НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра системного проектування

До захисту допущено:
Завідувач кафедри
_____ Вадим МУХІН
«___»_____2023 р.

Дипломна робота
на здобуття ступеня бакалавра

за освітньо-професійною програмою

«Інтелектуальні сервіс-орієнтовані розподілені обчислювання»

спеціальності 122 «Комп'ютерні науки»
на тему: «Використання сурогатних моделей для оптимізації систем керування складними об’єктами»

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут прикладного системного аналізу

Кафедра системного проектування

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Інтелектуальні сервіс-орієнтовані розподілені обчислювання»

ЗАТВЕРДЖУЮ

Завідувач кафедри
_____ Вадим МУХІН
«___»_____2023 р.

ЗАВДАННЯ

на дипломну роботу студенту

Попелюка Миколи Миколайовича

1. Тема роботи «Використання сурогатних моделей для оптимізації систем керування складними об’єктами», керівник роботи Чкалов О.В. к.т.н., доцент, затверджені наказом по університету від «06» червня 2022 р. №906-с

2. Термін подання студентом роботи 18.06.2023

3. Вихідні дані до роботи: Математичні моделі систем керування часто виявляються досить складними, що робить обчислення цільової функції занадто трудомістким. В цьому випадку рішенням може бути використання так званої сурогатної моделі, що імітує з певною точністю поведінку основної моделі і в той же час є значно простішою з точки зору обчислювальної трудомісткості

4. Зміст роботи

1. Ознайомитись з постановкою задач одно та багатокритеріальної оптимізації
2. Вивчити методику побудови сурогатної моделі об’єкта оптимізації.
3. Програмно реалізувати алгоритм побудови та налаштування сурогатних моделей.
4. Виконати тестування розробленої програмної реалізації
5. Виконати розв’язування тестової задачі оптимізації із використанням сурогатної моделі

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н. В., к.е.н., доцент		

7. Дата видачі завдання 03.02.2022

Календарний план

з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
	Отримання завдання	03.02.2022	
	Збір інформації	25.02.2022	
	Ознайомлення з літературою і підготовка теоретичної частини роботи	15.03.2022	
	Аналіз вимог завдання, вибір Методів	15.04.2022	
	Створення і тестування вимог на програмний проект	20.05.2022	
	Розробка економічної частини дипломного проекту	26.05.2022	
	Оформлення дипломної роботи	03.06.2022	
	Отримання допуску до захисту та подача роботи в ДЕК	12.06.2023	

Студент Попелюк Микола

Керівник Олексій ЧКАЛОВ

АНОТАЦІЯ

Метою роботи є дослідження алгоритмів сурогатного моделювання та їх практичне використання для оптимізації складного об’єкту.

В ході роботи було досліджено причини та історію виникнення сурогатного моделювання, а також його типи. Було досліджено сучасні модифікації алгоритмів, а також їх ефективність та приклади їх реалізації.

В результаті роботи було проведено тестування деяких алгоритмів на прикладі складного об’єкту та наочно продемонстровано актуальність такого роду підходів до оптимізації. Кінцевим результатом є оптимізована модель з можливістю встановлення необхідної точності.

Загальний обсяг роботи: 92 сторінки, 18 ілюстрацій, 7 таблиць, 33 посилання.

Ключові слова: сурогатне моделювання, оптимізація, управління складними об’єктами, моделювання.

ABSTRACT

The purpose of this work is to study surrogate modeling algorithms and their practical use for optimizing a complex object.

In the course of the work, the reasons and history of surrogate modeling, as well as its types, were investigated. Modern modifications of algorithms, as well as their effectiveness and examples of their implementation, were studied.

As a result of the work, some algorithms were tested on the example of a complex object and the relevance of such optimization approaches was clearly demonstrated. The final result is an optimized model with the ability to set the required accuracy.

Total volume of work: 92 pages, 18 illustrations, 7 tables, 33 references.

Keywords: surrogate modeling, optimization, management of complex objects, modeling.

ЗМІСТ

ВСТУП

У сфері інженерії, оптимізації та аналізу даних складність реальних систем часто створює значні проблеми. Ці системи, починаючи від складних фізичних процесів і закінчуючи комп'ютерними симуляціями, можуть вимагати значних обчислювальних ресурсів і часу для точної оцінки. Не зважаючи на ріст обчислювальних можливостей, у випадку складних об’єктів, процес їх оптимізації все ще може займати дні та тижні, у зв’язку з необхідністю багаторазової їх симуляції. У таких сценаріях розробка сурогатних моделей стає цінним підходом для наближення поведінки цих складних систем.

Для вирішення цієї проблеми було розроблено багато алгоритмів. В даній роботі розглядається один із таких способів – сурогатне моделювання. Перевагою цього підходу є можливість застосування менш точних моделей з заданою, для конкретної задачі, точністю, зі зменшенням її обчислювальної вартості. Створюючи спрощені математичні або обчислювальні моделі на основі обмежених вхідних-вихідних даних, сурогатне моделювання дозволяє аналітикам і дослідникам представляти поведінку складних систем у більш керованій формі. Цей підхід дозволяє уникнути повторних оцінок дорогих в обчислювальному плані симуляцій або експериментів, що, відповідно, зменшує обчислювальні витрати і послаблює часові обмеження.

Основною метою цієї роботи є розгляд сучасних рішень для оптимізації систем керування складних об’єктів за допомогою сурогатного моделювання, а також реалізація та тестування таких методів на прикладі конкретної задачі оптимізації моделі лопатки турбіни.

При виконанні дослідження будуть розглянуті різні методи сурогатного моделювання, методи однокритеріальної та багатокритеріальної оптимізації. Буде розроблено програмну реалізацію та застосовано методи сурогатного моделювання, проведено експерименти та аналіз отриманих результатів.

1. ОГЛЯД ЛІТЕРАТУРИ

1.1 Актуальність проблеми

Оптимізація складних систем є актуальною задачею в багатьох галузях. Проте, обчислювальна складність задач часто перешкоджає прямому застосуванню числових методів оптимізації. У зв'язку з цим, виникає потреба в розвитку альтернативних підходів, що забезпечують швидку та ефективну оптимізацію складних систем.

З одного боку, складні системи відомі своєю великою кількістю вхідних та вихідних параметрів, а також обчислювальною складністю моделей, які описують ці системи. Наприклад, складність моделі може полягати в наявності нелінійних або недиференційованих функцій. При прямому застосуванні числових методів оптимізації, які базуються на точних обчисленнях, виникають проблеми з обчислювальною ефективністю. Обчислення цільової функції може вимагати значних обчислювальних ресурсів та займати велику кількість часу. Це значно уповільнює процес оптимізації та ускладнює пошук оптимальних рішень для складних систем.

У зв'язку з цим, використання сурогатних моделей для оптимізації складних систем стає актуальним напрямом досліджень (Рис 1.1) [1-6]. Сурогатні моделі є математичними моделями, які апроксимують залежність між вхідними та вихідними змінними системи на основі обмеженої кількості вихідних даних. Вони дозволяють швидко оцінити значення цільової функції для різних значень вхідних параметрів без необхідності повторного обчислення складної моделі.

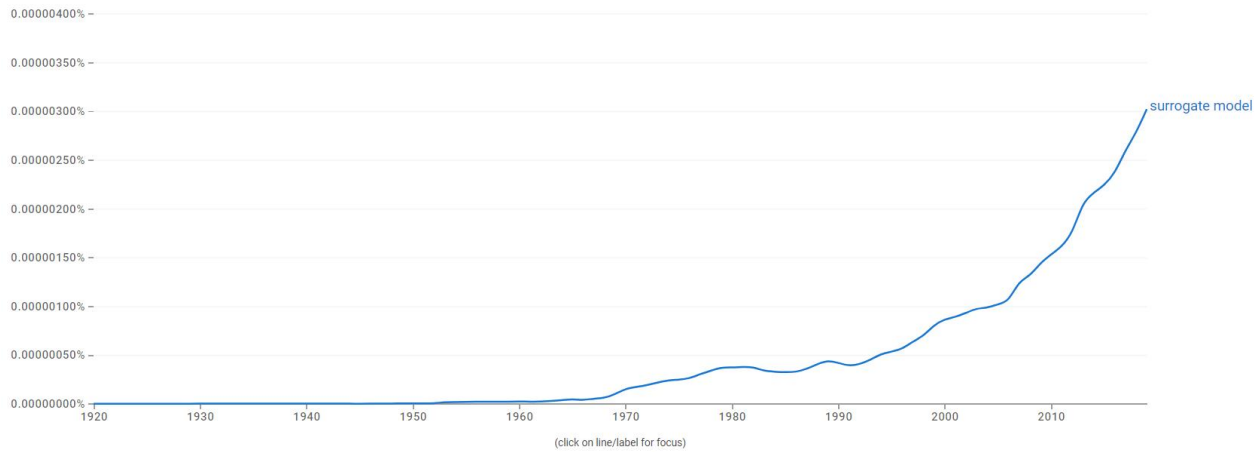

Рис. 1.1 Частота згадувань підходу сурогатних моделей в наукових публікаціях з ресурсу NGram Viewer

Основними підходами до побудови сургогатних моделей є:

- Методи регресії [13-15] - це поширені методи, які використовуються в сурогатному моделюванні. Вони передбачають підбір математичної функції, наприклад, лінійного або поліноміального рівняння, та використання її для наближення наявних точок даних.
- Гаусові процесі [7-9] - це гнучкі та потужні ймовірнісні моделі, які часто використовуються для сурогатного моделювання. Вони забезпечують непараметричний підхід для оцінки базової функції, припускаючи розподіл по функціях, а не конкретну функціональну форму. Моделі гаусових процесів можуть відображати складні взаємозв'язки, включати попередні знання та надавати оцінки невизначеності, що робить їх придатними для різних застосувань, таких як регресія, класифікація та оптимізація.
- Кригінг [10-12] - це метод статистичної інтерполяції, який часто використовується в сурогатному моделюванні. Він особливо корисний, коли має справу з просторово корельованими даними. Моделі кригінгу оцінюють значення неспостережуваних точок на основі спостережуваних точок даних та їхніх просторових взаємозв'язків. Вони дають не тільки прогнози, але й оцінку невизначеності, пов'язаної з цими прогнозами.
- Дерева рішень [19-20] - це техніка сурогатного моделювання, яка використовує деревоподібну структуру для представлення та прогнозування взаємозв'язків між

змінними. Вони рекурсивно розбивають вхідний простір на основі правил прийняття рішень, виведених з навчальних даних. Дерева рішень можуть обробляти як числові, так і категоріальні вхідні дані. Їх можна розширити до ансамблевих методів, таких як випадкові ліси та бустінг, щоб підвищити точність та надійність прогнозування.

