширенням класичного методу градієнтного спуску і дозволяє працювати з функціями, що мають кути, розриви або інші складні особливості. Субградієнт - узагальнення градієнта для негладких функцій, що не мають похідних в будь яких точках. Якщо функція не є гладкою в точці, субградієнт визначається як вектор, який задовольняє умові, що функція вище за лінійну апроксимацію в цій точці. Субградієнт може мати багато значень в одній точці, якщо функція має кути або розриви. Множина субградієнтів для функції f в точці x – це всі можливі субградієнти, тобто всі вектори g, які задовольняють умові:

$f(y) \geq f(x)+g^T(y-x)$ для всіх y

На відміну від градієнта для якого антиградієнт є напрямком зменшення значення функції антисубградієнт не є напрямком зменшення функції і не всі субградієнти відповідають стандартній схемі релаксаційних алгоритмів, що для пошуку мінімуму функції $f(x_k)$ будують послідовності

$x_{k,} \in E^n$, , $k = 0, 1 \ldots, f(x_{k,}) > f(x_{k+1})$.

з допомогою співвідношення

$x_{k+1} = x_k - h_k v_k \quad k = 0, 1 \ldots x_{k+1}, x_k \in E^n, h \in E^1$.

. На прикладах в роботі показано, що правило

$f(x_{k,}) > f(x_{k+1})$.

алгоритмів спуску не є обов'язковим для ефективних програмних реалізацій.

Мета дослідження. Мета дослідження – визначити та порівняти можливості методів, що дозволяють працювати з функціями, що мають кути, розриви або інші складні особливості та створення програмного засобу, що забезпечить інформаційну допомогу та формулювання рекомендацій по визначенню параметрів широкого кола алгоритмів безумовної опуклої оптимізації. Основною метою дослідження є визначення тих методів та технологій, що можуть бути практично застосовані для мінімізації функцій, що не мають похідних в деяких точках.

Для досягнення поставленої мети в роботі необхідно вирішити наступні проблеми:

- Дослідити особливості актуальних задач дослідження операцій, що потребують використання моделей, які сформулюються у вигляді задач нелінійного програмування з допомогою функцій, що не мають похідних в будь яких точках.
- Дослідити можливості програмних реалізацій найбільш поширених нерелаксаційних методів безумовної оптимізації та графічних засобів, які дозволяють проектувальникам формувати обґрунтовані рішення для вибору задач, що можуть бути ефективно досліджені методами негладкої оптимізації. Розробити і реалізувати програмно концептуальні та логічні моделі що є необхідними для відображення та порівняння результатів;
- для реалізації моделей та алгоритмів негладкої оптимізації обрати технології та засоби розробки, що реалізують найбільш ефективні та зручні технології програмування;
- Обґрунтувати вибір та реалізувати інструментальні засоби програмування задач з локалізацією до української мови.

Завдання дослідження. Розробити та експериментально підтвердити працездатність алгоритмів найбільш поширених нерелаксаційних методів негладкої безумовної оптимізації та методів пошуку.

Необхідними до програми є звичайні і стандартні вимоги, що корисними (або навіть обов'язковими) для всіх програмних засобів:

- зручність та нескладність використання та мінімальні витрати для розгортання;
- інтуїтивна зрозумілість стандартних інтерфейсів та практична наочність представлення результатів.

Об'єкт дослідження. Об'єктом є дослідження нерелаксаційних методів негладкої безумовної оптимізації та методів пошуку. Актуальні задачі, моделі та побудови та порівняння необхідних оптимізаційних алгоритмів для відомих задач мінімізації невизначених або недиференційованих функцій. Розглянуті задачі нелінійного та не опуклого програмування. Важливі для наочності методи згорнення обчисленої інформації включають засоби побудови ліній рівня мінімізованих функцій для представлення результатів.

Предмет дослідження. Предметом дослідження є технології та механізми для визначення можливості застосування методів мінімізації невизначених або негладких функцій. Методи пошуку та принципи програмної реалізації методів і алгоритмів для задач оптимізації мінімізації нелінійного та не опуклого програмування також є предметом дослідження

Методи дослідження. Основними методами дослідження проблеми є технологія та шаблони програмування Microsoft Visual Studio, що реалізує сучасні графічні програмні технології які потребує сформульована задача..

Практична цінність - дозволяє покращити якість методів мінімізації невизначених або функцій, що не мають похідних в будь яких точках.

Елементи наукової новизни. Результати дослідження дозволяють обґрунтувати вибір значного класу алгоритмів. Автору роботи програми, що

базуються на подібних принципах і мають наведені характеристики не відомі.

Практична значущість роботи. Описаний в даній роботі програмний продукт для визначення ефективності методів мінімізації невизначених або функцій, що не мають похідних в будь яких точках, є засобом, що може бути використаними для важливих задач дослідження операцій та дидактичним засобом. Програми, що базуються на подібних принципах і мають наведені характеристики не відомі.

Апробація результатів роботи. Результати роботи доповідались на VIII Міжнародній науково-практичній конференції «Мехатронні системи: інновації та інжиніринг» – «**MSIE**-2024» та були перевірені на значній кількості відповідних задач.

РОЗДІЛ 1. ЗАДАЧА ТА МЕТОДИ ДОСЛІДЖЕННЯ

1.1 Постановка задачі

Математичні методи оптимізації – наука, яка вивчає оптимізацію функціонування складних систем. Оптимізація - сукупність фундаментальних математичних результатів і чисельних методів, орієнтованих на пошук оптимального розв'язку практичних задач. На сьогодні теорія оптимізації широко використовується в усіх напрямках інженерної та економічної діяльності, наприклад, при проектуванні систем і їх складових частин, плануванні і аналізі роботи підприємств, управлінні динамічними системами [1, 2]. Під час навчання нейронних мереж головне завдання - мінімізувати (або зменшити) заданий набір помилки (або функції втрати). Функція помилок вимірює, наскільки прогнози моделі відрізняються від справжніх значень (мітків) у навчальних даних. Процес викладання нейронної мережі полягає у налаштуванні її параметрів (вага та переміщення), щоб значення функції помилок став максимально нижчим. Зазвичай це досягається за допомогою алгоритму оптимізації, наприклад, градієнтного спуску. Важливі задачі дослідження операцій і навчання нейронних мереж сформулюються у вигляді задач нелінійного програмування з допомогою функцій, що не мають похідних в будь яких точках[1-21]. Для аналізу таких задач використовують методи пошуку або субградієнтний спуск. Субградієнтний спуск — це метод оптимізації, який використовується для знаходження екстремумів невизначених або негладких функцій. Він є розширенням класичного методу градієнтного спуску і дозволяє працювати з функціями, що мають кути, розриви або інші складні

особливості [3, 21]. Субградієнт - узагальнення градієнта для недіфференційованих функцій. Якщо функція не є гладкою в точці, субградієнт визначається як вектор, який задовольняє умові, що функція вище за лінійну апроксимацію в цій точці. Субградієнт може мати багато значень в одній точці, якщо функція має кути або розриви. Множина субградієнтів для функції f в точці x – це всі можливі субградієнти, тобто всі вектори g, які задовольняють умові:

$f(y) \geq f(x) + g^T(y - x)$ для всіх y

На відміну від градієнта для якого антиградієнт є напрямком зменшення значення функції антисубградієнт не є напрямком зменшення функції і не всі субградієнти відповідають стандартній схемі релаксаційних алгоритмів, що для пошуку мінімуму функції $f(x_k)$ будують послідовності ∇

$x_{k,} \in E^n$, , $k = 0, 1 \ldots, f(x_{k,}) > f(x_{k+1})$.

з допомогою співвідношення

$x_{k+1} = x_k - h_k v_k \quad k = 0, 1 \ldots x_{k+1}, x_k \in E^n, h \in E^1$.

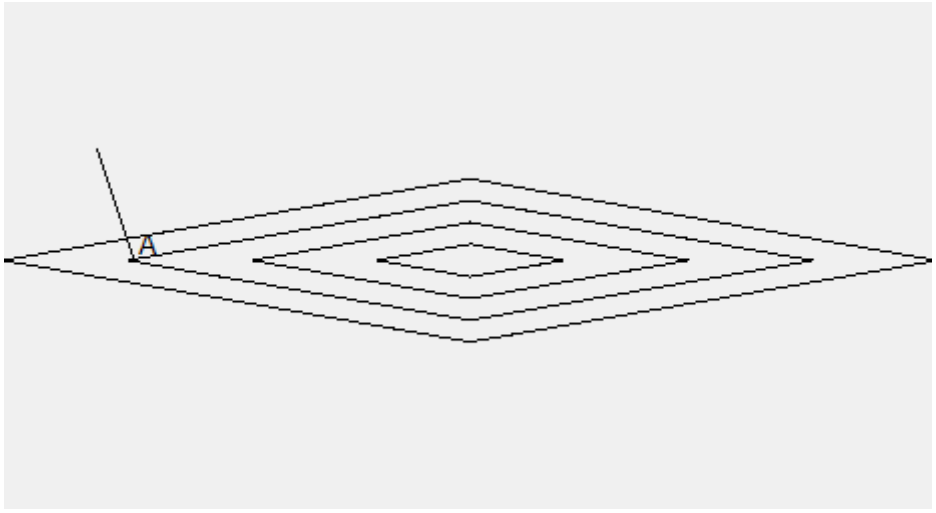

Рисунок. 1.1.1 Для стандартного градієнтного спуску в точці А градієнт не є напрямком спуску.

Рисунок. 1.1.1 ілюструє неспроможність градієнтних методів спуску для функцій, що не мають похідних в будь яких точках [1, 2]. Для стандартного градієнтного спуску в точці А градієнт не є напрямком спуску.

На прикладах в роботі показано, що правило

$f(x_{k,}) > f(x_{k+1})$.

алгоритмів спуску не є обов’язковим для ефективних програмних реалізацій.

Важливий метод для аналізу задач з обмеженнями – метод штрафних функцій. Штрафні функції майже завжди сформулюються у вигляді функцій, що не мають похідних в деяких точках. Лінії рівня таких функцій мають властивості, що демонструє Рис. 1.1.1 Методи штрафних функцій — це один з підходів до розв'язування задач оптимізації, зокрема, задач з обмеженнями. Вони використовуються для трансформації задач з обмеженнями в задачі без обмежень, де можна застосовувати класичні методи оптимізації, такі як градієнтні методи чи методи безпосереднього пошуку. Основна ідея методу штрафних функцій полягає в тому, щоб штрафувати порушення обмежень у функції цілі, перетворюючи задачу з обмеженнями в задачу без обмежень. Це дає можливість застосовувати стандартні методи для мінімізації або максимізації функції [4, 22]. На більшій частині допустимої множини задачі ці функції близькі до нуля (дорівнює 0). Кожна з них достатньо швидко зростає або при наближенні зсередини до межі множини (внутрішні або бар'єрні штрафні функції), або при виході за його межі і віддаленні від нього (зовнішні штрафні функції). Міра близькості штрафу до нуля і швидкість його зростання залежать від значення штрафного параметра та збільшуються з його зростанням. Функція штрафу додається до цільової функції, після чого розв'язується параметричне сімейство отриманих задач без функціональних обмежень, тобто сімейство задач безумовної оптимізації.

основні кроки алгоритму методу штрафних функцій:

Початковий етап:

Вибрати початкову точку x_0, яка задовольняє всі обмеження.

Вибрати початкове значення штрафного коефіцієнта r_0.

Основний етап:

На кожній ітерації k :

Визначити штрафну функцію P(x), яка враховує всі обмеження.

Сформулювати нову цільову функцію

$F(x) = f(x) + r_k P(x)$,

де f(x)) — початкова цільова функція, а r_k) — штрафний коефіцієнт на k-ій ітерації.

Мінімізувати нову цільову функцію F(x) без обмежень.

Оновити штрафний коефіцієнт $r_{k+1} = h^* r_k$, де $h > 1$ — коефіцієнт збільшення штрафу. Після кожної ітерації значення параметра штрафу ρ\rhoρ збільшується, що підвищує вагу порушень обмежень. Процес повторюється, поки не досягнуто бажаного рівня точності або поки штрафи за порушення обмежень не стануть дуже великими..

Перевірити умови зупинки (наприклад, якщо зміна в значеннях цільової функції або змінних менша за заданий поріг).

Завершення:

Процес повторюється до досягнення заданих умов зупинки, після чого отримується наближене розв’язання початкової задачі з обмеженнями.

Існує кілька варіантів методів штрафних функцій:

Метод зовнішнього штрафу: У цьому випадку штраф за порушення обмежень додається до функції цілі, і цей штраф може бути збільшений з часом. Це дозволяє зменшувати вплив порушень обмежень і зосереджуватися на пошуку оптимального розв'язку, що задовольняє обмеження.

Метод внутрішнього штрафу: Це метод, при якому штраф на порушення обмежень додається до функції цілі, але з вимогою, щоб розв'язок залишався всередині області допустимих значень. Внутрішні штрафи можуть використовуватися для задач, де важливо не виходити за межі допустимих значень навіть на початкових етапах оптимізації.

Найпоширенішими є два основні типи штрафних функцій [5,18]:

Абсолютний штраф: $P(x) = \sum_{i=1}^{m} \max(0, g_i(x))$, де $g_i(x) \leq 0$ — це обмеження.

Квадратичний штраф: $P(x) = \sum_{i=1}^{m} \max(0, g_i(x))^2$.

В кожному випадку функції не мають похідних в деяких точках

Для задач нелінійного програмування, що сформулюються у вигляді з функцій, що не мають похідних в будь яких точках існує альтернативна технологія дослідження – це методи пошуку.

Ці методи не вимагають обчислення похідних або градієнтів і часто використовуються, коли функція є негладкою або її похідні важко обчислити.

Опуклий симплекс (Метод проектування многогранника) не є традиційним методом для НЛП, але його варіанти можуть застосовуватись у задачах з лінійними та нелінійними обмеженнями (наприклад, комбінований метод для задач з додатковими нелінійними обмеженнями). Метод проектування многогранника (Polytope Projection Method) — це метод оптимізації, який використовується для розв'язання задач нелінійного програмування (НЛП), зокрема, коли задача має обмеження, описані багатогранниками (многогранниками), тобто множинами, обмеженими лінійними нерівностями. Цей метод може бути особливо корисним для задач з великими розмірами простору рішень, коли традиційні методи оптимізації, такі як методи градієнтного спуску, можуть бути менш ефективними або не забезпечують необхідної точності.

Метод проектування многогранника використовується для знаходження оптимальних рішень через проекцію на многогранники, які визначають множину допустимих рішень. Він є важливим інструментом у теорії оптимізації, зокрема, для задач з великими обмеженнями або складними геометрі ями.