- Генетичні алгоритми [16-18] - це популяційний метод пошуку та оптимізації, який зазвичай застосовується в сурогатному моделюванні. Натхненні природною еволюцією, вони використовують комбінацію операцій відбору, кросинговеру та мутації для ітеративного покращення популяції рішень-кандидатів. Генетичні алгоритми можуть ефективно досліджувати простір пошуку та визначати оптимальні або близькі до оптимальних рішення шляхом ітеративної еволюції популяції протягом декількох поколінь.

Кожен з цих методів має свої переваги та обмеження і може бути використаний залежно від конкретних вимог та характеристик задачі оптимізації.

1.2. Покращення та варіації сурогатних моделей

У світі досліджень з оптимізації складних систем, постійно розвиваються нові методи та покращення в сфері сурогатних моделей. Це обумовлено необхідністю досягнення ще більшої точності та ефективності при апроксимації цільової функції, а також покращення швидкості та здатності моделей до адаптації до змінних умов [21].

Одним з напрямків покращення є використання адаптивних методів побудови сурогатних моделей [18]. Ці методи дозволяють моделі самостійно налаштовуватись та покращувати свої апроксимаційні можливості під час процесу оптимізації. Наприклад, можуть бути використані алгоритми, які визначають оптимальну кількість точок для побудови моделі, а також вирішують проблему підбору найкращого типу моделі для конкретної задачі. Це дозволяє досягнути кращої точності та надійності результатів оптимізації.

Ще одним напрямком розвитку є комбінація сурогатних моделей з іншими методами оптимізації. Наприклад, можна використовувати гібридні підходи, які поєднують сурогатні моделі з еволюційними алгоритмами [16] або методами метаоптимізації. Це дозволяє поєднати переваги обох підходів і отримати кращі результати оптимізації.

Крім того, активно досліджуються питання покращення точності сурогатних моделей шляхом використання додаткової інформації [3]. Наприклад, можна використовувати дані з попередніх оптимізаційних ітерацій для поліпшення моделі, або використовувати додаткові властивості системи для побудови більш точних апроксимаційних функцій.

Дослідження в галузі сурогатних моделей також спрямовані на покращення ефективності обчислень. Це досягається шляхом розвитку швидких алгоритмів для побудови та оновлення моделей, а також використання розподіленого обчислення та паралельних обчислювальних ресурсів. Це дозволяє зменшити час, необхідний для побудови та використання сурогатних моделей, і забезпечує більшу швидкість оптимізаційного процесу.

1.3 Приклади практичного застосування

У сучасному інженерному та науковому середовищі сурогатні моделі широко використовуються для рішення різноманітних складних задач [21]. Нижче наведені деякі приклади практичного застосування сурогатних моделей в різних галузях.

- Аерокосмічна інженерія [3, 22] : У галузі аерокосмічної інженерії сурогатні моделі використовуються для оптимізації форми крила літальних апаратів, підвищення аеродинамічної ефективності та зниження опору повітря. Наприклад, дослідження проведені в [23] показали, що використання сурогатних моделей у комбінації з еволюційними алгоритмами дозволяє знаходити оптимальні конфігурації крила з меншою кількістю обчислювальних експериментів. Це значно зменшує час та затрати на проектування.

- Автомобільна промисловість: У дослідженні [24] використання сурогатних моделей для оптимізації структури каркасу автомобіля дозволило досягти значного зменшення маси без втрати міцності. Це призвело до поліпшення паливної ефективності та зниження викидів шкідливих речовин.
- Енергетика: У галузі вітроенергетики сурогатні моделі використовуються для оптимізації конфігурації вітрогенераторів та визначення оптимальної робочої точки системи. Дослідження [25] показало, що використання сурогатних моделей дозволяє покращити ефективність вітрових установок та забезпечити максимальний вихід потужності при різних умовах вітру.
- Хімічна промисловість: У хімічній промисловості сурогатні моделі використовуються для оптимізації процесів синтезу та проектування хімічних реакторів. Дослідження [26] показало, що використання сурогатних моделей у поєднанні з генетичними алгоритмами дозволяє знаходити оптимальні умови процесу синтезу з мінімальними витратами сировини та енергії.
- Екологічне моделювання: Сурогатні моделі застосовуються для моделювання та прогнозування впливу промислових процесів на навколишнє середовище. Дослідження [27] показало, що використання сурогатних моделей дозволяє швидше та ефективніше визначати вплив викидів шкідливих речовин на різні компоненти екосистеми, такі як повітря, вода та грунт.

Ці приклади ілюструють широкі можливості використання сурогатних моделей у різних галузях інженерії та науки. Використання цих моделей дозволяє досягти ефективніших та оптимальних рішень, зменшити час та витрати на проектування та оптимізацію систем.

2. ОДНОКРИТЕРІАЛЬНА ОПТИМІЗАЦІЯ

Проблема управління об’єктами означає знаходження найкращої комбінації вхідних параметрів при яких на виході будуть отримані найкращі значення, тобто є задачею оптимізації. В цьому розділі розглянута проблема однокритеріальної оптимізації, основні поняття, а також деякі методи їх вирішення [27].

Розглянемо задачу оптимізації в найбільш загальному вигляді.

Задля позбавлення необхідності уточнення позначень, які будуть зустрічатись протягом роботи, опишемо деякі з них:

- x – вектор вхідних змінних
- f(x) – вектор цільових функцій або критеріїв
- g(x), h(x) - вектори обмежень які накладаються на вхідні змінні
- N – кількість вхідних змінних, тобто довжина вектора x
- K – кількість цільових функцій або довжина вектора F
- M – кількість обмежень, тобто сумарна довжина векторів g и h
- Q – множина допустимих значень векторів x, які задовольняють всім обмеженням

2.1 Постановка задачі однокритеріальної оптимізації

Для задачі однокритеріальної оптимізації оберемо K=1, таким чином цю задачу можна описати як знаходження мінімального значення функції при певних обмеженнях або:

$$min_{x \in Q} f(x)$$

При (1.1)

$$g(x) \geq 0$$
$$h(x) = 0$$

При M = 0 задача називається безумовною оптимізацією, якщо ж обмеження існують, то це задача умовної оптимізації.

При відсутності умов ми говоримо про мінімізацію цільової функції оскільки задача знаходження максимума зводиться до задачі (1.1) простою заміною знака перед функцією на протилежний.

Результатом вирішення вказаної задачі (1.1) є знаходження точки мінімума яка задовольняє всім обмеженням.

2.2 Основні поняття

Визначимо поняття які необхідні для розуміння та розгляду задачі однокритеріальної оптимізації.

2.2.1 Поняття мінімуму

Як було зазначено вище, задача оптимізації передбачає знаходження мінімума функції. При її знаходженні необхідно розуміти що існує декілька типів мінімуму а саме локальний та глобальний що позначено на рис.2.1.

Локальний мінімум для цільової функції це x^* для якого виконується така умова:

$$\exists \varepsilon > 0 : f(x^*) < f(x)\ \forall x \in B_\varepsilon(x^*) \cap Q$$

Де $B_\varepsilon(z)$ – коло радіусом ε навколо точки z

Глобальним же мінімумом є точка x^* для якої виконується така умова:

$$f(x^*) < f(x)\ \forall x \in Q$$

Також існують строгі та нестрогі мінімуми. Єдиною різницею з вище описаним є зміна строгої нерівності на нестрогу.

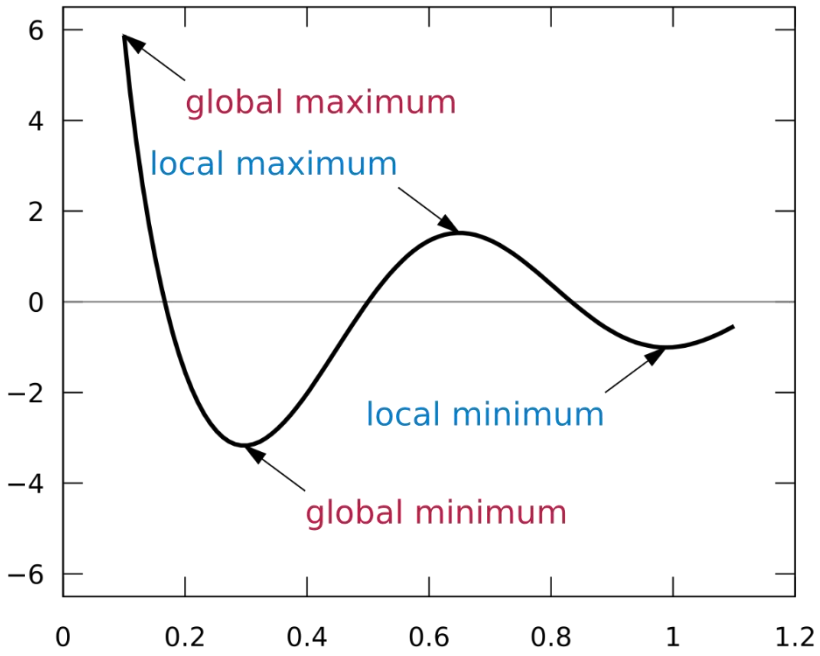

Рис.2.1. Глобальний та локальний мінімум та максимум

Більшість методів створені для пошуку лише локальних мінімумів, однак є методи і для пошуку глобальних. Проте це є більш складна задача, яка потребує більшої кількості обчислень цільової функції, що робить обчислення довшими.