Метод Нелінійного Програмування Великої Шкали (Genetic Algorithm, GA)

Генетичні алгоритми є методом еволюційного пошуку, що використовується для задач, де звичайні градієнтні методи не підходять через складність функцій чи обмежень. Вони працюють шляхом створення

популяції можливих рішень та їх еволюції через відбір, схрещування та мутації. Алгоритм наступний:

Генерація початкової популяції.

Оцінка фітнесу кожного індивіда.

Вибір пар для схрещування.

Мутація та комбінування для створення нових індивідів.

Повторення процесу до досягнення критеріїв зупинки.

Метод може ефективно вирішувати задачі з великими розмірами простору пошуку.

Підходить для складних і негладких задач. Але є недоліки:

Вимагає налаштування кількох параметрів.

Може бути менш точним за інші методи.

Метод рою частинок (Particle Swarm Optimization, PSO)

Метод рою частинок — це ще один метод еволюційного пошуку, який базується на концепції групового інтелекту. Рой частинок поширюється по просторі пошуку, і кожна частинка має свою швидкість і напрямок. Алгоритм наступний:

Ініціалізація частинок у просторі пошуку.

Оновлення швидкості та положення частинок на основі особистого та глобального досвіду.

Повторення до досягнення бажаного результату або досягнення ліміту ітерацій.

Метод може ефективно працювати в просторах з великою кількістю змінних.

Не вимагає градієнтів функції. Але має недоліки:

Може бути повільним.

Часом не гарантує досягнення глобального оптимуму.

Зважаючи наведене мета дослідження – визначити та порівняти можливості методів, що дозволяють працювати з функціями, що мають кути, розриви або інші складні особливості та створення програмного засобу, що

забезпечить інформаційну допомогу та формулювання рекомендацій по визначенню параметрів широкого кола алгоритмів безумовної опуклої оптимізації.

1.2. Огляд предметної області - дослідження операцій

Моделі прийняття рішень, які формулюють у вигляді задач нелінійного програмування, можуть включати [23,27]:

1. Задача квадратичного програмування: Мета – мінімізувати або максимізувати квадратичну функцію за певних обмежень.
2. Задача з нелінійними обмеженнями: Це може бути будь-яка функція, яка не є лінійною і має нелінійні обмеження.
3. Стохастичне нелінійне програмування: Оптимізація, яка враховує ймовірність і випадковість у моделі.
4. Моделі керування запасами: Оптимізація управління запасами з урахуванням нелінійних витрат на зберігання та замовлення.

Задачі кожного типу містять задачі, що можуть бути сформульовані як задачі з обмеженнями Важливий метод для аналізу задач з обмеженнями – метод штрафних функцій. Штрафні функції майже завжди сформулюються у вигляді функцій, що не мають похідних в деяких точках. Лінії рівня таких функцій мають властивості, що демонструє Рис. 1.1.1

Прикладом задачі дослідження операцій, що сформулюються у вигляді задач нелінійного програмування з допомогою функцій, що не мають похідних в будь яких точках є задача про оптимальне завантаження устаткування. Відома лінійна модель задачі про оптимальне завантаження устаткування підкуповує простотою, але не враховує час переналагодження устаткування.

У більшості реальних ситуацій для того, щоб реалізувати необхідний спосіб виробництва, устаткування потрібно зупинити й переналагодити. Якщо $t(x_j)$ - час, необхідний для організації j - того способу виробництва, то реальний час, протягом якого устаткування буде виробляти продукцію T - $\sum_{j=1}^{n} t(x_j)$. Підсумовування здійснюється тільки по тим компонентам

вектора X, що більше 0. Функція $t(x_j)$ =0 якщо $x_j=0$ і $t(x_j)$ =t_j якщо $x_j>0$ не є гладкою. З огляду на властивості задачі лінійного програмування можна припустити, що відмінних від нуля компонент у векторі вирішення задачі X буде r= min (n,m) (m-рівно кількості обмежень). Якщо m і n достатньо великі числа, то час, протягом якого устаткування буде виробляти продукцію відповідно до нашої програми, може бути менше 0.

Цю перешкоду легко перебороти за допомогою нелінійної, що не є гладкою або дискретної моделі, проте відомі дискретні моделі цієї задачі будуть мати таку розмірність, що їхній реальний аналіз неможливий, а вирішення нелінійної задачі є цілком реальним.

Витрати на переналагодження обладнання є одноразовими і займають фіксований час. Реальна функція витрат на реалізацію j - того способу виробництва не є гладкою і виглядає в такий спосіб $c_j(x_j)=$ 0 , якщо $x_j =0$, $c_j(x_j)=$ c_{pj} , якщо $0<x_j <t_j$,, $c_j(x_j)= c_{pj}+c_j*(x_j - t_{pi})$, якщо $x_j > t_{pi}$

Ця функція має розрив в точці $x_j =0$. Тут c_j можна інтерпретувати як приведені витрати на реалізацію і- того способу виробництва в одиницю часу, а c_{pi} як витрати на переналагодження устаткування з метою реалізації j того способу виробництва.

Виробництво продукції j тим способом, з урахуванням переналагоджень можна описати виразом $a_{ij}(x_i) =$ 0, якщо $0 <x_i < t_i$, $a_{ij}(x_i) = ap_{ij}(x_j - t_j)$ якщо

$x_j \geq t_j$. Ця функція не має похідних в точці $x_j =$ але є безперервною.

Тепер адекватна модель задачі планування завантаження устаткування має вид

$$\sum_{j=1}^{n} c_j(x_j) \text{ -> min}$$

$$\sum_{j=1}^{n} a_{ij}(x_i) \geq b_i , \ i = 1, \ldots , m,$$

$$\sum_{j=1}^{n} x_j \leq T$$

$$x_j \geq 0..$$

T - тривалість інтервалу планування

Ця задача є типовою задачею дискретного програмування[1], тобто належить колу задач, для яких не відомі ефективні алгоритми вирішення (для відомих алгоритмів кількість обчислень для вирішення задачі є експоненціальною функцією від кількості змінних). Задача є задачею дискретного програмування тому, що функція $c_j(x_j)$ є розривною.

Нижче приведена задача нелінійного програмування яка є еквівалентною задаче 1, але може бути досліджувана більш ефективними методами, що гарантують успіх. Ця задача відрізняється лише виглядом цільової функції. Модифікована функція витрат на реалізацію j - того способу виробництва виглядає в такий спосіб $c_j(x_j) = x_j * (c_{pj} / t_j,)$ якщо $0 < x_j < t_j,,$ $c_j(x_j) = c_{pj} + c_j * (x_j - t_{pi})$, якщо $x_j > t_{pi}$

Задача з модифікованою функцією витрат є еквівалентною задачі дискретного програмування (розв'язок нелінійної задачі є розв'язками дискретної), є задачею з безперервною цільовою функцією і для неї можуть бути застосовані алгоритми нелінійного програмування.

Необхідно додати що буть яке розв'язання задач з обмеженнями методом штрафних функцій теж приводить до мінімізації негладких функцій.

1.3 Огляд предметної області-навчання нейронних мереж

Навчання нейронних мереж — це процес оптимізації параметрів (ваг і зсувів) нейронної мережі так, щоб вона здатна була виконувати певне завдання, наприклад, класифікацію, регресію або інші види прогнозування, на основі вхідних даних. Основна мета навчання — це мінімізація функції втрат (або "ліхоманки"), яка вимірює різницю між передбаченнями мережі та реальними значеннями (цільовими даними).

Процес навчання нейронних мереж можна розділити на кілька основних етапів:

Ініціалізація мережі

Нейронні мережі складаються з кількох шарів нейронів: вхідного шару, кількох прихованих шарів і вихідного шару. Кожен нейрон в мережі має ваги (коефіцієнти, які визначають важливість кожного вхідного сигналу) і зсуви (bias), які допомагають моделювати складні функції. На початковому етапі ці параметри зазвичай визначають випадковим чином. На самому початку мережа має випадково визначені ваги та зміщення. Це є початковими значеннями, з якими нейронна мережа розпочинає навчання.

Прямий прохід [24] - Це етап, коли вхідні дані подаються до нейронної мережі і обчислюються передбачення:

Вхідні дані передаються до кожного нейрону в мережі. Кожен нейрон обчислює свій вихід на основі вхідних значень, ваг і зсуву, застосовуючи певну активаційну функцію (наприклад, ReLU, сигмоїду, тангенс гіперболічний і т. д.). Для кожного прикладу навчання мережа приймає вхідні дані, проходить через усі шари та обчислює вихід.

Вхідні дані подаються до першого шару нейронів. Кожен нейрон у шарі здійснює лінійне зважене сумування вхідних значень (ваги множаться на відповідні входи та додаються зміщення). Це значення передається через функцію активації (наприклад, ReLU, Sigmoid, Tanh), яка визначає, чи буде нейрон активований. Процес повторюється для кожного наступного шару до вихідного. Результати передаються далі через всі шари до вихідного шару.

Обчислення функції втрат (Loss Function)

Після того як нейронна мережа зробила передбачення, необхідно оцінити, наскільки ці передбачення близькі до фактичних значень. Для цього використовується функція втрат (loss function). Найпоширеніші функції втрат:

Середньоквадратична помилка (MSE, Mean Squared Error) — часто використовується для задач регресії.

Крос-ентропія (Cross-Entropy) — використовується для задач класифікації.

Функція втрат оцінює різницю між передбаченнями нейронної мережі та реальними значеннями, і ця різниця визначає, наскільки добре або погано

працює мережа. Для кожного прикладу навчання мережа приймає вхідні дані, проходить через усі шари та обчислює вихід. Вхідні дані подаються до першого шару нейронів.

Кожен нейрон у шарі здійснює лінійне зважене сумування вхідних значень (ваги множаться на відповідні входи та додаються зміщення).

Це значення передається через функцію активації (наприклад, ReLU, Sigmoid, Tanh), яка визначає, чи буде нейрон активований.

Процес повторюється для кожного наступного шару до вихідного.

Зворотний прохід (Backpropagation)

Зворотний прохід — це метод, що дозволяє нейронній мережі коригувати свої параметри на основі того, як сильно кожен параметр (вага чи зсув) вплинув на загальну помилку.

Обчислення градієнтів: Зворотний прохід починається з обчислення градієнта функції втрат по кожному параметру (вазі і зсуву) мережі. Це дозволяє зрозуміти, як змінити параметри, щоб зменшити помилку. Щоб зменшити похибку, потрібно оновити ваги. Це відбувається через зворотне поширення помилки:

Визначається похибка на виході мережі.

За допомогою методів диференціації (наприклад, градієнтного спуску) похибка поширюється зворотно через мережу.

Для кожного нейрона обчислюється, як його вага впливає на загальну похибку.

Оновлення параметрів: Використовуємо методи оптимізації для оновлення ваг і зсувів, щоб мінімізувати функцію втрат. Одним з найбільш популярних методів мінімізації функції втрат є градієнтний спуск.

Функція втрат є однією з найважливіших складових процесу навчання нейронної мережі. Вона визначає, як сильно передбачення моделі відрізняються від реальних значень, і служить орієнтиром для коригування ваг нейронної мережі через зворотне поширення помилки. Функція втрат приймає два параметри (дві групи параметрів):

y — реальні значення (маркери класів або істинні числові значення).

y^ — передбачення нейронної мережі (вихідні значення, отримані після прямого проходження через мережу).

Мета навчання — мінімізувати значення функції втрат, тобто зробити різницю між реальними та передбаченими значеннями мінімальною.

Функція втрат вимірює, наскільки "погано" модель передбачає результат, і на основі цієї величини нейронна мережа коригує свої параметри (ваги) для покращення результатів. В більшості випадків функції втрат це функції, що не мають похідних в деяких точках[24-26].

1.4. Висновки до розділу

Важливі задачі дослідження операцій і навчання нейронних мереж сформулюються у вигляді задач нелінійного програмування з допомогою функцій, що не мають розривів але і не мають похідних в деяких точках. Для аналізу таких задач використовують методи пошуку або субградієнтний спуск. Кожна група методів має значну кількість модифікацій і важливо визначити наперед де вони можуть бути застосовані. Наприклад. Субградієнтний спуск — це метод оптимізації, який використовується для знаходження екстремумів невизначених або негладких функцій. Він є розширенням класичного методу градієнтного спуску і дозволяє працювати з функціями, що мають кути, розриви або інші складні особливості. вирішувати проблеми, що виникають при прийнятті рішень.

Можливість ефективного застосування різних алгоритмів є невизначеною тому корисно розглянути на наочних прикладах можливості різних модифікацій алгоритмів. У роботі розглядаються приклади простих алгоритмів для аналізу задач знаходження екстремумів негладких функцій. Частина алгоритмів не є релаксаційними – не задовольняють умові $f(x_{k,}) > f(x_{k+1})$.*н*а кожній ітерації але є ефективними. Графічні засоби програми дозволять підтвердити ефективність алгоритмів На рис. 1.1.1,

наведені лінії рівня функції, що не має похідних в точках де один з узагальнених градієнтів дорівнює 0.

РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА ОБҐРУНТУВАННЯ НЕОБХІДНИХ МОДЕЛЕЙ І МЕТОДІВ

2.1 Принципи оцінки якості алгоритмів безумовної оптимізації

Оцінка якості алгоритмів оптимізації є важливою складовою їхнього вибору та застосування в задачах нелінійного програмування, що сформульовані з допомогою функцій, які не мають розривів але і не мають похідних в деяких точках. Для оцінки ефективності алгоритмів оптимізації використовують різні критерії, залежно від специфіки задачі та алгоритму. Наведемо основні принципи, за якими оцінюють якість алгоритмів оптимізації[28-30]:

1. Точність

Точність рішення — це здатність алгоритму знаходити рішення, яке є найбільш наближеним або ідеальним для даної оптимізаційної задачі. Наприклад, для задачі мінімізації це може бути значення функції вартості в глобальному мінімумі або близькому до нього.

Алгоритм може бути "точним" не лише за результатом, але й за здатністю досягти рішення, яке є хорошим або оптимальним навіть при обмеженнях часу чи ресурсів.

2. Швидкість

Це час, за який алгоритм знаходить рішення, тобто обчислювальна складність алгоритму. Вона зазвичай вимірюється через час виконання на різних тестових наборах даних або через асимптотику складності (O-нотація).

Для великих задач важливо, щоб алгоритм був ефективним з точки зору часу та ресурсів, а не просто точним.

3. Загальна складність

Часова складність описує, як швидко зростає час виконання алгоритму при збільшенні розміру задачі. Це може бути лінійна, квадратична, кубічна чи експоненціальна складність.

Просторова складність відноситься до кількості пам'яті, яку алгоритм потребує для обробки даних. Важливо враховувати як часову, так і просторову складність при розробці алгоритму для великих задач.