2.2.2 Обмеження

Функції-обмеження представляють собою широко розповсюджений та практично значущий спосіб опису допустимої множини Q. Існують два основних типи обмежень - обмеження рівності h і обмеження нерівності g.

Особливо варто відзначити підмножину обмежень нерівності $x_i + a \geq 0$, які часто називаються коробковими обмеженнями. Легко помітити, що такі обмеження ставлять умови безпосередньо на діапазон зміни окремих вхідних змінних. Часто завдання, які складаються лише з коробкових обмежень, відносять до задач безумовної оптимізації. Для багатьох алгоритмів оптимізації наявність коробкових обмежень є обов'язковою, особливо в підході з використанням сурогатних моделей.

2.2.3 Неперервність вхідних змінних

Надалі ми вважатимемо, що вхідні змінні неперервні. Однак важливе практичне значення мають і завдання, в яких всі або частина змінних є дискретними, наприклад, можуть мати лише цілі значення. Це, відповідно, завдання цілочисельної та змішаної оптимізації.

Умова цілочисельності змінних виникає безпосередньо з їхнього фізичного сенсу в задачі яка вирішується, конкретно, в цільовій функції. Якщо змінна має вигляд, наприклад, довжини, то вона логічно може приймати будь-які позитивні значення. Однак, якщо змінна є кількістю будь-яких предметів, які не можна поділити, або товарів, то можливими є лише натуральні числа або 0.

2.2.4 Детермінованість цільової функції

У деяких задачах оптимізації виникає ситуація, коли цільову функцію та/або обмеження не вдається описати точно, так як вони залежать від деяких стохастичних змінних, значення яких невідомі на момент постановки задачі. Приклади таких задач часто виникають у різних галузях, наприклад, в економіці, коли розв’язок може залежати від майбутніх характеристик ринку або ще не прийнятих рішень.

Подібні задачі можуть вирішуватися у випадку, якщо для подібного роду змінних є змога підібрати певну модель, яка може добре передбачати їх значення. Такою моделлю може стати скінченний набір сценаріїв, з визначеною заздалегідь ймовірністю, або функція розподілу.

В даній роботі ми робимо припущення, що завдання оптимізації не містить стохастичних змінних. Це означає, що при багаторазових обчисленнях в одній і тій же точці, цільова функція, а також всі обмеження, ніколи не змінюють своїх значень.

2.3 Методи оптимізації

Однокритеріальна оптимізація є добре вивченою задачею, тому існує багато методів її розв’язку. Алгоритми оптимізації часто ітеративні, основний їх алгоритм можна описати наступною послідовністю дій:

1. Визначення початкової точки.
2. Згідно з певним критерієм визначається розмір та напрям кроку.
3. Згідно з певним критерієм визначається чи продовжувати оптимізацію. Якщо він не виконується, то повертаємося до пункту 2, інакше завершуємо.

Критерієм зупинки може бути кількість ітерацій, досягнення збіжності, тощо.

Одним з прикладів є метод градієнтного спуску. Початкова точка може визначатись випадково. Для визначення наступної точки необхідно знайти градієнт цільової функції, домножити його на коефіцієнт, який визначає швидкість алгоритму, та зсунути точку в протилежному напрямку.

Вибір коефіцієнту є доволі важливим. Якщо взяти його занадто малим, то алгоритм буде збігатися повільно, якщо ж його взяти занадто великим, то є можливість “перестрибнути” через мінімум або алгоритм може не збігатися.

3. БАГАТОКРИТЕРІАЛЬНА ОПТИМІЗАЦІЯ

При оптимізації складних об’єктів зазвичай необхідно враховувати декілька критеріїв, які можуть конфліктувати одне з одним. Ситуація ускладнюється тим, що не завжди вдається визначити який критерій є найважливішим. Саме в таких випадках виникає задача багатокритеріальної оптимізації [27].

Головною відмінністю від однокритеріальної оптимізації, крім наявності декількох критеріїв, є те, що, як правило, розв’язком задачі багатокритеріальної оптимізації є не одна точка, а певна їх множина, серед яких неможливо виявити фаворита. Подібна множина називається Парето-оптимальна множина. Остаточний вибір розв’язку залишається за експертами.

Основною метою цього розділу є розглянути основні поняття багатокритеріальної оптимізації, а також найбільш відомі шляхи розв’язку такого роду задач.

3.1 Парето-оптимальність

В однокритеріальному випадку будь-які дві точки x_1 та x_2 із допустимої множини Q завжди можуть бути порівняні: $f(x1) \leq f(x2)$ або $f(x1) > f(x2)$. Таким чином, цільова функція визначає відношення повного порядку на допустимій множині. Однак, коли цільових функцій стає декілька, то виникає і третій варіант - точки можуть бути непорівнюваними, і тоді множина Q стає частково впорядкованою. Більш формально, для будь-яких двох векторів у просторі цільових функцій u і v:

$$u = v \Leftrightarrow \forall i \in \{1, 2, \ldots, K\}\, u_i = v_i$$
$$u \leq v \Leftrightarrow \forall i \in \{1, 2, \ldots, K\}\, u_i \leq v_i$$
$$u < v \Leftrightarrow \forall i \in \{1, 2, \ldots, K\}\, u_i < v_i$$

Відношення > та $\geq$ визначаються аналогічно.

На рис. 3.1 зображено приклад. Для точок всередині зафарбованих областей маємо строгу нерівність, а не межі - нестрогу. Точки A і E є непорівнюваними.

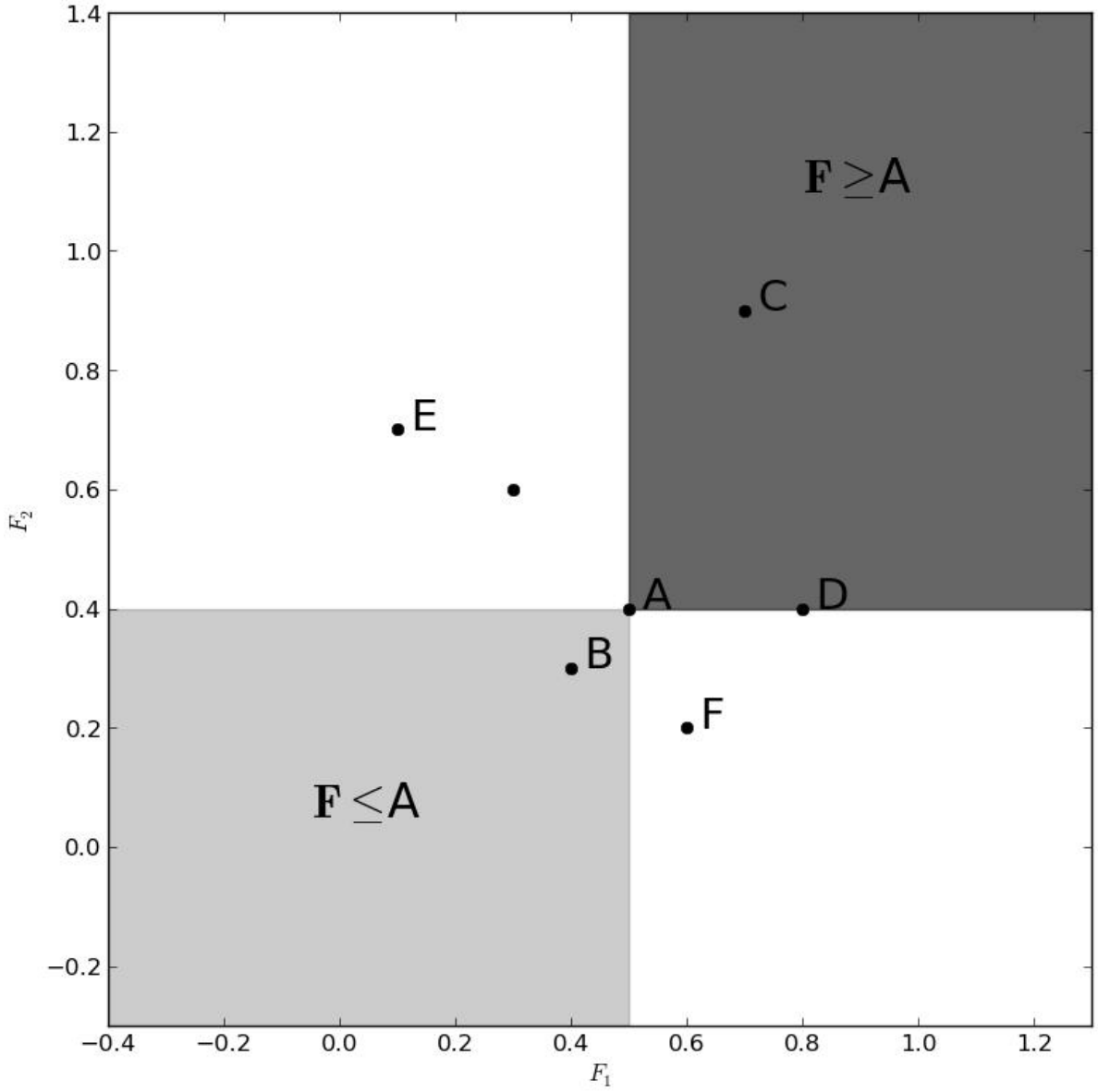

Рис. 3.1. Порівняння векторів [27]

Таким чином, у просторі вхідних змінних для будь-яких двох точок x_1 та x_2 можливі п’ять різних випадків:

$$f(x_1) \leq f(x_2),\ f(x_1) < f(x_2),$$
$$f(x_1) \geq f(x_2),\ f(x_1) > f(x_2),$$
$$f(x_1) \not\leq f(x_2) \cap f(x_1) \not\geq f(x_2).$$

Вводять поняття Парето-домінування: для будь яких точок x_1 та x_2 вхідних змінних

$$x_1 \prec x_2(\textit{домінування}) \Leftrightarrow f(x_1) < f(x_2)$$

$$x_1 \preccurlyeq x_2 (\textit{слабке домінування}) \Leftrightarrow f(x_1) \leq f(x_2)$$

$$x_1 \backsim x_2 (\textit{непорівнювані}) \Leftrightarrow f(x_1) \not\leq f(x_2) \cap f(x_1) \not\geq f(x_2)$$

Спираючись на ці поняття, можна ввести визначення парето-множини [18] – це множина всіх допустимих точок $x \in Q$, для кожної з якої не знайдеться точки яка буде домінувати над нею:

$$\nexists x_0 \in Q: x_0 \preccurlyeq x$$

Таким чином можна сказати, що розв'язком багатокритеріальної задачі оптимізації є або одна парето-оптимальна точка, або ж кінцева апроксимація парето-множини. Першу називають локальною багатокритеріальною задачею, останню ж – глобальною.