4. Стійкість

Стійкість алгоритму означає його здатність працювати без значних коливань чи непередбачуваних результатів на різних вхідних даних, а також його здатність досягати подібних результатів при повторному виконанні (відсутність великих варіацій у результатах).

Важливо, щоб алгоритм був стабільним навіть при наявності шуму в даних або невизначеності.

5. Глобальність

Це здатність алгоритму знаходити глобальне оптимальне рішення, а не лише локальні оптимуми. У задачах з багатьма локальними мінімумами алгоритм має здатність уникати "пасток" і знаходити найкраще можливе рішення на всьому просторі.

Алгоритми, такі як генетичні алгоритми, симульоване відпікання або мурашині алгоритми, зазвичай мають кращу здатність до глобальної оптимізації.

6. Здатність до масштабування

Здатність алгоритму ефективно працювати при збільшенні розміру задачі або кількості змінних. Алгоритм повинен бути здатним адаптуватися до великих даних і розв'язувати складніші задачі без значного зростання часу виконання або використання пам'яті.

Це важливо, коли задача вимагає оптимізації при великій кількості параметрів або змінних.

7. Універсальність

Це здатність алгоритму бути застосованим до різних типів оптимізаційних задач. Наприклад, деякі алгоритми можуть бути універсальними і підходити для чисельних, комбінаторних або навіть стохастичних задач, тоді як інші можуть бути спеціалізованими для конкретних типів задач.

8. Чутливість до параметрів

Багато алгоритмів мають параметри, які треба налаштовувати. Чутливість алгоритму до цих параметрів визначає, наскільки ефективно алгоритм працюватиме при різних значеннях параметрів.

Якщо алгоритм сильно залежить від вибору параметрів (наприклад, швидкість навчання в нейронних мережах), це може вимагати додаткових зусиль для налаштування і оптимізації.

9. Економія ресурсів

Важливо, щоб алгоритм використовував мінімальні ресурси (пам'ять, процесорний час, енергоспоживання). Це особливо критично для великих систем, де ефективність використання ресурсів може бути ключовим фактором.

Особливо в контексті обчислювальних кластерів або обмежених обчислювальних потужності економія ресурсів може бути важливою.

10. Зручність і простота реалізації

Деякі алгоритми прості в реалізації і не потребують великої кількості спеціальних знань чи складних математичних моделей, тоді як інші можуть бути досить складними і вимагати багато додаткових кроків для правильного застосування.

Це важливий критерій для вибору алгоритму в реальних додатках, де потрібно зберігати баланс між складністю розробки та ефективністю.

11. Надійність

Це здатність алгоритму працювати у випадку непередбачених або ненормальних вхідних даних, таких як шуми, викиди або неповні дані.

Надійний алгоритм повинен мати здатність до адаптації до таких змін без значної втрати продуктивності.

12. Гнучкість

Гнучкість алгоритму означає його здатність працювати з різними видами задач і умовами. Це важливо, оскільки багато реальних задач можуть включати обмеження (в даному випадку не гладкість) або інші складні характеристики, з якими алгоритм повинен справлятися.

Частіше за все реалізують наступні задачі.

Тестування на реальних даних: Використання реальних даних для оцінки ефективності алгоритму.

Симуляція: Створення моделі, яка імітує реальні умови, для випробування алгоритму.

Метрики ефективності: Використання таких метрик, як час виконання, пам'ять, точність, для порівняння алгоритмів.

Аналіз чутливості: Оцінка, як зміни вхідних даних впливають на результати алгоритму.

Контрольні тести: Використання набору контрольних даних для перевірки правильності роботи алгоритму.

В роботі для перевірки якості алгоритмів використовується 4 типи тестових завдань. Для цих функцій були розроблені програми, що відображають лінії рівня.

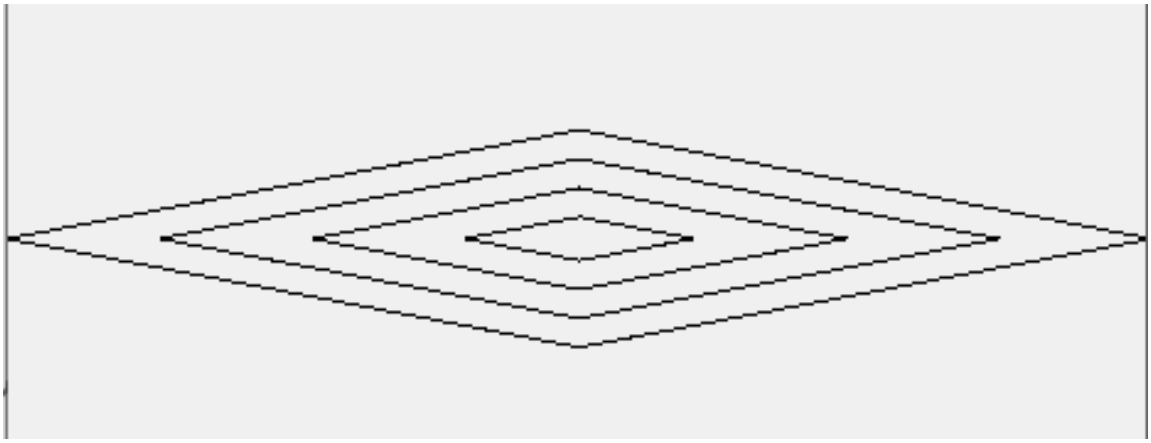

Рис 2.1.1.Лінії рівня негладкої функції

Тест для цього задання є базовим. Функція визначається наступними співвідношеннями

Якщо (x >=0 & y >=0) f(x,y) = x + 5 * y;

якщо (x >= 0 & y <=0) f(x,y)= x -5 * y;

якщо (x <=0 & y<=0) f(x,y) = -x - 5 * y;

якщо (x <= 0 & y >= 0) f(x,y)= -x + 5 * y;

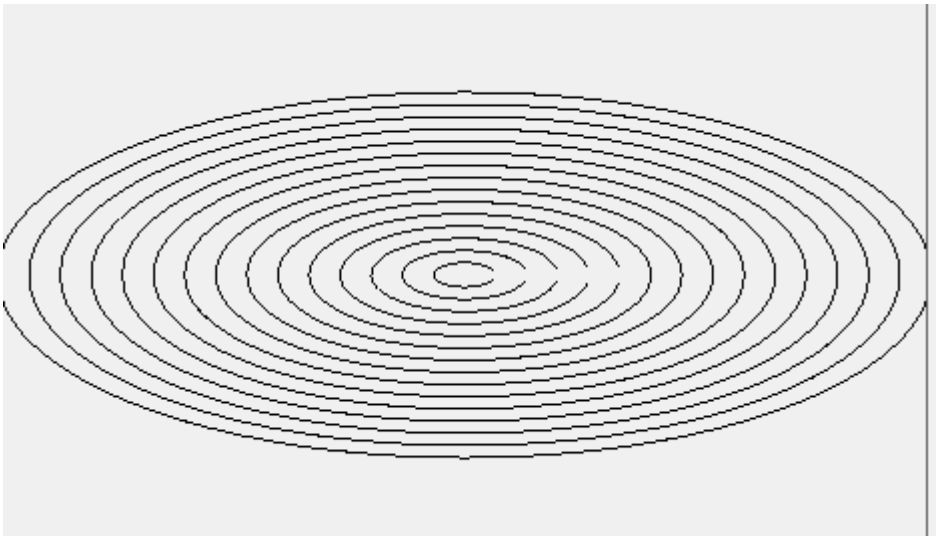

Рис 2.1.2Лінії рівня сепарабельної функції

Тест для цього задання є найпростішим

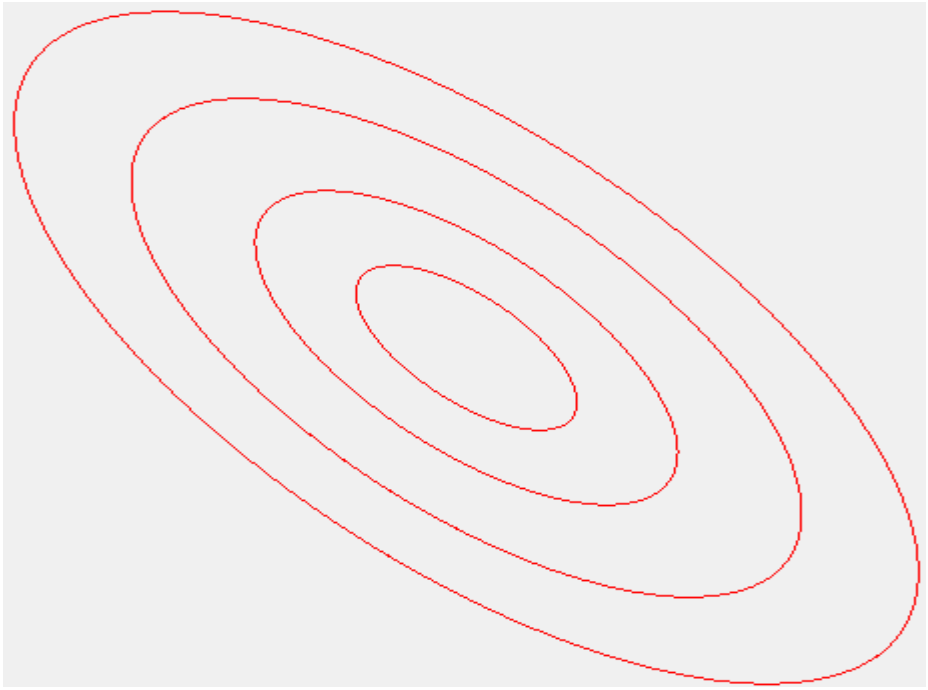

Рис 2.1.3 Лінії рівня еліптичної функції

Тест для цього задання є складнішим ніж тест сепарабельної функції. Властивості наведених функцій є зрозумілими і наочними. Складнішим випадком є наступна функція.

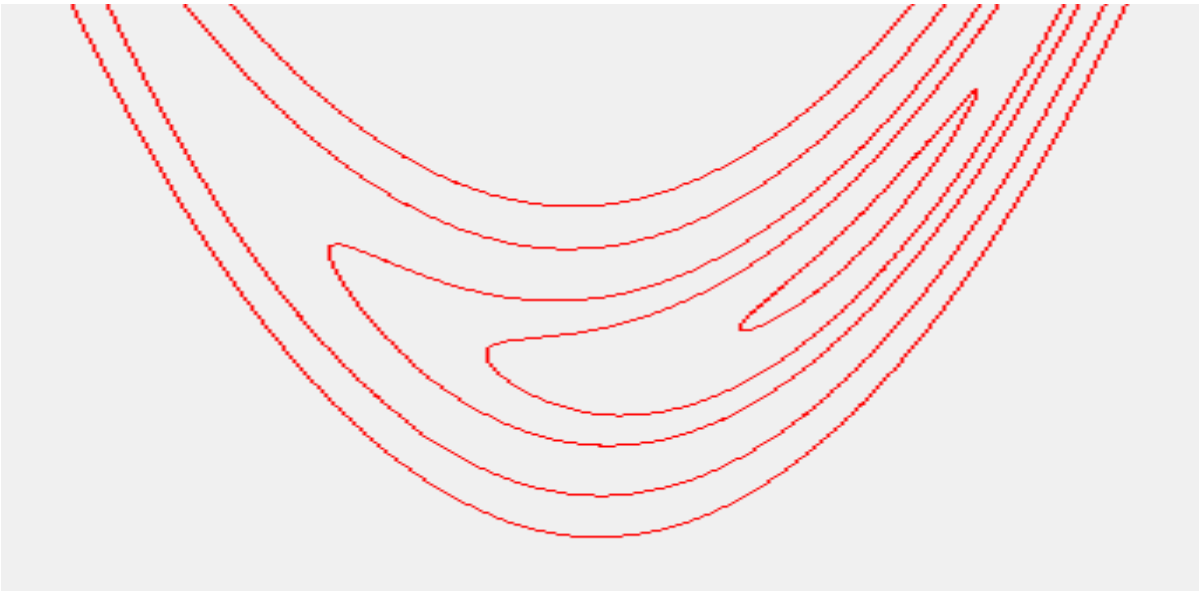

Рис 2.1.4. Функція Розенброка

Функція Розенброка (іноді зветься Розенброком або "Розенброком на площині") широко використовується для оцінки алгоритмів оптимізації. Вона була запропонована у 1960 році для тестування методів оптимізації багатьох змінних. Функція Розенброка є важливим інструментом для тестування алгоритмів оптимізації завдяки її складності й характерним властивостям. Вона дозволяє оцінити здатність алгоритмів досягати точних рішень у складних, але добре визначених просторах, де є одна точка мінімуму, але для її досягнення може знадобитися значний обчислювальний ресурс. Загальний вигляд функції для двох змінних (найпоширеніший варіант:

$$f(x, y) = (a - x)^2 + b(y - x^2)^2$$

У цьому випадку функція має мінімум, коли x=a і y=a^2, тобто для a=1, глобальний мінімум буде на точці (x,y)=(1,1)

Для багатьох змінних функція Розенброка виглядає наступним чином:

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} [a - x_i]^2 + b \left(x_{i+1} - x_i^2\right)^2$$

Функція Розенброка має характерний вигляд "каньйону". Вона має вузьку "долину" (область мінімуму), по якій розташовані точки, що призводять до мінімального значення функції, при цьому значення функції в інших областях простору швидко зростає.

Глобальний мінімум функції досягається в точці (1,1,...,1) для всіх змінних a=1.

Мінімум є м'яким, тобто зменшення функції може бути не дуже різким, що ускладнює задачу для деяких алгоритмів оптимізації.

Властивості функції Розенброка

1. Глобальний мінімум: Функція має глобальний мінімум в точці (1,1,...,1), де значення функції дорівнює f(1,1,...,1)=0.
2. Локальні мінімуми: Оскільки функція має складну форму, де зростання функції навколо мінімуму повільно відбувається по всіх напрямках, це може ускладнити пошук мінімуму, особливо для градієнтних методів (через широкі "плоскі" ділянки).
3. Вузька долина мінімуму: Простір оптимізації дуже вузький у напрямку до мінімуму, що призводить до проблем при застосуванні числових методів, особливо для методів з градієнтним спуском, через необхідність сильно точних налаштувань параметрів кроку.
4. Необхідність правильно налаштовувати крок: Для деяких алгоритмів оптимізації, зокрема для градієнтного спуску, необхідно ретельно налаштовувати розмір кроку, щоб уникнути того, що алгоритм не зможе ефективно рухатися по вузькому каналу мінімуму.