Щоб завершити обговорення структури вирішення багатокритеріальних завдань, залишається ще одне важливе питання – яке наближення Парето множини можна вважати якісним. Зазвичай розглядаються наступні основні вимоги. По-перше, потрібно як можна ближче наблизитися до істинного Парето-фронту. По-друге, треба покрити як можна більшу частку фронту, тобто отримати максимально можливий діапазон значень кожної цільовий функції в отриманому розв'язку. По-третє, цей діапазон важливо заповнити розв'язками в деякому сенсі рівномірно.

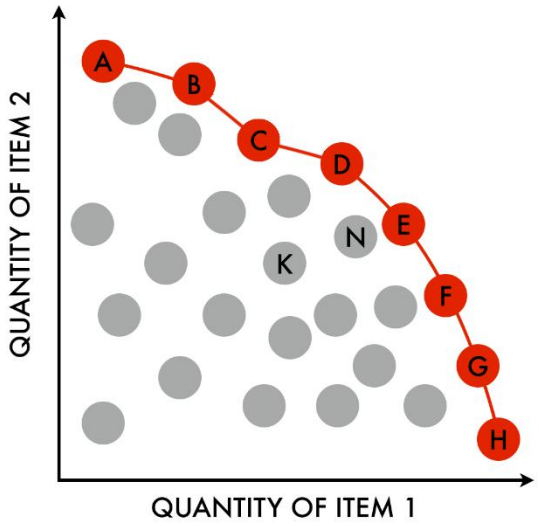

Рис. 3.2. Парето фронт [28]

На рисунку 3.2 наведено практичний приклад вирішення, отриманий за допомогою деякого чисельного алгоритму.

Незважаючи на те, що формально розв'язком задачі є наближення Парето-оптимальної множини, як правило, увагу приділяють вигляду отриманого наближення Парето фронту, оскільки він більш наочний.

Також, на цьому рисунку зазначено ще кілька важливих, до цього не згаданих точок. Насамперед, потрібно звернути увагу на так звані якірні точки A та H – фактично, вони відповідають розв'язкам однокритеріальних завдань мінімізації кожною компоненти вектора $f(x)$ окремо. Пошук таких точок є ключовим кроком для багатьох чисельних алгоритмів.

3.2 Методи розв'язку

3.2.1 Скаляризація

Під скаляризацією в контексті багатокритеріальної оптимізації [28] розуміється сукупність алгоритмів, що дозволяє для будь-якої багатокритеріальної задачі створити однокритеріальну задачу, таким чином щоб її розв'язок входив у множину парето-оптимальних розв'язків для початкової задачі.

На виході правильно скаляризованої задачі постає парето-оптимальна точка. В деяких випадках цього буває достатньо, проте зазвичай необхідно отримати парето-фронт, а для цього необхідно розв'язати низку таких задач.

Оскільки скаляризація – це ціла група підходів, варто зазначити на які класи за принципом дії вона поділяється.

Перший підхід базується на виборі скаляризаційної функції (функції корисності). Вибір такої функції може бути зроблений алгоритмічно або ж вручну.

Розглянемо деякі види скаляризаціних функцій [28]:

- Зважена сума

$$f(x) = \sum_{i=1}^{k} w_i f_i(x)$$

- Експоненціальне зважження

$$f(x) = \sum_{i=1}^{k} exp\,(pw_i - 1) exp\,(pf_i(x))$$

- Зважена степінь

$$f(x) = \sum_{i=1}^{k} w_i f_i(x)^p$$

- Зважена норма

$$f(x) = (\sum_{i=1}^{k} w_i |f_i(x)|^p)^{1/p}$$

та багато інших, де w_i – ваги до кожної з функцій.

Зазвичай суму ваг беруть рівною одиниці:

$$\sum_{i=1}^{k} w_i = 1$$

За допомогою ваг ми можемо позначити, що один з критеріїв є більш пріоритетним ніж інші. Для більш детального розгляду та простоти оберемо зважену суму з двома цільовими функціями. Тоді скаляризована функція буде мати наступний вигляд:

$$f(x) = w_1 f_1(x) + w_2 f_2(x)$$

Розглянемо, як за допомогою лінійної скаляризації отримати розв'язок багатокритеріальної задачі. Найпростішим є взяти n значень першої ваги рівномірно з відрізку [0, 1]:

$$w_{1i} = \frac{i}{n-1}, i \in \{0,1, \dots, n-1\}$$

w_2 визначається з обмеження $w_1 + w_2 = 1$. Розв'язуючи отриману однокритеріальну задачу для усіх вибраних значень w_{1i}, отримаємо наближення Парето-множини.

Другим способом скаляризації після вибору функції корисності є метод обмежень.

Припустимо, що розв'язавши незалежно задачі мінімізації кожної цільової функції $f_i(x)$, ми знайшли всі якірні точки. Після цього можна зробити наступним чином – вибрати одну з цільових функцій (наприклад f_1) і поставити таку задачу:

$$min_{x \in Q} f_1(x)$$
$$g(x) \geq 0$$
$$h(x) = 0$$
$$-f_i(x) - \varepsilon_i \geq 0, i \in \{2, .., K\}$$

Таким чином, ми будемо вирішувати однокритеріальну завдання мінімізації $f_1(x)$ з додатковими обмеженнями. Варіюючи значення параметрів ε_i в межах діапазону, що задається якірними точками, можна отримувати різні Парето-оптимальні розв'язки. Так само, як і у випадку скаляризаційної функції для рівномірного покриття Парето фронту потрібно буде застосувати адаптивний підбір цих параметрів.

Опишемо основні недоліки та переваги скаляризації.

Одразу очевидною є головна перевага вибору цього методу, а саме спрощення багатокритеріальної задачі до однокритеріальної, що дає можливість використовувати один з багатьох добре досліджених однокритеріальних методів.

Недоліками методу скаляризації є, по-перше, те, що рівномірність покриття Парето-множини дуже залежить від геометрії задачі. Іноді навіть може бути неможливим досягнення деяких її ділянок. По-друге, незалежний розв'язок багатьох однокритеріальних задач призводить до втрати

інформації про цільові функції, що накопичується на кожному кроці. Це призводить до того, що для розв’язку глобальної багатокритеріальної задачі доводиться більше разів розраховувати цільові функції.

3.2.2 Генетичні алгоритми

Генетичні алгоритми – широкий клас методів оптимізації, що симулюють процес природної еволюції [16-18].

Усі ці методи оперують набором розв’язків-кандидатів. Спрощено, цей набір ітеративно модифікується за допомогою двох основних принципів еволюції: відбору та варіації.

Для живих організмів відбір означає конкуренцію за ресурси. Більш пристосовані організми мають більшу ймовірність вижити і залишити потомство. В еволюційних алгоритмах природний відбір симулюється стохастичним процесом вибору найкращих розв’язків. Кожний розв’язок отримує шанс на відтворення залежно від своєї якості. Як правило, якість оцінюється скалярною функцією пристосованості, вибір якої є одним із ключових елементів будь-якого генетичного алгоритму. Другий принцип, варіація, реалізується імітацією таких біологічних процесів як мутації.

Функція пристосованості може бути як побудована за принципами, вже розглянутими в огляді методів скаляризації, так і за більш наближеними до багатокритеріальних задач критеріям домінування. Обидва підходи широко поширені, однак варто зазначити, що використання в якості функції пристосованості традиційних скаляризаційних функцій призводить до того, що побудований генетичний алгоритм фактично успадковує всі недоліки, властиві скаляризації.

Незважаючи на ідейну простоту, генетичні алгоритми показують хороші результати на багатьох задачах. Їхня пристосованість до вирішення багатокритеріальних задач багато в чому пов'язана з можливістю отримати

за один запуск багато розв’язків, тобто, власне, шукане наближення Парето множини.

Серед переваг генетичних алгоритмів можна назвати:

- малу чутливість до шуму в цільових функціях
- можливість глобального пошуку
- можливість паралельного обчислення багатьох точок під час розв’язку

Недоліком же є ускладнення при вирішенні завдань, в яких є досить жорсткі обмеження, особливо обмеження типу рівності.

3.2.3. Багатокритеріальні градієнтні методи

Градієнтні методи [28] – можливо, найбільш поширені алгоритми оптимізації, що спираються на побудову локальної моделі функції на основі її похідних. Фактично кожному, хто стикався з оптимізацією при неперервних змінних, знайомі такі алгоритми як метод найшвидшого спуску або метод Ньютона.

Однокритеріальні градієнтні методи стали одними з перших сучасних алгоритмів оптимізації, однак їх багатокритеріальні узагальнення отримані порівняно недавно. Розглянемо основні ідеї, що лежать за цими методами.

Виділимо дві ключові компоненти багатокритеріальних градієнтних методів:

- спуск до фронту - розв’язок локальної багатокритеріальної задачі, пошук хоча б однієї Парето-оптимальної точки.
- розсіювання - спосіб зміститись в просторі вхідних змінних так, щоб залишитися на Парето-фронті у просторі цільових функцій.

Повторюючи ці кроки багаторазово, можна відновити Парето множину з високою точністю.

Можна назвати багато переваг такого підходу:

- наявність чисельних оцінок похідних дозволяє алгоритму контролювати виконання умов оптимальності, що у багатьох випадках дозволяє отримувати найточніші розв’язки.
- ефективна робота з обмеженнями, зокрема, з найбільш складними для інших алгоритмів обмеженнями типу рівності.
- пристосованість градієнтних методів щодо відносно високих розмірностей – як простору вхідних змінних N, так і простору цільових функцій K.

Проте, є і недоліки, повністю успадковані від однокритеріальних градієнтних методів:

- висока чутливість до шуму в функціях і обмеженнях, яка тільки накопичується в більш досконалих методах з використання високого порядку похідних
- проблема організації глобального пошуку - спроба узагальнити градієнтні методи для глобальної оптимізації наштовхується на безліч перешкод. Це ускладнює їх використання при вирішенні задач з безліччю локальних мінімумів

4. ОПТИМІЗАЦІЯ З ВИКОРИСТАННЯМ СУРОГАТНИХ МОДЕЛЕЙ

На сьогоднішній день, перед нами постають все складніші оптимізаційні задачі. Незважаючи на досить швидкий ріст обчислювальних можливостей, який відкрив можливості до оптимізації нових галузей, обчислення точних та складних моделей може займати досить багато часу: від декількох хвилин до днів та тижнів. До складності обчислень моделі додається потреба знаходження саме глобального мінімуму.

Одним із способів покращити ситуацію є побудова сурогатних моделей. Основною задачею сурогатної моделі є максимально точна передача поведінки функції, що моделюється, залишаючись при цьому обчислювально простою. Саме тому сурогатні моделі здобули популярності в прикладних сферах таких як інженерне проектування.

На сьогодні розроблена велика кількість методів побудови такого роду моделей. В цьому розділі будуть розглянуті найвідоміші з них.

4.1 Загальний алгоритм

Загальна схема використання сурогатних моделей зображена на рис 4.1. Для зручності позначимо функцію, для якої будується сурогатна модель, як $y(x)$, а її сурогатну модель як $\tilde{y}(x)$.

Як позначено на схемі, першим пунктом йде вивчення об’єкта моделювання. Це може допомогти в подальшому, наприклад, можна сформулювати додаткові обмеження. Також на цьому етапі визначається тип сурогатної моделі яка буде використовуватись, що залежить від багатьох факторів. Назвемо деякі з них:

- Знання об’єкта моделювання
- Особливості моделі

- Обмеження певних типів сурогатних моделей

Наступний крок – побудова вибірки, на якій модель буде навчатись.

Третім йде створення датасету вигляду $S = \{(x_i, y_i), i \in [1, n]\}$. Тобто необхідно порахувати значення в точках які були обрані на минулому кроці: $y_i = y(x_i)$. Оскільки розрахунок функції в різних точках є незалежним, то його можна виконувати паралельно на різних кластерах.

Рис. 4.1. Загальна схема дослідження функції за допомогою сурогатних моделей

Після отримання значень починається етап побудови сурогатної моделі, яку було обрано на першому етапі.

Після цього ми маємо вже певну апроксимовану функцію $\tilde{y}(x)$.

Якщо точність моделі є задовільною, то зупиняємось, інакше розширюємо вибірку та повертаємось до третього пункту.

4.2 Дослідження об’єкту оптимізації

Перед побудовою сурогатної моделі, важливо дослідити об’єкт оптимізації.

Одним з важливих аспектів попереднього дослідження є правильне розбиття простору параметрів. Розбиття простору включає визначення діапазонів значень кожного параметра, а також створення попередньої вибірки. Правильне розбиття простору дозволяє досягти балансу між точністю моделі та обчислювальними витратами.

Попереднє дослідження об'єкту оптимізації також допомагає зрозуміти поведінку функцій та взаємозв'язки між ними. Це може включати аналіз залежностей, ідентифікацію границь та врахування особливостей функцій. Ці знання є цінними при виборі підходів до побудови сурогатних моделей, виборі методів оптимізації та формулюванні обмежень.

Одним з аспектів на які варто звернути увагу є однорідність та зв’язність області вхідних параметрів моделі. Може виявитися, що ця область є об’єднанням декількох незв’язних множин. Тоді глобальна сурогатна модель, побудована на вибірці, що є об’єднанням декількох кластерів точок, в загальному випадку буде мати низьку точність. Тому буває виправданим попередньо додатково розбити таку вибірку на декілька однорідних підвибірок, кожна з яких лежить у зв’язній області простору вхідних параметрів, після чого будувати локальні апроксимації по відповідним підвибіркам.

Іншим суттєвим моментом є те, що набір вхідних даних задачі може бути надлишковим в одному з двох сенсів:

- Вхідні дані можуть бути пов’язані між собою. В найпростішому випадку це означає те, що декілька вхідних змінних значно скорельовані.

- Функція може залежати не від усіх вхідних змінних. Основними є такі два сценарії: функція слабко залежить від декількох змінних, або функція залежить від проекції вхідних змінних на деякий лінійний підпростір.

Крім того, попереднє дослідження дозволяє визначити, які додаткові дані можуть бути необхідними для побудови сурогатної моделі. Це можуть бути результати попередніх обчислень, експериментів або симуляцій, які можуть використовуватися для тренування моделей або валідації їх точності.

4.3 Створення початкової вибірки

Побудова вибірки має вплив на ефективність та якість оптимізаційного процесу, а також на отримані результати. Ось деякі ключові аспекти, які варто враховувати при побудові вибірки:

1. Покриття простору параметрів: Важливо, щоб вибірка покривала всі значущі області простору параметрів. Якщо певні діапазони значень параметрів виключені, можливо, буде пропущена можливість знайти оптимальні розв'язки в цих областях. Вибірка повинна бути репрезентативною для всього простору параметрів, щоб уникнути пропускання потенційно оптимальних розв'язків.

2. Рівномірний розподіл точок: Бажано, щоб точки вибірки були розподілені рівномірно по всьому простору параметрів. Це допомагає забезпечити, щоб жодна область простору параметрів не буде надмірно згущеною або розрідженою. Рівномірний розподіл дозволяє оптимізаційному алгоритму ефективно досліджувати різні області та знаходити оптимальні розв'язки.

3. Репрезентативність: Вибірка повинна бути репрезентативною для функцій або систем, що оптимізуються. Це означає, що вона повинна включати точки, які представляють різні варіанти або стани системи.

Важливо враховувати особливості функцій та їх залежності для вибору відповідного розподілу точок.

4. Розмір вибірки: Вибірка повинна бути достатньо великою, щоб достатньо охопити простір параметрів та забезпечити надійну оцінку функцій. Занадто маленька вибірка може призвести до недостатнього покриття простору параметрів і ненадійних результатів.

Правильна побудова вибірки допомагає забезпечити широке дослідження простору параметрів, рівномірний розподіл точок та надійні результати оптимізації. Це покращує шанси знайти оптимальні розв'язки та підвищує ефективність оптимізаційного процесу.

При побудові початкової вибірки для задач оптимізації існує кілька підходів [29], серед яких рівномірна сітка, стохастична вибірка та латинський гіперкуб. Усі три методи використовуються для ефективного покриття простору параметрів та забезпечення рівномірного розподілу початкових точок.

1. Рівномірна сітка (grid sampling). Цей метод передбачає розміщення точок в просторі параметрів за допомогою рівномірної сітки. Він базується на розбитті простору параметрів на рівновіддалені інтервали або сітку, і вибір точки з кожного інтервалу. Наприклад, якщо є одновимірний простір параметрів з діапазоном від a до b, можна вибрати точки з рівномірним кроком $\frac{b-a}{n}$, де n - кількість точок.

2. Стохастична вибірка (random sampling). Цей метод полягає у виборі точок за допомогою деякого випадкового розподілу. Зазвичай використовується рівномірний розподіл. Цей метод є простим у реалізації та не має внутрішньої структури вибірки.

3. Латинський гіперкуб (latin hypercube sampling). Цей метод забезпечує більш рівномірний розподіл точок у просторі параметрів. Він ґрунтується на розділенні кожного параметра на n рівних інтервалів та виборі по одній

точці з кожного інтервалу. При цьому забезпечується умова, що жодна точка не знаходиться у одному рядку або стовпці з будь-якою іншою точкою. Це дозволяє краще покрити простір параметрів та забезпечити рівномірний розподіл точок.