Використання в оптимізації

Функція Розенброка є стандартним прикладом для тестування алгоритмів оптимізації завдяки своїм особливостям:

Вона має глобальний мінімум, що дозволяє перевірити, чи алгоритм здатний знайти найкраще рішення.

Вона є неопуклою (має складну форму) і має плоскі області, що робить її хорошим прикладом для тестування стійкості алгоритмів до локальних мінімумів та перевірки їх здатності адаптуватися до складної форми простору розв'язків.

2.2 Методи узагальненого градієнту

Методи узагальненого градієнту це ітеративні алгоритми, що для отримання розв’язку задачі мінімізації негладких функцій будують послідовності

$$x_1, \ldots, x_k, x_{k+1} \ldots \in E^n,$$

Послідовності необов’язково є релаксаційними але повинні збігатись до x_k, що може бути розв’язкам задачі мінімізації. Послідовності будуються відповідно за спільною для всіх градієнтних алгоритмів формулою

$$x_{k+1} = x_k - h1_k v_k, - h2_k g_k \quad k = 0, 1 \ldots x_{k+1} \in E^n \quad , \; x_k , g_k \in E^n \quad h_k \in E^1$$

$h1_k$, $h2_k$ належать множині субградієнтів.

Обов’язкове правило алгоритмів спуску

$$f(x_k) > f(x_{k+1}).$$

не є обов’язковим для ефективних програмних реалізацій.

Субградієнт — це узагальнення поняття градієнта для функцій, які не є диференційованими в усіх точках, але є конвексними. Якщо функція є конвексною, то навіть у точках, де вона не є гладкою (не має похідної), можна визначити субградієнт. Це важливе поняття у конвексному аналізі та оптимізації, особливо при роботі з функціями, які мають "зламану" або "негладку" структуру, наприклад, у задачах з L1-регуляризацією або у слабко диференційованих функціях.

Множина субградієнтів для функції f в точці x – це всі можливі субградієнти, тобто всі вектори g, які задовольняють умові:

$f(y) \geq f(x) + g^T(y - x)$ для всіх y

Ця нерівність говорить про те, що лінійна функція, побудована через субградієнт g в точці x, буде **нижньою границею** для функції f(x) у всіх точках x.

Конвексність функції гарантує, що функція не має різких змін, і існує лінійний наближення (лінійний субградієнт), який "підтримує" графік функції знизу.

У точці x, якщо функція є гладкою, то субградієнт дорівнює звичайному градієнту.

Якщо функція має "зламану" точку або не є диференційовною в точці, то в цій точці може бути декілька можливих субградієнтів, що визначають лінійні наближення знизу.

Алгоритм для знаходження субградієнта наступний:

Визначення точки, в якій шукаємо субградієнт: для цього вибираємо точку x , в якій функція не є диференційованою, але вона є конвексною.

Побудова нерівності: для кожної точки x ми повинні побудувати лінійну функцію через точку , що буде нижньою границею для функції. Якщо функція гладка в цій точці, то це просто її градієнт. Якщо функція незгладжена, то потрібно обчислити або знайти наближене значення субградієнта.

Метод множників Лагранжа або графічний підхід: якщо функція складна, можна використовувати числові методи для знаходження субградієнтів, наприклад, методи оптимізації, що вимагають наявності градієнта (в тому числі субградієнта).

Розглянемо функцію:

$f(x) = |x|$

Вона є конвексною, але не диференційовною в точці x=0. Тут ми можемо визначити субградієнт за допомогою концепції конвексності. Для f(x)=|x| субградієнт у точці x=0 буде множиною значень від −1- до 1, тобто:

$\partial f(0) = [-1, 1]$.

Це означає, що лінії, які проходять через точку (0,0) з нахилами в межах від −1 до 1, будуть підставою для лінійних наближень цієї функції. У решті точок, де x≠0, функція є диференційовною, і її субградієнт дорівнює градієнту (*тобто,* ***f'(x)=1для x>0 і f'(x)=−1*** *для x<0).*

Функція, що містить L1-норму, також не є диференційованою в точці x=0. (приклад 1.1.1) Нехай:

$f(x)=\|x\|_1 - \sum|x_i|$

Тут субградієнт для кожного компонента хіфункції f(x) можна визначити так:

Якщо xi≠0 то субградієнт дорівнює ***sign(xi)*** (де sign(xi) — знак функції).

Якщо xi=0x_i = 0xi=0, то субградієнт може бути будь-яким значенням в інтервалі [−1,1].

Таким чином, субградієнт функції буде вектором, кожен елемент якого є:

sign(xi)), якщо xi≠0,

будь-яким значенням з інтервалу [−1,1], якщо xi=0.

Метод субградієнтного спуску) — є аналогом класичного методу градієнтного спуску, але з використанням субградієнтів для нелінійних або негладких функцій але є питання конкретної реалізації.

Надалі в якості множин узагальненого градієнта точки x_0 будемо використовувати:

- напрямки вздовж координатних осей – будемо називати алгоритм 1;
- градієнт точки яка максимально наближена до точки x_0 (в околі що визначається точністю обчислень) - будемо називати алгоритм 2.

Стратегії вибору кроку для методів, що будемо досліджувати є критично важливим для ефективності алгоритму. Якщо крок занадто малий, алгоритм буде повільно сходитися, а якщо він занадто великий — може не збігатися або навіть збігтися в неправильну точку. Наведемо кілька основних стратегій вибору кроку для градієнтного спуску:

1. Фіксований крок

Найпростіша стратегія — це вибір постійного кроку h, який не змінюється протягом всього процесу оптимізації. Це означає, що на кожній ітерації ви просто рухаєтеся у напрямку антиградієнта на фіксовану відстань.

Вважається що цей метод має наступні переваги та недоліки:

Простота реалізації.

Підходить для задач, де градієнт не змінюється різко і оптимальний крок можна обрати заздалегідь.

Недоліки:

Потрібно точно підібрати значення кроку, яке працює на всіх етапах оптимізації.

Якщо крок занадто великий, алгоритм може «перехопити» мінімум і не збігатися.

Якщо крок занадто малий, навчання буде дуже повільним.

Адаптивне зменшення кроку

Ця стратегія передбачає поступове зменшення кроку в міру того, як наближаємося до мінімуму. Зазвичай для цього використовуються різні математичні функції, які визначають, як змінюється значення кроку по ходу оптимізації.

На початкових етапах, коли далеко від мінімуму, крок може бути більшим для швидшого сходження.

Після досягнення мінімуму (або наближення до нього) крок зменшується, що дозволяє зробити кроки більш точними.

Але є недоліки:

Потрібно налаштувати параметри для темпу зменшення кроку.

Може призвести до надмірного зменшення кроку, що ускладнює подальше навчання.

Адаптивні методи адаптують крок не тільки для кожної ітерації, але й для кожної змінної (параметра), враховуючи їх специфічні властивості. Вони застосовують різні стратегії для змінного темпу навчання, що дозволяє

алгоритму швидше сходитися і коректно масштабувати крок в залежності від характеристик даних.

AdaGrad: адаптує крок на основі історії градієнтів для кожної змінної – чим більша часткова похідна тем менше крок для змінної.

$$\alpha_t = \frac{\alpha}{\sqrt{G_t} + \epsilon}$$

G_t — сума квадратів градієнтів (часткова похідна) на ітераціях до t, є— мале число для уникнення ділення на нуль.

Метод RMSProp: покращення AdaGrad, де використовується монотонне згладжування для історії градієнтів:

$$v_t = \beta v_{t-1} + (1-\beta)\nabla f(\mathbf{x}_t)^2$$

$$\alpha_t = \frac{\alpha}{\sqrt{v_t} + \epsilon}$$

Метод лінійного пошуку - Ця стратегія полягає в тому, щоб на кожному кроці вибирати оптимальний крок α , який мінімізує функцію на напрямку поточного градієнта:

$$\alpha_t = \arg\min_{\alpha} f(\mathbf{x}_t - \alpha \nabla f(\mathbf{x}_t))$$

Для 4 типів задач були проведені чисельні експерименти і результатами були оцінки кількості обчислень різними методами та графічні траєкторії наближення до рішення які наведені нижче.

Для сепарабельної функції всі алгоритми ефективно визначили мінімум. Найкращі результати продемонстрували методи де використовується монотонне згладжування для історії градієнтів, та метод лінійного пошуку (кількість ітерацій) - Рис 2.2.1.

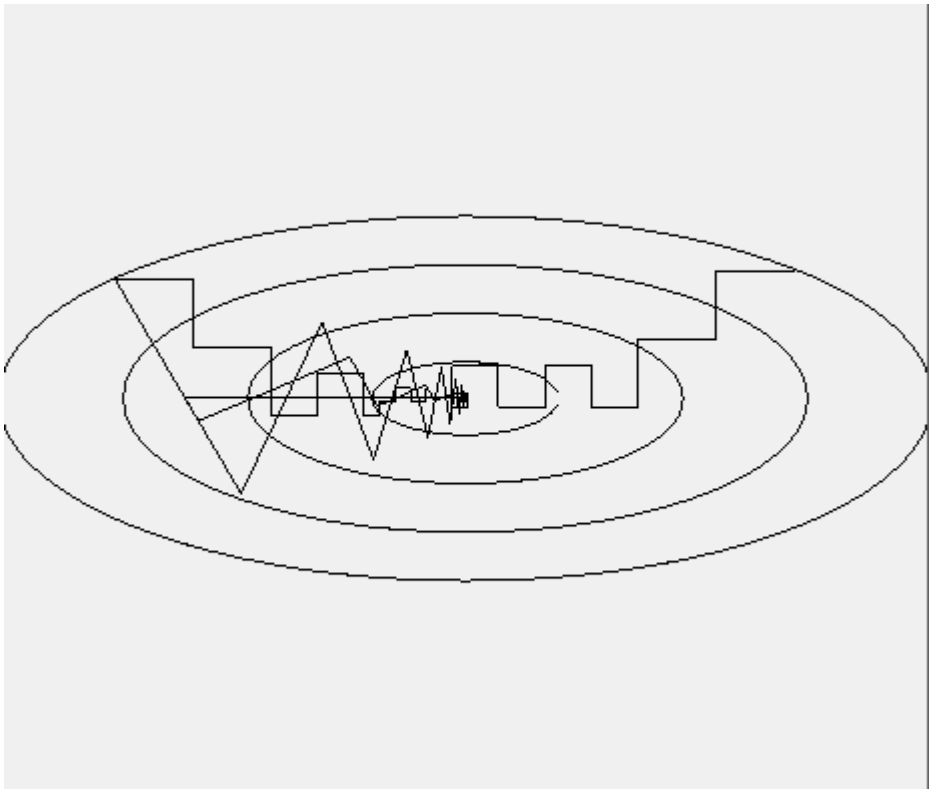

Рис 2.2.1 Траєкторії субградієнтних алгоритмів для сепарабельної функції.

На рис 2.2.1 можна побачити що правило $f(x_k) > f(x_{k+1})$. алгоритмів спуску виконувалось для всіх методів крім покординатного спуску с постійним кроком.

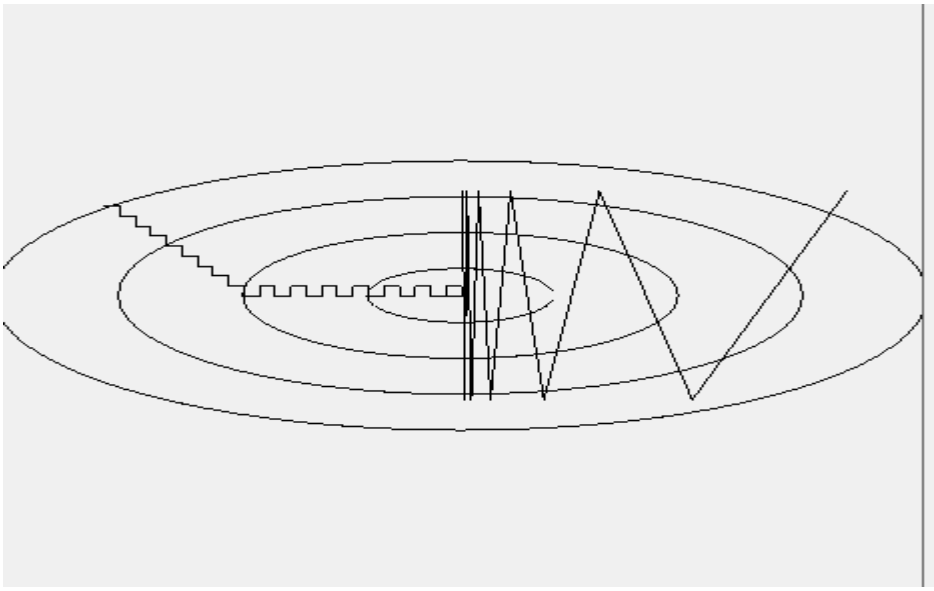

Рис 2.2.2 Траєкторії нерелаксаційних субградієнтних алгоритмів для сепарабельної функції (алгоритм 1 та алгоритм 2).

Нерелаксаційні алгоритмів для сепарабельної функції (алгоритм 1 та алгоритм 2)теж дозволяють отримати результат.

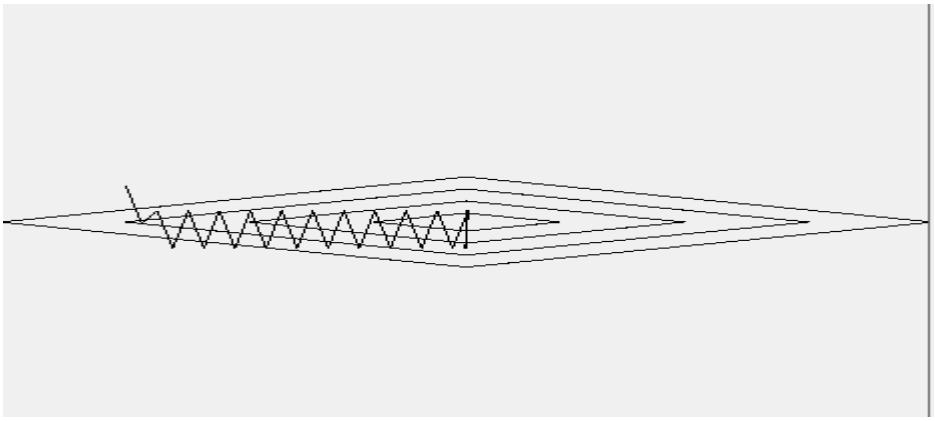

Рис 2.2.3.1 Траєкторії субградієнтних алгоритмів для ї функції що не має похідних (алгоритм 2).

Наочний варіант нерелаксаційних алгоритмів демонструє наступний малюнок.