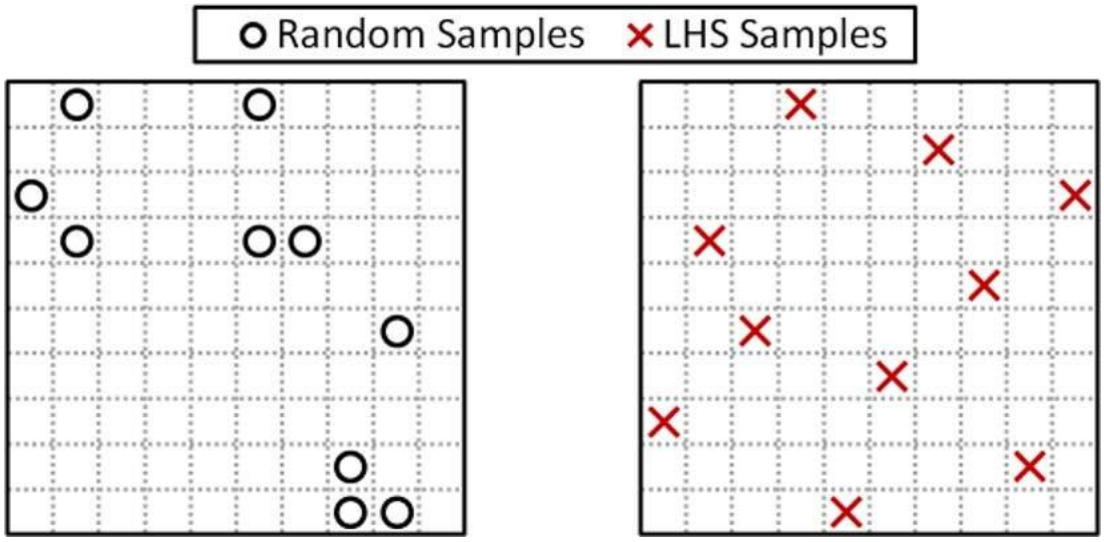

Рис. 4.2. Порівняння вибірок, згенерованих за допомогою випадкового рівномірного розподілу (зліва) та латинського гіперкубу (справа) [4]

4.2 Використання репресій

4.2.1 Лінійна регресія

Розглянемо лінійну регресію [13]. Як випливає з назви, при апроксимації використовується лінійна функція яка в загальному випадку має вигляд:

$$f(x,b) = \sum_{i=0}^{N} x_i * b_i \qquad (4.1)$$

Нагадаємо що N – довжина вектору змінних, x_i – елементи вектору змінних, b_i – коефіцієнти, або іншими словами – швидкість зміни змінної при цьому факторі при фіксованих всіх інших факторах. Також вираз (4.1) можна записати у векторному вигляді

$$f(x,b) = x^T b$$

Таким чином ця сурогатна модель повністю задається вектором коефіцієнтів b.

Для пошуку значень b, які б максимально відповідали вхідній моделі існує багато методів, найпопулярнішим з яких є метод найменших квадратів. Він полягає в мінімізації різниці квадратів між початковою функцією та сурогатною моделлю в заданих точках:

$$min \sum_{i} (y_i - f_i(x))^2$$

Для вирішення цієї задачі, можна використовувати будь-який метод однокритеріальної оптимізації.

Перевагами цього методу є:

- Простота
- Мала обчислювальна складність на кожній ітерації

Разом з тим очевидні і недоліки використання цього методу для створення сурогатної моделі:

- Модель занадто проста, що означає ігнорування нюансів функції взятої для регресії. Тому ця модель буде давати малу точність, за винятком випадків, коли вхідна функція близька до лінійної

Приклад роботи лінійної регресії можна побачити на Рис.4.3.

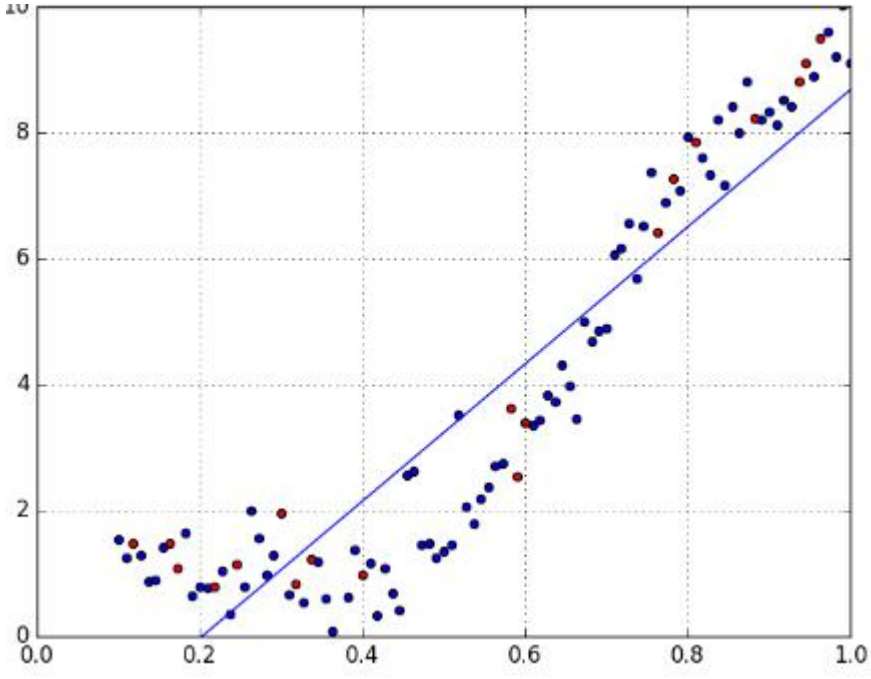

Рис 4.3. Приклад лінійної регресії [13]

4.2.2 Поліноміальна регресія

Більш складною модифікацією лінійної регресії є поліноміальна регресія [14].

Основна відмінність від лінійної регресії полягає в тому, що лінійна функція моделі замінюється поліномом певного порядку. Таким чином формула (4.2) набуває вигляду:

$$f(x, b) = \sum_{i}^{N} \sum_{j}^{L} x_i^j \cdot b_{ij} \qquad (4.2)$$

Де L – степінь поліному, N – довжина вектору вхідних змінних.

Важливим параметром є степінь регресії L. Якщо взяти його занадто малим, то модель може мати малу точність. Якщо ж взяти його занадто великим, то модель може перенавчитися (overfit) і погано відтворювати поведінку початкової моделі для нових точок.

Коефіцієнти b_{ij} знаходяться тими ж методами, що й для лінійної регресії, наприклад методом найменших квадратів.

Приклад поліноміальної регресії можна побачити на Рис 4.4.

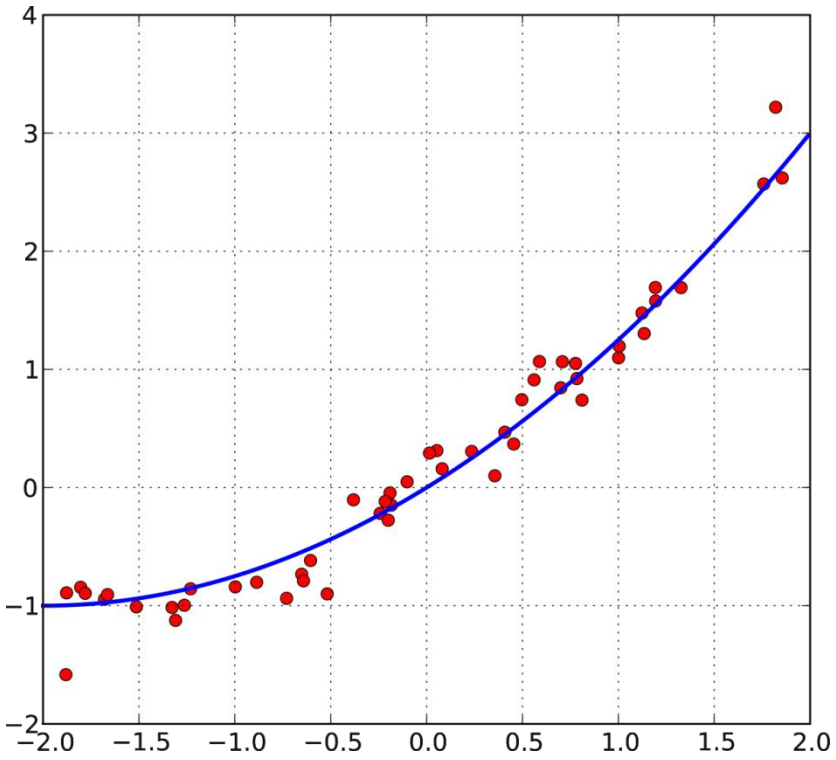

Рис 4.4. Приклад поліноміальної регресії [14]

4.3 Апроксимація за допомогою радіально-базисних функцій

Метод апроксимації за допомогою радіально-базисних функцій [30] полягає у представленні сурогатної моделі у вигляді лінійної комбінації деяких базисних функцій. Базисні функції залежать лише від відстані до центра, і відцентровані на точках вибірки:

$$f(x) = \sum_{i=1}^{n} \phi(x, x_i) w_i$$

де сума йде по точкам вибірки x_i, $\phi(x, x_i) = \phi(|x - x_i|)$ - базисні функції, w_i- ваги. Ваги підбирають так, щоб модель проходила через точки початкової вибірки. Тоді їх можна знайти із системи лінійних рівнянь:

$$\sum_{i=1}^{n} \phi(x_j, x_i) w_i = y_j, j = 1, \ldots, n$$

В якості базисних функцій зазвичай використовують функцію Гауса [30]:

$$\phi(x, x_i) = exp\left(-\frac{|x - x_i|^2}{d^2}\right)$$

4.4 Кригінг

Наступним розглянемо метод кригінгу [12].

Головною особливістю цього методу є використання статистичної моделі даних, що дає багато додаткових можливостей. Наприклад, кригінг дає можливість не лише передбачати значення, а й оцінювати похибку передбачення.

Нехай ми маємо задачу отримати значення функції $\tilde{y}$ в точці x. Основне припущення кригінга – в кожній точці значення функції розглядається як реалізація випадкової величини $Y(x)$, яка має нормальний розподіл з середнім μ та дисперсією σ^2. Якщо функція $y(x)$ неперервна, то для будь-яких двох точок x_1 та x_2 це означає що якщо відстань між ними мала, то й зміна функції теж мала. З точки зору статистики це означає, що якщо відстань між точками мала то між випадковими величинами $Y(x_1)$ та $Y(x_2)$ є сильна кореляція. Кореляцію часто моделюють такою функцією [12]:

$$Corr\left[Y(x_i), Y(x_j)\right] = K(x_i, x_j) = exp\left(-\sum_{k=1}^{N} \theta_k \left|x_{ik} - x_{jk}\right|^2\right)$$

де x_{ik} - k-та компонента вектора x_i. Ця функція дорівнює одиниці, при $x_i = x_j$, та прямує до нуля, коли ці два вектора віддаляються один від одного. Параметр θ_k визначає наскільки швидко кореляція спадає вздовж k-ої компоненти вхідної змінної. При збільшенні θ_k функція буде змінюватись швидше.

У випадку n точок ми представляємо наші знання про поведінку функції у вигляді випадкового вектора $Y = |Y(x_1), \ldots, Y(x_n)|$. Середнє значення даного вектора це $\mu 1$, а матриця коваріації $Cov(Y) = \sigma^2 R$.Тут 1 - вектор одиниць, R - матриця розміром n x n, в якій елемент (i, j) дорівнює $K(x_i, x_j)$. Таким чином, розподіл вектора Y повністю задається параметрами μ, σ^2 та θ_k, k = 1, …, N.

Для підбору цих параметрів використовується метод максимальної правдоподібності. Нехай вектор $y = (y_1, ..., y_n)$ - вектор значень функції y у точках вибірки. Тоді функція правдоподібності визначається як [12]:

$$\frac{1}{(2\pi)^{\frac{n}{2}}(\sigma^2)^{\frac{n}{2}}|R|^{\frac{1}{2}}} exp\left(\frac{-(y-\mu 1)^T R^{-1}(y-\mu 1)}{2\sigma^2}\right)$$

Смисл максимізації правдоподібності - підібрати такі значення параметрів, при яких є максимальною ймовірність спостерігати задану вибірку як реалізацію випадкового вектора Y. На практиці зручніше розглядати максимізацію логарифма функції правдоподібності, який має вигляд (без константних факторів):

$$-\frac{n}{2}log\,(\sigma^2) - \frac{1}{2}log\,(|R|) - \frac{(y-\mu 1)^T R^{-1}(y-\mu 1)}{2\sigma^2}$$

Прирівнюючи похідні цієї функції по μ та σ^2 до нуля та розв'язуючи отримані рівняння, отримуємо оптимальні значення відповідних параметрів:

$$\hat{\mu} = \frac{1^T R^{-1} y}{1^T R^{-1} 1}$$

$$\hat{\sigma}^2 = \frac{(y-\hat{\mu}1)^T R^{-1}(y-\hat{\mu}1)}{n}$$

Підставляючи ці значення у вираз для логарифму функції правдоподібності, отримаємо:

$$-\frac{n}{2}log\,(\hat{\sigma}^2) - \frac{1}{2}log\,(|R|)$$

Такий запис функції правдоподібності залежить лише від матриці R, тобто від параметрів функції кореляції θ_k . Ця функція максимізується чисельно для отримання оцінок θ_k.

Щоб отримати передбачення моделі в точці x^*, додаємо її до вибірки та підбираємо значення y^*, щоб отримати максимальну правдоподібність. Вектор кореляцій випадкової величини $Y(x^*)$ з точками вибірки:

$$r = \left|Corr\,[Y(x^*), Y(x_1)], \ldots, Corr\,[Y(x^*), Y(x_n)]\right|$$

Тоді доповнена матриця кореляції:

$$\widetilde{R} = \begin{pmatrix} R & r \\ r^T & 1 \end{pmatrix}$$

Для зміни логарифма функції правдоподібності отримуємо:

$$\frac{-1}{2\hat{\sigma}^2\left(1-r^T R^{-1} r\right)}(y^* - \hat{\mu})^2 + \frac{r^T R^{-1}(y-\hat{\mu}1)}{\hat{\sigma}^2\left(1-r^T R^{-1} r\right)}(y^* - \hat{\mu}) + \{\textit{доданки, що не містять } y^*\}$$

Диференціюючи цей вираз по y^* та прирівнюючи до нуля, знаходимо оптимальне значення:

$$\hat{y}(x^*) = \hat{\mu} + r^T R^{-1}(y - \hat{\mu}1)$$

Це і є передбачення за методом кригінга. Крім того, метод кригінга дозволяє отримати вираз для дисперсії цього передбачення [12]:

$$s^2(x^*) = \hat{\sigma}^2\left[1 - r^T R^{-1} r + \frac{(1 - r^T R^{-1} r)^2}{1^T R^{-1} 1}\right]$$

На рис. 4.5 показано приклад використання методу кригінга.

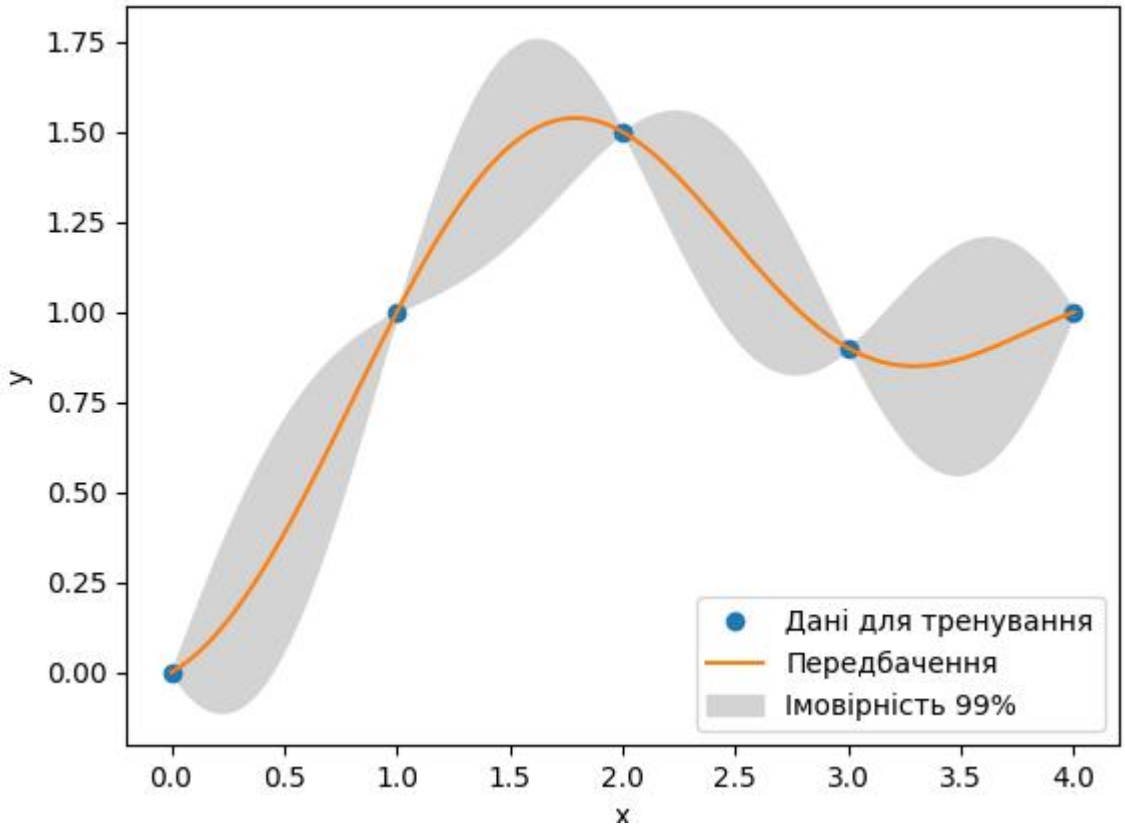

Рис 4.5. Приклад використання крінгінгу [33]

Крім тільки що розглянутого базового методу кригінгу, існують також його модифікації, такі як KPLS та KPLSK.

4.4.1. KPLS

Kriging with Partial Least Squares (KPLS) [12] представляє собою ефективний метод в області сурогатного моделювання, що поєднує переваги двох методів: Kriging та Partial Least Squares (PLS). Цей підхід є особливо корисним при роботі з високовимірними просторами параметрів, де традиційний Kriging може зіткнутися з проблемами обчислювальної ефективності.

В KPLS, сурогатна модель розробляється в два основні кроки. Спочатку, метод PLS використовується для виконання лінійного витягу основних компонентів з вихідних даних. Це дозволяє зменшити вимірність простору параметрів, зберігаючи при цьому максимальну кількість важливої інформації. Результат цього етапу - набір нових змінних, що відображають більшість варіацій вихідних даних.

На другому кроці, традиційний Kriging використовується для пошуку оптимальних значень цих змінних. Цей процес включає в себе використання автокореляції між змінними для моделювання їх поведінки. Оскільки ці змінні є результатом трансформації PLS, вони мають меншу вимірність, ніж оригінальні дані, що забезпечує більшу обчислювальну ефективність.

Таким чином, KPLS забезпечує ефективність PLS у відношенні до зменшення вимірності, одночасно використовуючи точність прогнозування Kriging для отримання оптимальної сурогатної моделі.

4.4.2. KPLSK

Кригінг з частковими найменшими квадратами та ядром (KPLSK) [12] - це прогресивний метод в області сурогатного моделювання. Він спирається на сильні сторони методів кригінгу, часткових найменших квадратів (PLS) та ядра, щоб забезпечити надійне та ефективне рішення для моделювання.

Цей метод відрізняється від методу KPLS тим, що між PLS та кригінгом додається ще один крок - ядрове перетворення, що дозволяє отримати нелінійне представлення даних. Ядрові методи можуть ефективно охоплювати складні взаємозв'язки та взаємодії, які можуть бути неадекватно представлені в лінійному просторі PLS, підвищуючи прогностичну спроможність моделі.

Таким чином, KPLSK поєднує в собі ефективність зменшення розмірності PLS, потужність нелінійного представлення даних ядерних методів і прогностичну точність кригінгу для створення оптимальної сурогатної моделі.

4.5. Оцінка точності сурогатної моделі

Для оцінки точності сурогатної моделі можна використати наступні оцінки, кожна з яких надає різний погляд на роботу моделі [13]:

- Середньоквадратична помилка (MSE): MSE - це загальновживана функція втрат регресії, яка обчислює середнє значення квадратів різниць між прогнозованими та фактичними значеннями. Це чудовий вибір для багатьох

задач, але він має недолік - великий вклад викидів через піднесення до квадрату кожної помилки.