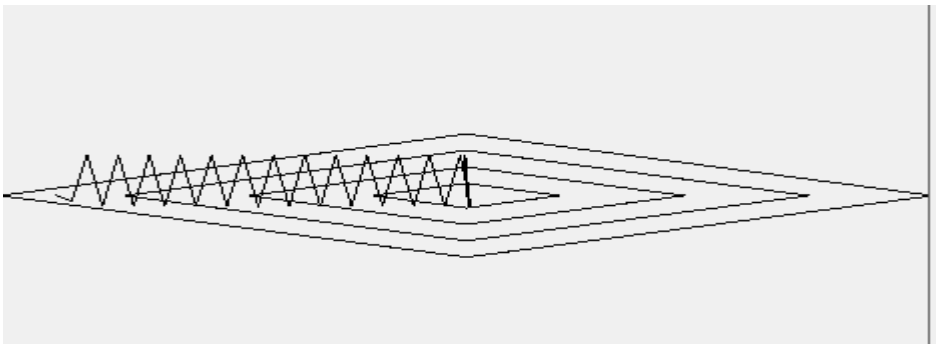

Рис 2.2.3.2 Траєкторії субградієнтних алгоритмів для функції що не має похідних (алгоритм 2 з постійною відстанню між послідовними планами).

На Рис 2.2.3.2 можна чітко побачити, що більшість ітерацій алгоритму порушує правила алгоритмів спуску

$$f(x_{k,}) > f(x_{k+1}).$$

Всі алгоритми дозволяють отримати розв’язок задачі оптимізації.

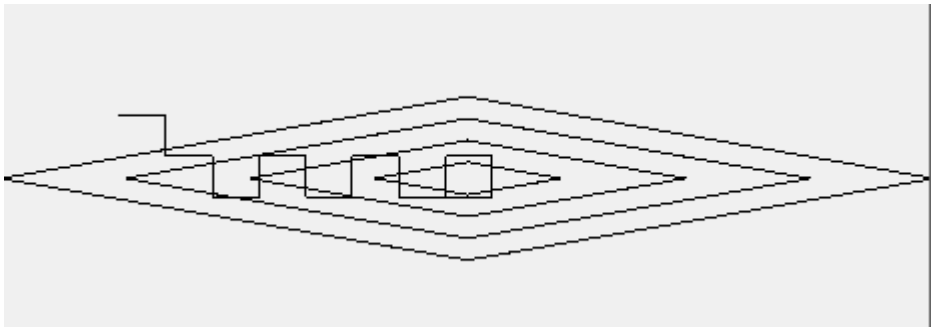

Рис 2.2.4 Траєкторії субградієнтних алгоритмів для ї функції що не має похідних (+алгоритм 1 – координатний крок з постійною відстанню між послідовними планами).

Рис 2.2.1 - Рис 2.2.4 – демонструють можливості субградієнтних алгоритмів з постійною відстанню між послідовними планами. Всі алгоритми дозволяють отримати розв'язок задачі оптимізації.

Наступний приклад демонструє можливості найскорішого спуску або метод лінійного пошуку. В околі точки А нема немає напрямків, що дозволяють зменшити значення функцій з точністю обчислень (крок менше допустимого).

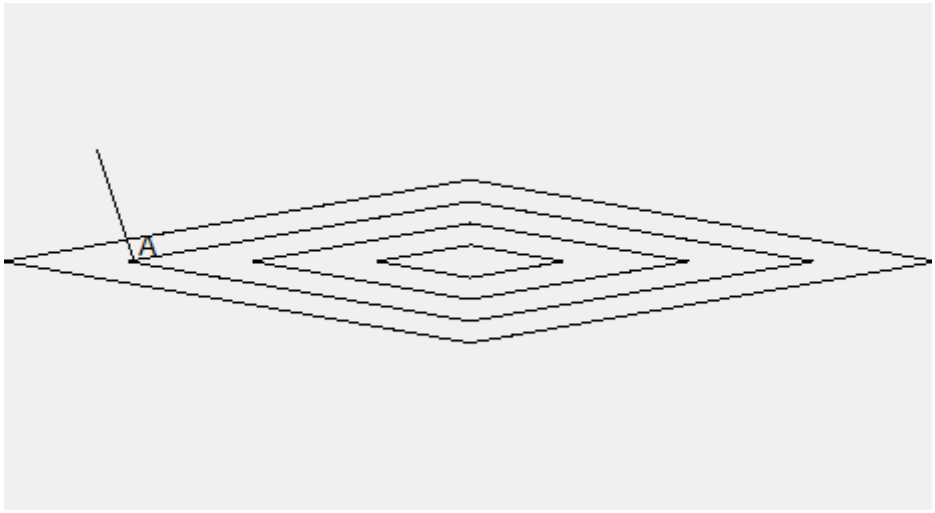

Рис 2.2.5. Метод лінійного пошуку для негладкої функції.

Метод лінійного пошуку – в цьому випадку стратегія полягає в тому, щоб на кожному кроці вибирати оптимальний крок α , який мінімізує функцію на напрямку поточного градієнта (субградієнта):

$$\alpha_t = \arg\min_{\alpha} f(\mathbf{x}_t - \alpha \nabla f(\mathbf{x}_t))$$

Метод виявився повністю непрацездатним так як і інші методи із вибором кроку..

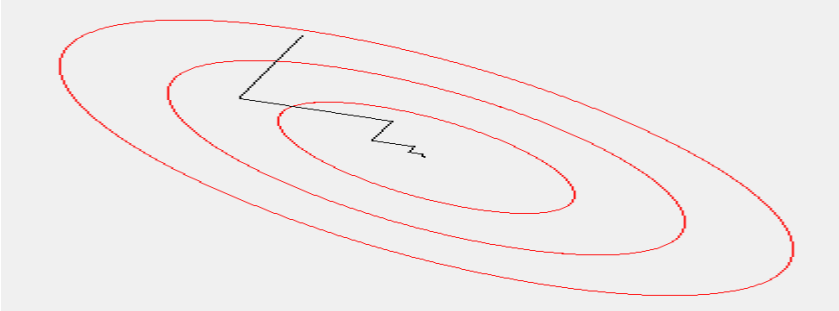

Рис 2.2.5. Метод лінійного пошуку для еліптичної функції.

Метод лінійного пошуку для еліптичної функції виявився одним із найкращих.

Субградієнтний алгоритм 2 з постійною відстанню між послідовними планами виявився найкращим для функції Розенброка.

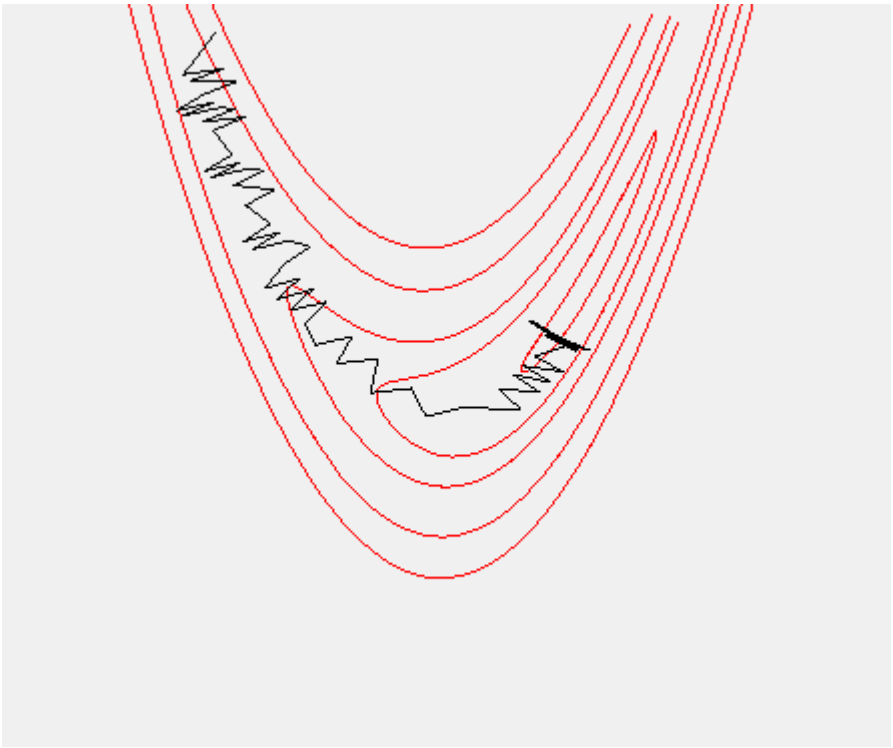

Рис 2.2.6 Траєкторії субградієнтних алгоритмів для функції Розенброка

(алгоритм 2 з постійною відстанню між послідовними планами)

Субградієнтний алгоритм 2 з постійною відстанню між послідовними продемонстрував переваги:

- Можливість оптимізації для недиференційованих функцій.
- Простота реалізації та універсальність.
- Здатність працювати з великими розмірами даних і складними структурами.

Суттєво гірший результат продемонстрував по координатний спуск для функції Розенброка. Будь який вибір кроку лише погіршував результат. Найкращими виявилися технології з постійним кроком але результат був суттєво гіршим ніж алгоритм 2 з постійною відстанню між послідовними планами.

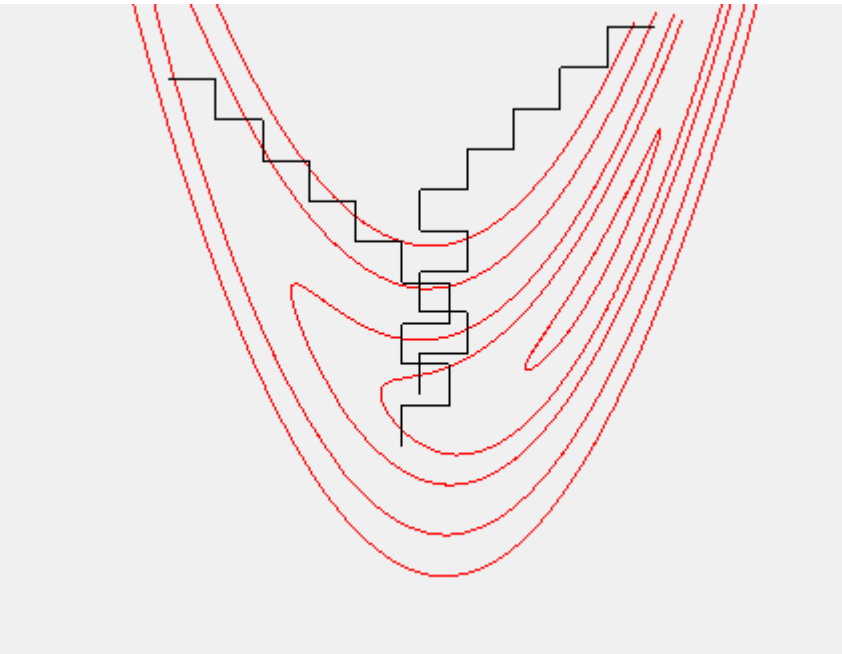

Рис 2.2.7 Нерелаксаційний по координатний спуск для функції Розенброка (крок 0.3)

Наступний малюнок демонструє залежність по координатного алгоритму 2 з постійною відстанню між послідовними планами від вибору кроку.

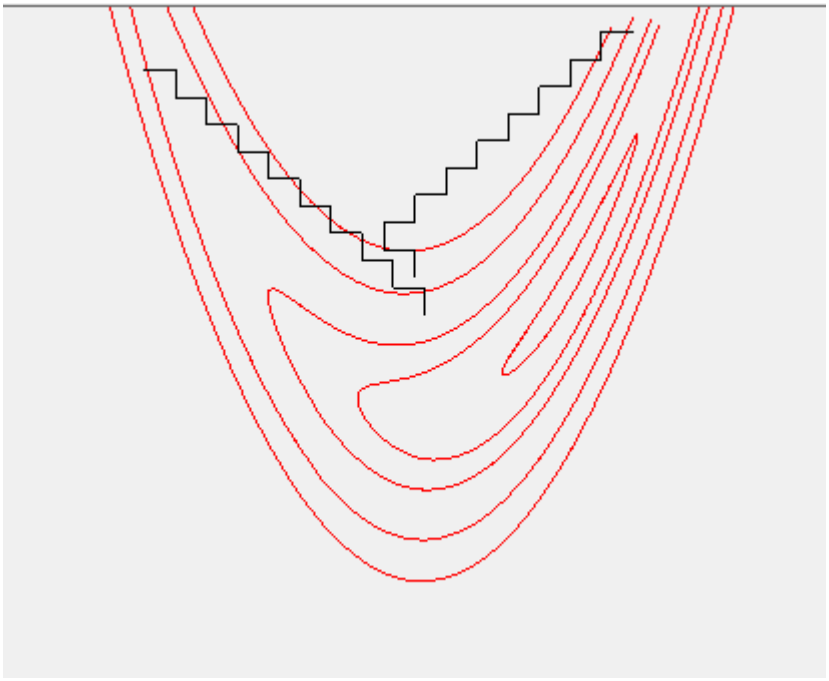

Рис 2.2.8 Нерелаксаційний по координатний не спуск для функції Розенброка (крок 0.3)

Рис 2.2.7 - Рис 2.2.8 демонструють залежність по координатного алгоритму 2 з постійною відстанню між послідовними планами від вибору кроку. Ця залежність є критичною.

2.3 Методи пошуку

Оптимізаційні алгоритми пошуку — це методи, які значення функцій застосовуються для знаходження найкращого рішення або оптимального результату для заданої оптимізаційної задачі. Принципи їх роботи залежать від того, чи йдеться про точні методи (які шукають точно оптимальне рішення) або приблизні методи (які можуть знаходити рішення, близьке до оптимального, за обмежений час). Ключові принципи роботи таких алгоритмів.

1. Пошук у просторі роз в’язків або планів.

Всі оптимізаційні алгоритми шукають найкраще рішення в деякому просторі планів, що представляє собою множину всіх можливих рішень задачі. Простір роз в’язків може бути:

Дискретним (наприклад, для задач комівояжера).

Неперервним (наприклад, для задач мінімізації функцій).

Алгоритм повинен ефективно шукати в цьому просторі та знаходити оптимум, максимізуючи або мінімізуючи цільову функцію.

2. Ітеративний процес

Більшість алгоритмів оптимізації працюють за принципом ітеративного пошуку, поступово наближаючись до оптимального рішення через серію кроків. Кожен крок змінює поточне рішення, базуючись на певних правилах, досягнення кращого результату або покращення цільової функції.

3. Перевірка умов оптимальності

Алгоритми часто використовують критерії оптимальності для визначення, коли слід припинити пошук або коли досягнуте оптимальне рішення (або близьке до оптимального):

Для точних методів: умови оптимальності можуть бути отримані з аналізу похідних (для задач мінімізації — коли похідна функції дорівнює нулю).

Для числових або наближених методів: пошук триває, доки не буде досягнута певна точність або кількість ітерацій.

4. Локальні і глобальні оптимуми

Локальний оптимум — це точка в просторі розв'язків, де функція не може бути покращена (зменшена або збільшена) шляхом незначних змін в околі цієї точки.

Глобальний оптимум — це точка в просторі розв'язків, де функція приймає своє найкраще можливе значення серед усіх варіантів.

Алгоритм повинен бути здатним або уникати локальних мінімумів, або швидко досягати їх, щоб забезпечити ефективне знаходження глобального оптимуму. Наприклад, алгоритми типу генетичних алгоритмів чи симульованого горіння мають вбудовані механізми для уникнення локальних мінімумів, досліджуючи більш широкі області простору розв'язків.

5. Рішення на основі оцінки (метод фітнесу)

Багато оптимізаційних алгоритмів використовують функцію оцінки або функцію фітнесу (fitness function) для порівняння різних можливих рішень. Ця функція визначає якість рішення в контексті оптимізаційної задачі:

Для задачі мінімізації — це буде функція, що дає менше значення для кращих рішень.

Для задачі максимізації — чим більше значення функції фітнесу, тим краще рішення. Алгоритм зазвичай працює, спрямовуючи пошук у бік більш високих (або низьких, залежно від задачі) значень цієї функції.

6. Ініціалізація та початкові умови

Ініціалізація алгоритму — це вибір початкових умов для пошуку, який може істотно вплинути на ефективність та результат алгоритму. В деяких випадках вибір початкової точки може бути важливим для досягнення глобального оптимуму.

Наприклад, у методах градієнтного спуску вибір початкової точки може визначати, чи алгоритм застрягне в локальному мінімумі.

7. Еволюційні методи та стохастичні підходи

У деяких оптимізаційних алгоритмах використовуються еволюційні або стохастичні принципи:

Стохастичні методи (наприклад, симульоване відпікання, метод рою часток) базуються на випадкових або ймовірнісних компонентах для дослідження простору розв'язків, що дозволяє уникнути застрягання в локальних мінімумів і забезпечити більш широке охоплення простору.

Еволюційні методи (наприклад, генетичні алгоритми, диференційна еволюція) використовують механізми, схожі на природний відбір, комбінуючи старі і нові рішення для створення більш якісних рішень.

8. Грід-пошук і стохастичні методи

Грід-пошук (grid search) полягає в тому, що всі можливі значення параметрів чи змінних перебираються за певною сіткою. Це точний метод, але зазвичай дуже витратний обчислень.

Стохастичні методи шукають рішення, використовуючи випадкові варіації і велику кількість пробних значень (наприклад, методи Монте-Карло або генетичні алгоритми).

9. Паралельний пошук

Деякі алгоритми, особливо в складних чи великих просторах розв'язків, застосовують паралельний пошук для підвищення ефективності. Це дозволяє одночасно досліджувати кілька напрямків пошуку та отримати результати швидше.

10. Перевірка умов зупинки

Кожен алгоритм оптимізації має критерії зупинки, за якими він припиняє пошук:

Досягнення заданої точності рішення.

Досягнення певної кількості ітерацій.

Випадковий чи стохастичний критерій для припинення на основі статистичних результатів.

11. Адаптивні стратегії

Деякі алгоритми застосовують адаптивні стратегії, де поведінка алгоритму змінюється в залежності від процесу пошуку:

Наприклад, у методах адаптивного градієнтного спуску (як Adam) використовуються змінні темпи навчання, що дозволяє алгоритму більш ефективно адаптуватися до конкретних особливостей задачі.

В особливо складних випадках використовують методи випадкового пошуку.

досліджувати простір рішень

Класифікація методів випадкового пошуку:

За рівнем використання випадковості

Повністю випадкові методи

Генерація рішень створюється повністю випадковим чином без урахування результатів попередніх ітерацій.

Приклад: класичний метод Монте-Карло.

Частково випадкові методи

Використовують випадковість у поєднанні з детермінованими правилами для підвищення ефективності пошуку.

Приклад: метод симульованого відпалу, стохастичний градієнт.

1) локальний випадковий пошук з поверненням;

2) локальний випадковий пошук за найкращою пробою; 3) локальний випадковий пошук за статистичним градієнтом;

4) глобальний випадковий пошук із незалежним вибором щільності розподілу пробних кроків;

5) генетичні алгоритми та еволюційне програмування. Розглянемо для прикладу основні елементи алгоритму локального випадкового пошуку (локального випадкового пошуку з поверненням, який на сьогодні є найбільш поширеним). Алгоритм полягає в наступному: 1-й крок. У випадковому напрямку здійснюється зміна значень параметрів, за якими відбувається пошук екстремуму

Кроки алгоритму:

1. Ініціалізація:

Вибрати початкову точку x

Встановити параметри: максимальну кількість ітерацій, радіус.

2. Пошук в околі:

На кожній ітерації згенерувати випадкову точку в околі, що визначає радіус

$$[x - R, x + R].$$

Обчислення припиняються згідно з критерієм або кількістю ітерацій.

Найкращі результати досягаються в поєднанні з регулярними методами, що досліджені в цій роботі. А найкращим вважають опуклий симплекс-методу для задач нелінійного програмування. Перевірка можливостей опуклого симплекс-методу для негладких задач проведена в цій роботі.

Алгоритм опуклого симплекс-методу для НЛП наступний.

Початкове рішення вибирається як одна з допустимих вершин многогранника, визначеного обмеженнями задачі.

Ця вершина повинна бути допустимою, тобто всі обмеження (як рівності, так і нерівності) повинні бути задоволені.

Пошук напрямку оптимізації:

Як і в класичному симплекс-методі, для кожної вершини шукається напрямок, в якому цільова функція покращується. Однак, на відміну від лінійного випадку, для нелінійних задач використовується методи градієнтного спуску чи методи Ньютона для визначення напрямку руху.

Якщо цільова функція опукла, то існує можливість рухатися в напрямку, що покращує функцію мети, і кожен крок може наближати до глобального мінімуму.

Оновлення рішення: Після вибору напрямку відбувається рух в сторону нової вершини багатогранника. У випадку нелінійних задач це може бути виконано через методи, що враховують кривизну функції мети та обмежень (наприклад, методи градієнтного спуску або методи другорядних умов).

Перевірка оптимальності:

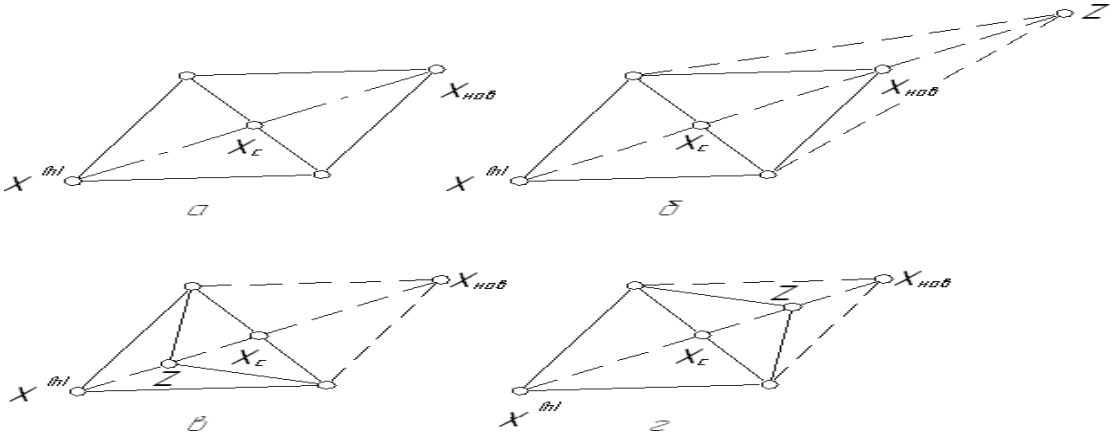

Рис. 2.3.1.Операціїї перетворення симплексу

а – проектування, б – розтяг, в – стиск, г – стиск

Оскільки задача нелінійна, необхідно перевіряти, чи не покращується більше цільова функція на поточному кроці. Якщо напрямок оптимізації більше не може покращити функцію мети (це можна визначити через умови

першого або другого порядку оптимальності), то знаходиться оптимальне рішення.

Повторення ітерацій:

Кроки повторюються, поки не буде досягнуто оптимального рішення або мінімуму функції мети.

Як показали чисельні експерименти метод проектування багатокутника є ефективним для задач опуклого програмування як гладких так і ні.

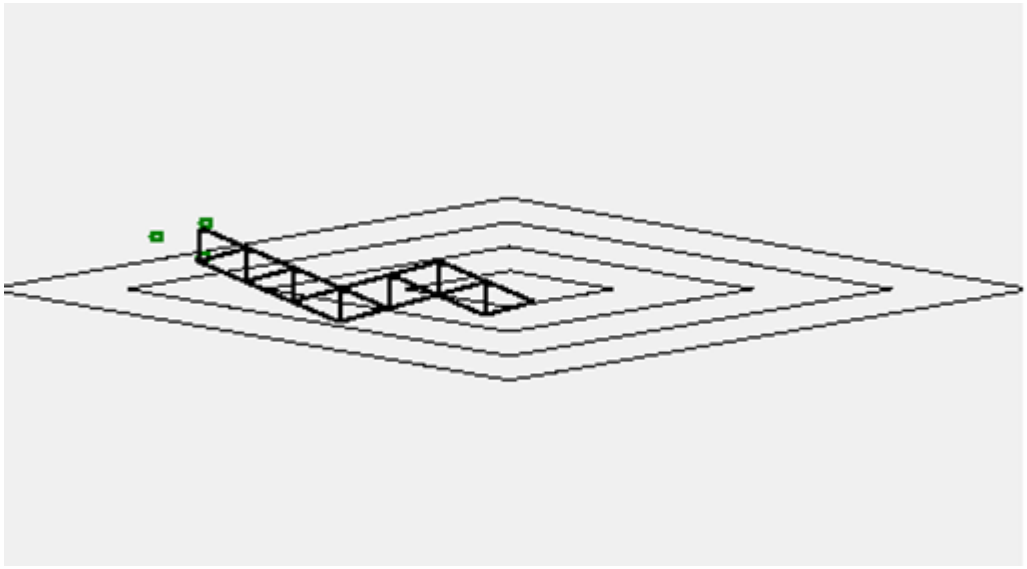

Рис. 2.3.2. Траєкторія методу проектування симплексу для негладкої задачі.

Для функції Розенброка модифікація методу проектування симплексу з деформуванням багатокутника виявилась кращою ніж методи з регулюванням кроку але гіршою ніж метод нерелаксаційного узагальненого градієнту.

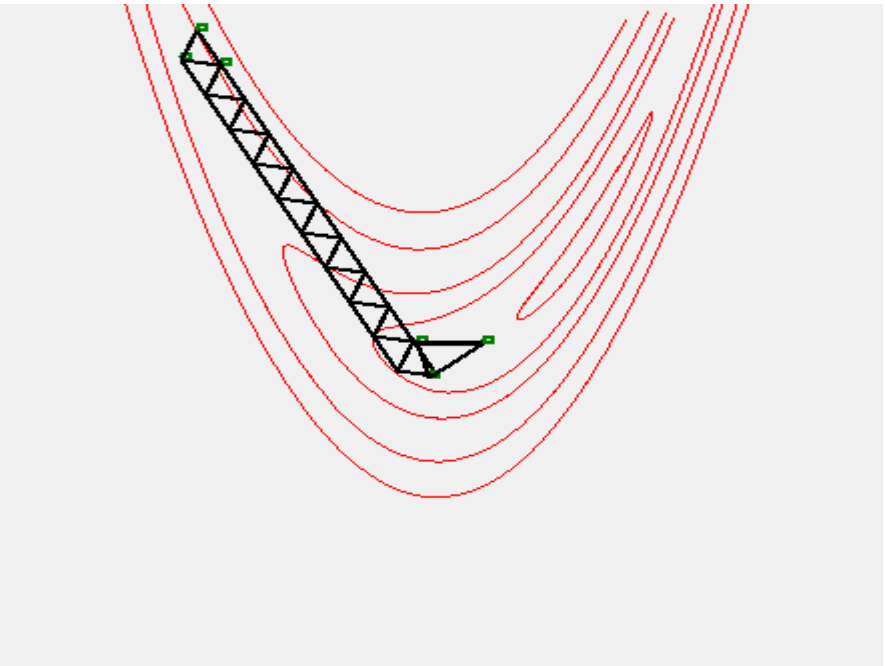

Рис. 2.3.2. Траєкторія методу проектування симплексу для неопуклої функції Розенброка.

Траєкторія методу проектування симплексу для неопуклої функції Розенброка з будь якими параметрами і модифікаціями методу демонструє недостатню точність

2.4. Висновки до розділу

Мета дослідження – визначити та порівняти можливості методів, що дозволяють працювати з функціями, що мають кути, розриви або інші складні особливості та створення програмного засобу, що забезпечить інформаційну допомогу та формулювання рекомендацій по визначенню параметрів широкого кола алгоритмів безумовної опуклої оптимізації. Основною метою дослідження є визначення тих методів та технологій, що можуть бути практично застосовані для мінімізації функцій, що не мають похідних в деяких точках.

Для досягнення поставленої мети в роботі були вирішені наступні проблеми:

Було виконано дослідження особливості актуальних задач дослідження операцій, що потребують використання моделей, які сформулюються у

вигляді задач нелінійного програмування з допомогою функцій, що не мають похідних в будь яких точках.

Було виконано дослідження можливостей програмних реалізацій найбільш поширених нерелаксаційних методів безумовної оптимізації та графічних засобів, які дозволяють проектувальникам формувати обґрунтовані рішення для вибору задач, що можуть бути ефективно досліджені методами негладкої оптимізації. Розробити і реалізувати програмно концептуальні та логічні моделі що є необхідними для відображення та порівняння результатів;

В роботі для перевірки якості алгоритмів використовується 4 типи тестових завдань. Для цих функцій були розроблені програми, що відображають лінії рівня. На наочних графічних відображеннях результатів чисельних експериментів продемонстровані переваги нерелаксаційних алгоритмів для задач негладкої оптимізації.

Чисельні експерименти демонструють можливості субградієнтних алгоритмів з постійною відстанню між послідовними планами. Всі алгоритми дозволяють отримати розв’язок всіх типів задач оптимізації

Субградієнтний алгоритм з постійною відстанню між послідовними планами продемонстрував переваги:

- Можливість оптимізації для недиференційованих функцій.
- Простота реалізації та універсальність.
- Здатність працювати з великими розмірами даних і складними структурами.

Цей алгоритм в якості множин узагальненого градієнта точки x_0 використовує градієнт точки яка максимально наближена до точки x_0 (в околі що визначається точністю обчислень)

РОЗДІЛ 3 ОБҐРУНТУВАННЯ ЗАСТОСОВАНИХ ПРОГРАМНИХ ТЕХНОЛОГІЙ РОЗРОБКИ

3.1. Обґрунтування програмних технологій

Програмний засіб розроблений методами шаблонів та інструментів Microsoft Visual Studio. Базовий засіб програмування C#.

Microsoft Visual Studio — це потужне середовище для розробки програмного забезпечення, яке підтримує широкий спектр мов програмування та платформ. Для програм, що потребують великої кількості обчислень і потужного графічного це потужне середовище надає значні переваги.

1. Багатофункціональність. Visual Studio підтримує розробку програм для різних платформ: Windows, macOS, Linux, а також мобільні платформи (через Xamarin). Це дозволяє створювати як настільні, так і веб-додатки, ігри, а також мобільні додатки на одному середовищі.

2. Підтримка багатьох мов програмування. Visual Studio підтримує різні мови програмування, такі як C#, C++, Visual Basic, F#, Python, JavaScript, TypeScript та інші. Це дозволяє використовувати одну платформу для розробки проектів на різних мовах.

3. Потужні інструменти для налагодження. Visual Studio має розширені інструменти для налагодження коду, включаючи точки зупину, покрокове виконання, перегляд змінних і стека викликів, профілювання продуктивності, а також віддалене налагодження для додатків, що працюють на інших машинах.

4. Інтеграція з Git та іншими системами контролю версій. Visual Studio має вбудовану інтеграцію з Git, що дозволяє зручно працювати з репозиторіями без виходу з середовища розробки. Це також включає підтримку GitHub та Azure DevOps.

5. Інтелектуальний автодоповнення та рефакторинг Інструмент IntelliSense автоматично пропонує автодоповнення для коду, параметрів

функцій, методів, класів тощо. Це значно підвищує продуктивність, особливо при роботі з великими кодовими базами. Крім того, Visual Studio підтримує рефакторинг коду, що допомагає підтримувати його чистоту та організованість.

6. Підтримка розширень. Visual Studio має величезний набір доступних розширень, що дозволяють додавати нові функціональні можливості, підтримку мов програмування, інструменти для тестування та багато іншого.

7. Інтеграція з Azure. Visual Studio має пряме з'єднання з Microsoft Azure, що спрощує розробку, деплоймент і керування хмарними додатками та сервісами.

8. Тестування та CI/CD. Visual Studio надає потужні інструменти для юніт-тестування, інтеграційного тестування та автоматизації процесів CI/CD (безперервна інтеграція/безперервний деплоймент) за допомогою Azure DevOps або інструментів сторонніх розробників.

9. Дружній інтерфейс та зручність використання. Інтерфейс Visual Studio зрозумілий і адаптований для користувачів різного рівня досвіду. Він підтримує шаблони проектів, що дозволяють швидко стартувати нові розробки, а також візуальні інструменти для розробки інтерфейсів (наприклад, для WPF, WinForms).

10. Продуктивність та оптимізація. Visual Studio оптимізовано для роботи з великими проектами і надає можливість роботи з величезними кодовими базами без значних затримок. Профілювання та аналіз продуктивності дозволяють ефективно виявляти та усувати проблеми з швидкодією.

11. Безкоштовні варіанти. Для індивідуальних розробників або малих команд доступна безкоштовна версія Visual Studio — Visual Studio Community, яка містить більшість основних функцій середовища, що робить її доступною для широкого кола користувачів.

13. Інтеграція з іншими продуктами Microsoft. Visual Studio прекрасно інтегрується з іншими інструментами та технологіями Microsoft, такими як

SQL Server, Power BI, Office 365, а також з такими продуктами, як Microsoft Teams і SharePoint.

Microsoft Visual Studio — це потужний, універсальний інструмент для розробки, який пропонує великий набір функцій для будь-якого типу програмування, будь то створення веб-додатків, настільних програм чи мобільних додатків. Він особливо корисний для розробників, які працюють у екосистемі Microsoft, або тих, хто шукає зручне середовище для розробки.

C# (C-Sharp) — це мова програмування, розроблена компанією Microsoft, яка є частиною платформи .NET. Вона має багато переваг, що роблять її популярною серед розробників.

1. Простота вивчення та використання

C# має синтаксис, який є досить простим для вивчення, особливо для тих, хто вже знайомий з іншими мовами програмування, такими як Java або C++. Оскільки C# є об'єктно-орієнтованою мовою, це дозволяє краще структурувати код і полегшує підтримку великих проектів.

2. Підтримка об'єктно-орієнтованого програмування (ООП)

C# підтримує всі основні принципи ООП, такі як наслідування, поліморфізм, інкапсуляція та абстракція. Це дозволяє створювати модульні, гнучкі та масштабовані додатки.

3. Потужна інтеграція з .NET Framework і .NET Core

C# розроблений для роботи з платформою .NET, яка надає великий набір бібліотек для різних типів програм. Це включає підтримку веб-додатків (ASP.NET), роботу з базами даних (Entity Framework), а також інструменти для розробки десктопних та мобільних додатків. Завдяки .NET Core (тепер .NET 5+) C# став справжньою кросплатформенною мовою, яка підтримує Windows, macOS та Linux.

4. Підтримка асинхронності та багатозадачності

C# має потужні механізми для асинхронного програмування (через async/await), що дозволяє створювати додатки з високою продуктивністю та

швидким відгуком. Це особливо корисно для розробки веб-додатків, які повинні працювати з великою кількістю одночасних запитів.

5. Безпека типів

C# є сильно типізованою мовою, що означає, що багато помилок можна виявити на етапі компіляції, ще до запуску програми. Це зменшує кількість помилок у коді і підвищує стабільність додатків.

6. Підтримка сучасних технологій.C# активно підтримує нові технології та парадигми програмування. Це включає:

LINQ (Language Integrated Query) — потужний механізм для запитів до колекцій даних.

Lambda-вирази — дозволяють використовувати компактні функціональні стилі програмування.

Tuples — для зручного повернення кількох значень з методу.

Pattern Matching — покращене розпізнавання шаблонів для складних типів даних.

Це дозволяє писати більш виразний та компактний код.

7. Кросплатформеність. Завдяки .NET Core (тепер .NET 5+), C# став кросплатформенною мовою, що дозволяє створювати додатки не тільки для Windows, але й для macOS і Linux. Розробники можуть створювати серверні, веб і десктопні додатки для різних операційних систем, не змінюючи значно код.

8. Підтримка сучасних інструментів і середовищ розробки. C# має відмінну інтеграцію з Microsoft Visual Studio, яке є одним з найпотужніших середовищ для розробки програм. Це середовище надає зручні інструменти для автодоповнення коду, налагодження, рефакторингу, тестування та інтеграції з системами контролю версій.

9. Велика спільнота та документація. C# має велику спільноту розробників, багато відкритих бібліотек і ресурсів. Оскільки мова активно підтримується Microsoft, є велика кількість навчальних матеріалів,

документації та прикладів коду, що допомагають швидше освоїти мову та вирішувати проблеми.

10. Широке застосування в індустрії. C# активно використовується в багатьох галузях- розробка веб-додатків (ASP.NET Core, Blazor). розробка ігор (Unity — одна з найбільш популярних ігрових платформ використовує C#). мобільні додатки (Xamarin дозволяє створювати мобільні додатки для iOS та Android).Робота з базами даних (Entity Framework). Десктопні додатки (WPF, WinForms).Вбудовані системи та IoT.

11. Підтримка інтероперабельності. C# дозволяє інтегруватися з іншими мовами програмування, такими як C++, COM-об'єкти, а також може використовувати сторонні бібліотеки через інтерфейси і P/Invoke. Це робить мову універсальною та гнучкою для інтеграції з іншими системами.

12. Стабільність та підтримка. Як частина екосистеми .NET, C# отримує регулярні оновлення та покращення від Microsoft. Це дає впевненість, що мова буде підтримуватися в майбутньому, і її можливості будуть розширюватися відповідно до вимог ринку.

13. Швидкість виконання.. Програми на C# компілюються в байт-код, який виконується на .NET CLR (Common Language Runtime). Завдяки JIT-компіляції (Just-In-Time), C# може досягти високої продуктивності, особливо в порівнянні з інтерпретованими мовами.

C# є потужною, зручною та ефективною мовою програмування, яка підтримує сучасні парадигми розробки, пропонує високий рівень безпеки типів та продуктивності. Вона ідеально підходить для розробки кросплатформенних додатків, веб-сервісів, ігор і мобільних додатків. C# також надає можливість працювати з багатими екосистемами .NET, що спрощує розробку та підтримку програмного забезпечення.

3.2 Програмна реалізація застосування

Необхідними для дослідження і демонстрації можливостей алгоритмів негладкої оптимізації є наступні програмні завдання.

1.Програмування та графічне надання зображень ліній рівня для функцій, які дозволяють дослідити та продемонструвати особливості задач негладкої та неопуклої оптимізації. Порівняти особливості застосування методів з зручними задачами гладкого та опуклого програмування. Типові функції, відповідно до загально прийнятих стандартів, можуть бути використані для демонстрації типових варіантів нерелаксаційних алгоритмів..

2. Програмування найбільш надійних та поширених алгоритмів які теоретично можуть бути використані для оптимізації негладких та неопуклих функцій (субградієнтні методи та методи пошуку).

3. Програмування засобів для графічної візуалізації ліній, що визначають траєкторію наближення до екстремуму алгоритмів (послідовність кроків).

Результати роботи програми наведені та проаналізовані в розділі 2.

Для реалізації використані шаблони Windows Forms — це традиційний фреймворк для створення графічних інтерфейсів користувача (GUI) для Windows-додатків. Цей фреймворк дозволяє швидко створювати прості і багатофункціональні десктопні додатки за допомогою компонентів, таких як кнопки, текстові поля, панелі, тощо.

Головна форма програми наведена на малюнку 3.2.1. Форма призначена для визначення параметрів функцій, що досліджуються та параметрів методів оптимізації.

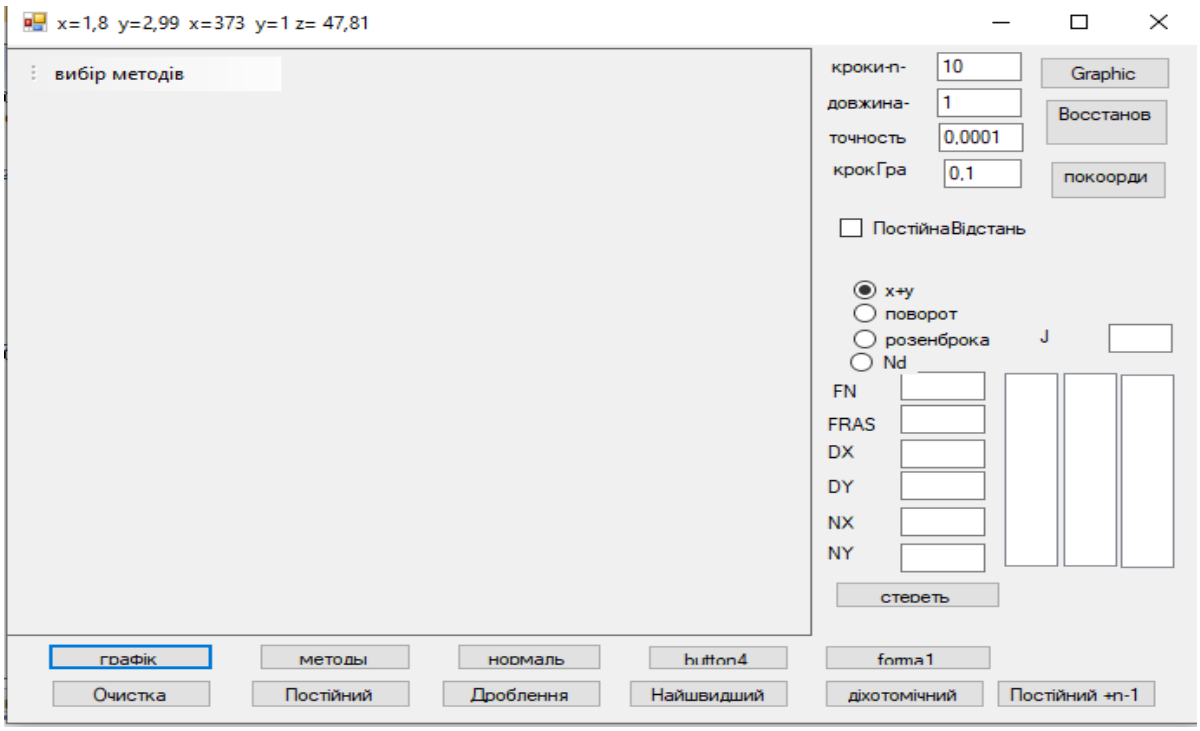

Рис. 3.2.1. Головна форма програми

Форма Рис. 3.2.1 призначена для визначення параметрів методів оптимізації. Програмні можливості, що реалізовані в класі форми, дозволяють порівняти можливості різних методів для задач одного класу. Результати порівняння наведені в розділі 2. Приклад наведений нижче.

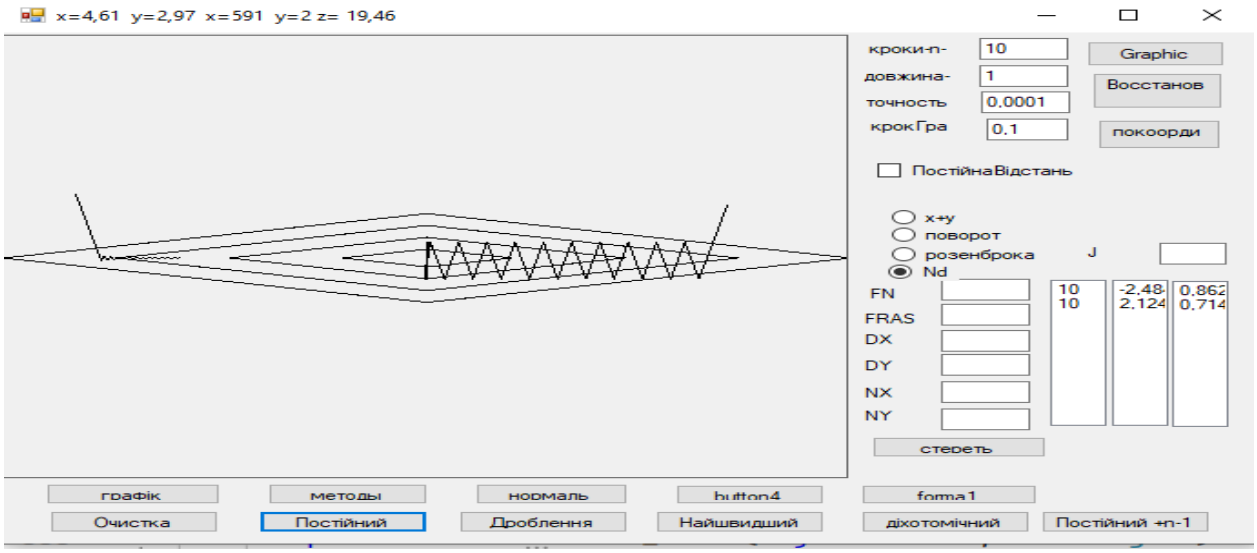

Рис. 3.2.2. Метод з постійним кроком і дробленням кроку

Рис. 3.2.2. демонструє переваги нерелаксаційних методів для оптимізації негладких функцій.

Наступний малюнок дозволяє порівняти метод проектування багатогранника і найскоріший спуск.

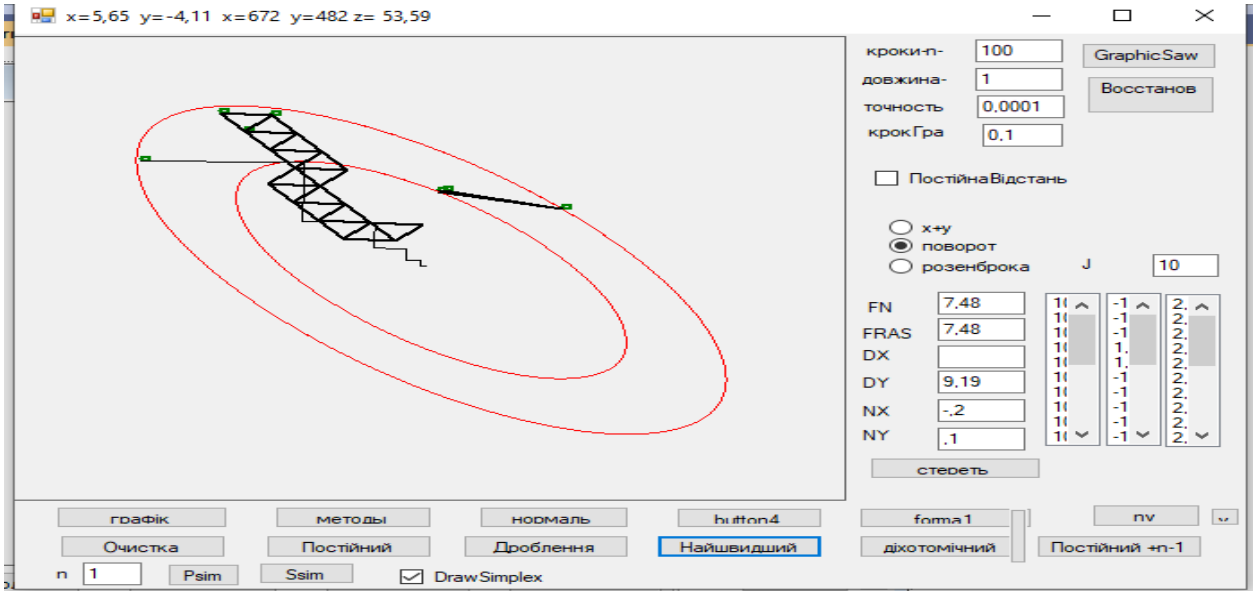

Рис. 3.2.3. Метод пошуку з постійним кроком і найскоріший спуск.

Останній малюнок демонструє, що методи пошуку для опуклих задач не є ефективними. Тест програми спуску наведений нижче.

```
int n,j=1,i,lc;
            Pen mypen = new Pen(Color.Black, 0.01F);
n = Convert.ToInt16(textBox1.Text);
double  x2tmp, y2tmp,xx2,yy2;
double fz=fn,tv=tvp*10;
    x2tmp = xnn; y2tmp = ynn;
//
    normal(ref  x2tmp, ref  y2tmp);
    dpoint p1 = new dpoint(xnn, ynn), p2 = new dpoint(2*x2tmp, 2*y2tmp),p
    for (lc=1;lc<=n; lc++)
    {
        x2tmp = xnn; y2tmp = ynn; xx2 = xnn; yy2 = ynn;
        //повертає нормаль до град -дотичну до лінії рівня
        normal(ref  x2tmp, ref  y2tmp);
        for (j = 1; j <=19; j=j+3)
        {
            xx2 = xnn + (x2tmp * step)/j;
            yy2 = ynn + (y2tmp * step)/j;

            if (Math.Abs(Math.Abs(fz) - Math.Abs(f(xx2, yy2))) < tv) brea

        }
        //повертає уточнену точку на нормалі до град
         stepnormal(ref xx2, ref yy2);
        mg.DrawLine(new Pen(Color.Red, 0.01F),
        Convert.ToSingle(xnn), Convert.ToSingle(ynn),
    Convert.ToSingle(xx2), Convert.ToSingle(yy2));
        mg1.DrawLine(new Pen(Color.Red, 0.01F),
 Convert.ToSingle(xnn), Convert.ToSingle(ynn),
 Convert.ToSingle(xx2), Convert.ToSingle(yy2));
        xnn = xx2;
        ynn = yy2;
```

Незважаючи на цей приклад, метод Нелдера—Міда залишається корисним інструментом для чисельної оптимізації, особливо коли похідні функції невідомі.

3.3. Висновки до розділу 3

Цей підрозділ роботи посвячений обґрунтування зробленого вибору програмних технологій для функцій програмного засобу.

Необхідними для дослідження і демонстрації можливостей алгоритмів негладкої оптимізації потрібні наступні можливості:

1.Програмування та графічне надання зображень ліній рівня для функцій, які дозволяють дослідити та продемонструвати особливості задач негладкої та неопуклої оптимізації. Порівняти особливості застосування методів з зручними задачами гладкого та опуклого програмування. Типові функції, відповідно до загально прийнятих стандартів, можуть бути використані для демонстрації типових варіантів нерелаксаційних алгоритмів..

2. Програмування найбільш надійних та поширених алгоритмів які теоретично можуть бути використані для оптимізації негладких та неопуклих функцій (субградієнтні методи та методи пошуку).

3. Програмування засобів для графічної візуалізації ліній, що визначають траєкторію наближення до екстремуму алгоритмів (послідовність кроків).

Для реалізації цих функцій обґрунтований вибір сучасної технології програмування шо надає Visual Studio 2022. В розділі визначені засоби реалізації функцій програмної системи.

ВИСНОВКИ

В кваліфікаційній роботі розглянуті теоретичні і практичні аспекти програмної реалізації алгоритмів безумовної оптимізації нульового порядку (методів пошуку) Об'єктом є дослідження нерелаксаційних методів негладкої безумовної оптимізації та методів пошуку. Досліджені актуальні задачі, моделі для побудови та порівняння необхідних оптимізаційних алгоритмів для відомих задач мінімізації невизначених або недиференційованих функцій. Розглянуті задачі нелінійного та не опуклого програмування. Важливі для наочності методи згорнення обчисленої інформації включають засоби побудови ліній рівня мінімізованих функцій для представлення результатів.

Описаний в даній роботі програмний продукт для визначення ефективності методів мінімізації невизначених або функцій, що не мають похідних в будь яких точках, є засобом, що може бути використаними для важливих задач дослідження операцій та дидактичним засобом. Розроблені програмні засоби побудови ліній рівня для типових, з точки зору оптимізації класів функцій. Програмна реалізація алгоритмів дозволила провести експериментальне дослідження та порівняння основних модифікацій алгоритмів безумовної оптимізації негладких задач (методи, що можуть бути застосовані до мінімізації недиференційованих функцій). Ці модифікації методів оптимізації є важливими і мають принципове значення при вирішенні задач навчання нейронних мереж та багатьох різних задач оптимального планування і проектування. Програмний засіб дозволяє наочно проаналізувати особливості алгоритмів мінімізації недиференційованих функцій з допомогою графічних засобів. Субградієнтний алгоритм з постійною відстанню між послідовними планами продемонстрував переваги:

Можливість оптимізації для недиференційованих функцій.

Простота реалізації та універсальність.

Здатність працювати з великими розмірами даних і складними структурами.

Цей алгоритм в якості множин узагальненого градієнта точки x_0 використовує градієнт точки яка максимально наближена до точки x_0 (в околі що визначається точністю обчислень)

Була досягнута мета дослідження – порівняти можливості методів, що дозволяють працювати з функціями, що мають кути, розриви або інші складні особливості. Створення програмного засобу, що забезпечить інформаційну допомогу та формулювання рекомендацій по визначенню параметрів широкого кола алгоритмів безумовної опуклої оптимізації.

ДЖЕРЕЛА ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Щербань В.Ю., Краснитський С. М. Астістова Т. І. , Яхно В. М. Методи представлення,збереження та аналізу даних інформаційних систем –К. “Фастбінд Україна”, 2023, 480 сторінок.
2. Боровик О. Л. Дослідження операцій в економіці : навч. посіб./ О. Л. Боровик, Л. В. Боровик. – Київ : Центр учбової літератури, 2017. –424 с.
3. Боровська Т. М. Основи теорії управління та дослідження операцій :навч. посіб. / Т. М. Боровська, І. С. Колеснік, В. А. Северілов. – Вінниця :УНІВЕРСУМ-Вінниця, 2018. – 242 с.
4. Голіков А. П. Економіко-математичне моделювання світогосподарських процесів : навч. посіб. / А. П. Голіков. – 3-тє вид., перероб. і допов. – Київ : Знання, 2019. – 222 с.
5. Економіко-математичне моделювання : навч. посіб. / Т. С. Клебанова, О. В. Раєвнєва, С. В. Прокопович та ін. – Харків : ВД "ІНЖЕК", 2010. – 352 с.
6. Сікора Я. Б., Щехорський А.Й., Якимчук Б.Л.Методи оптимізації та дослідження операцій [Текст] : навчальний посібник /Укладачі: Я. Б. Сікора, А.Й. Щехорський, Б.Л. Якимчук. – Житомир: Вид-во ЖДУ ім. Івана Франка, 2019. – 148 с
7. Рогоза М. Є. Нелінійні моделі та аналіз складних систем: навч. посіб. в 2 ч. Ч. 1 / М. Є. Рогоза, С. К. Рамазанов, М. К. Мусаєва. – Полтава : РВВ ПУЕТ, 2017. – 300 с.
8. Роман Л. Л. Дослідження операцій. Курс лекцій / Л. Л. Роман. –Львів : Видавництво Тараса Сороки, 2018. – 272 с.
9. Порівняльний аналіз сучасних технологій зберігання даних у хмарних обчисленнях http://perspectives.pp.ua/index.php/nts/article/view/14408/14478
10. Сучасні можливості інформаційного моделювання з використанням цифрових методів для побудови цифрової моделі http://perspectives.pp.ua/index.php/nts/article/view/13505/13570
11. Methods and algorithms of optimization in computer engineering: review and comparative analysis

https://dm.ageditor.ar/index.php/dm/article/view/257

12. Вітлінський В. В., Савіна С. С. Математичні методи та моделі: оптимізація : навч.посіб. Київ : КНЕУ, 2021. 303 с.

13. Воронков О. О. Оптимізаційні методи і моделі : конспект лекцій з курсу. Харків : ХНУМГ ім. О. М. Бекетова, 2022. 110 с.

14. Дослідження операцій в економіці : підруч. / О. І. Черняк та ін. ; ред. О. І. Черняка. Миколаїв : МНАУ, 2020. 398 с.

15. Дослідження операцій : метод рекоменд. для самост.роботи студентів ден. та заоч. форм навчання напряму підготов. /О. В. Шебаніна та ін. Миколаїв : МНАУ, 2014. 98 с.

16. Дослідження операцій : курс лекцій / О. В. Шебаніна та ін. Миколаїв : МНАУ, 2015. 248 с.

17. Дослідження операцій в економіці : підруч. / І. К Федоренко. та ін. ; за ред. І. К. Федоренка, О. І. Черняка. Київ : Знання, 2007. Київ : Знання, 2017. 558 с.

18. Дослідження операцій. Практичний курс : навч. посіб. / В. Є. Березовський та ін. Умань : Видавець «Сочінський», 2019.238 с.

19. Зайченко О. Ю., Зайченко Ю. П. Дослідження операцій : збірник задач. Київ : Видавничий дім «Слово», 2007. 472 с.

20. Савчук, О. О. Дослідження операцій: Теорія та методи оптимізації. Київ: Видавництво "Лібра".2019.258 с.

21. Дубровін, О. М., Різник, В. І. . Дослідження операцій: Моделі та методи. Харків: ХНУРЕ. 2021.268 с.

22. Молодов, О. П. . Основи дослідження операцій. Львів: Світ. 2019.238 с.

Taha, H. A. (2017). Operations Research: An Introduction. 10th Edition. Pearson. 2019.628 с.

23. Дослідження операцій [навчальний посібник] / МеньшиковаО.В., Чмир О.Ю., Карабин О.О. – Львів : ЛДУ БЖД, 2019. – 196с.

24. Дослідження операцій : конспект лекцій / О. В. Шебаніна, В. П. Клочан, І. В. Клочан та ін. – Миколаїв : МНАУ, 2021. – 150 с

25. Дослідження операцій та методи оптимізації : практикум : у 2-х ч. Частина 1 [Електронний ресурс] / Л. М. Малярець, І. Л. Лебедєва, Л. О. Норік. – Харків : ХНЕУ ім. С. Кузнеця, 2017. – 169 с.

26. Дослідження операцій та методи оптимізації: лабораторний практикум в середовищі MATLAB. Малярець, Л. М., Ковальова, К. О., Малярец, Л. М., Ковалева, Е. А. (2018).

29. Дослідження операцій та методи оптимізації: методичні рекомендації до практичних завдань для студентів усіх спеціальностей першого (бакалаврського) рівня / уклад. С. В.

Прокопович, О. В. Панасенко, Л. О. Чаговець. – Харків : ХНЕУ ім. С. Кузнеця, 2019. – 64 с.

30. Дослідження операцій. Конспект лекцій / Уклад.: О.І. Лисенко, І.В. Алєксєєва, – К: НТУУ «КПИ», 2016. – 196 с.

31. Дослідження операцій. Частина 4 : Нелінійне програмування : підручник / М. Я. Бартіш, І. М. Дудзяний. – Львів : ЛНУ ім. Івана Франка, 2019. - 208 с.