- Середня абсолютна похибка (MAE): MAE, з іншого боку, обчислює середнє значення абсолютної різниці між прогнозованими та фактичними значеннями. Оскільки вона не підносить помилки до квадрату, викиди мають меншу вагу порівняно з MSE. Це робить MAE показником більш стійким до викидів.

- Середня абсолютна відсоткова похибка (MAPE): MAPE обчислює середнє значення абсолютної відсоткової різниці між прогнозованим і фактичним значеннями. Це особливо корисно, коли необхідно зрозуміти похибку в термінах відносного розміру прогнозів, що робить його популярним вибором для проблем прогнозування. Однак MAPE може призвести до проблем, коли фактичні значення мають нульові або близькі до нуля значення, оскільки це може призвести до невизначених або надзвичайно високих відсоткових помилок.

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(F(x_i) - \hat{f}(x_i))^2$$

$$MAE = \frac{1}{n}\sum_{i=0}^{n}|F(x_i) - \hat{f}(x_i)|$$

$$MAPE = \frac{100\%}{n}\sum_{i=0}^{n}\left|\frac{F(x_i) - \hat{f}(x_i)}{F(x_i)}\right|$$

5.РЕАЛІЗАЦІЯ АЛГОРИТМУ ПОБУДОВИ СУРОГАТНИХ МОДЕЛЕЙ

5.1 Вибір технологій

5.1.1 Python

Для реалізації сурогатних моделей використано мову Python.

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

Вона має наступні переваги:

1. Широкий спектр бібліотек: Python має велику кількість бібліотек, які спрощують побудову та роботу з сурогатними моделями. Наприклад, бібліотеки, такі як scikit-learn, TensorFlow, PyTorch та Keras, надають потужні інструменти для побудови різних типів моделей, включаючи нейромережеві моделі, регресійні моделі та моделі на основі дерев рішень.

2. Велика спільнота розробників: Python має активну та велику спільноту розробників, що сприяє доступності документації, підтримці та обміну знаннями. Це дозволяє швидко знаходити рішення на форумах та отримувати оновлення та покращення для бібліотек.

3. Легкість використання та зрозумілість: Python є дуже зрозумілою та легкою в освоєнні мовою програмування. Вона має простий та читабельний синтаксис, що дозволяє швидко розробляти та налагоджувати код. Це особливо корисно для дослідників та інженерів, які швидко хочуть перевірити концепцію та виконати експерименти.

4. Інтеграція з іншими інструментами: Python добре інтегрується з іншими інструментами та платформами. Наприклад, його можна поєднувати з Matlab, базами даних, системами візуалізації та іншими інструментами, що дозволяє зручно обробляти та аналізувати дані, які використовуються для побудови сурогатних моделей.

5.1.2 NumPy

NumPy (Numerical Python) [32] є потужною бібліотекою для обчислювальної математики у мові програмування Python. Вона надає широкі можливості для маніпулювання багатовимірними масивами даних, включаючи матриці та вектори, а також функції для ефективного виконання різних обчислювальних операцій.

Однією з головних переваг NumPy є його високооптимізована реалізація, яка дозволяє здійснювати швидкі обчислення над масивами даних. В основі NumPy лежить об'єкт ndarray (N-dimensional array), який представляє собою багатовимірний масив одного типу даних. Цей об'єкт забезпечує ефективне зберігання та обробку даних, що дозволяє виконувати широкий спектр обчислювальних завдань.

Завдяки NumPy можна виконувати базові математичні операції над масивами, такі як додавання, віднімання, множення та ділення, які виконуються поелементно. Також NumPy надає функціональні можливості для розширених математичних операцій, таких як лінійна

алгебра, розклад матриць, операції над комплексними числами, статистичні обчислення та інше.

Окрім того, NumPy має розширені можливості для індексації та зрізів масивів, що дозволяє ефективно виконувати вибірку та модифікацію підмасивів даних. Це робить NumPy потужним інструментом для обробки та аналізу великих обсягів даних.

Бібліотека NumPy є основою для багатьох інших популярних бібліотек для наукових обчислень у Python, таких як SciPy, pandas, matplotlib та smt, які були використані в даній роботі.

5.1.3 Surrogate modeling toolbox

Surrogate modeling toolbox (SMT) [33] - це пакет Python з відкритим вихідним кодом, що складається з бібліотек методів сурогатного моделювання (наприклад, радіальних базисних функцій, кригінгу), методів вибірки та задач бенчмаркінгу. SMT розроблений для того, щоб полегшити розробникам реалізацію нових сурогатних моделей на добре протестованій і добре документованій платформі, а користувачам - мати бібліотеку методів сурогатного моделювання, з якою можна використовувати і порівнювати методи.

Цей пакет містить багато типів сурогатних моделей. Серед них радіальні базисні функції, поліноміальна регресія, кригінг, KPLS, KPLSK та маргінальний гаусовий процес. Також він містить набір стандартних функцій, які можна використовувати для тестування моделей та реалізовані алгоритми вибірки, такі як стохастична вибірка та латинський гіперкуб.

5.1.4 MATLAB

Для симуляції фізичних процесів використано мову MATLAB.

MATLAB(скорочення від "Matrix Laboratory") - це високорівнева мова програмування та середовище розробки, розроблені спеціально для вирішення завдань наукового та технічного характеру. Ось декілька причин, чому варто використовувати MATLAB:

1. Потужні можливості математичного моделювання: MATLAB надає багатий набір інструментів для роботи з лінійною алгеброю, чисельними методами, оптимізацією, статистикою та іншими математичними операціями. Це дозволяє вирішувати широкий спектр завдань, починаючи з простих обчислень і закінчуючи складними математичними моделями. MATLAB надає інструменти для виконання різних математичних операцій, таких як обчислення з матрицями, розв'язання систем лінійних рівнянь, чисельне інтегрування та диференціювання функцій, розв'язання диференціальних рівнянь та багато іншого. Це дозволяє швидко та ефективно вирішувати складні математичні задачі без необхідності вручну розробляти складні алгоритми.

2. Обширна бібліотека функцій: MATLAB має обширну бібліотеку функцій, яка включає спеціалізовані інструменти для обробки сигналів та зображень, символьних обчислень, обробки даних та багато іншого.

3. Інтерактивне середовище розробки: MATLAB надає зручне та інтуїтивно зрозуміле інтерактивне середовище розробки, яке дозволяє швидко прототипувати та налагоджувати код. Можна виконувати окремі команди безпосередньо в командному рядку або створювати та запускати скрипти та функції.

4. Візуалізація даних: MATLAB пропонує потужні інструменти візуалізації даних, які дозволяють створювати графіки, діаграми та інші типи графічних представлень. Це допомагає аналізувати та інтерпретувати результати, а також візуально подавати складні дані.

5. Інтеграція з іншими мовами та додатками: MATLAB легко інтегрується з іншими мовами програмування, такими як C++, Java та Python, що дозволяє використовувати MATLAB як частину більш широких систем. Також існують спеціалізовані інструменти для інтеграції MATLAB з популярними додатками, такими як Microsoft Excel та Simulink.

6. Підтримка спільноти: MATLAB має велику активну спільноту користувачів, де ви можете знайти багато ресурсів, включаючи документацію, форуми, блоги та готові рішення. Ви можете обговорювати свої питання та проблеми, а також ділитися своїми ідеями та проектами з іншими користувачами MATLAB.

5.2. Опис програмної реалізації

Програмна реалізація практичної частини складається з наступних частин:

- Генерація початкової вибірки
- Тренування сурогатної моделі
- Візуалізація ітерацій тренування моделі

Повний лістинг програми знаходиться в додатку А.

Генерацію початкової вибірки робимо за допомогою алгоритму латинського гіперкуба з бібліотеки smt. Оскільки розрахунок точок вхідної моделі зазвичай займає досить багато часу, то виконуємо його заздалегідь і зберігаємо результат у вигляді csv файлу, який потім завантажуємо перед тренуванням.

Для того, щоб оцінювати точність моделі під час тренування, генеруємо дві вибірки - одну для тренування та одну для валідації. Після чого на кожній ітерації тренування моделі розраховуємо метрики MSE, MAE та MAPE для валідаційної вибірки.

Алгоритм тренування моделі має наступний вигляд:

1. Завантажуємо початкову вибірку
2. Тренуємо модель на точках вибірки
3. Додаємо нові точки до вибірки
4. Розраховуємо метрики моделі на валідаційній вибірці
5. Якщо точність модель є задовільною, то зупиняємось, інакше повертаємось до пункту 2

Для ймовірнісних моделей (кригінг та його модифікації) в якості нових точок беремо точки, для яких максимальна дисперсія передбачення сурогатної моделі. Щоб знайти ці точки, генеруємо ще одну вибірку за допомогою алгоритму латинського гіперкубу та вибираємо з неї точки з максимальною дисперсією. Оскільки ця вибірка використовується лише з сурогатною моделлю, яка розраховується набагато швидше, ніж початкова модель, то для цієї вибірки можна згенерувати набагато більше точок, ніж для початкової вибірки.

Для детерміністичних моделей (наприклад, QP) поняття дисперсії передбачення не має сенсу. Тому для них нові точки вибираємо випадковим чином.

Для реалізації сурогатних моделей використовуємо бібліотеку smt. Використовуємо такі моделі:

- QP - поліноміальна регресія другого порядку
- Кригінг
- KPLS
- KPLSK

У якості умови зупинки використовуємо кількість епох або поріг по метриці MAPE.

Візуалізацію процесу тренування виконуємо за допомогою пакету matplotlib.

Для оптимізації отриманої сурогатної моделі було використано функціонал бібліотеки SciPy.

6. РОЗВ'ЯЗОК ТЕСТОВОЇ ЗАДАЧІ ОПТИМІЗАЦІЇ ЗА ДОПОМОГОЮ СУРОГАТНИХ МОДЕЛЕЙ

6.1. Постановка тестової задачі

Задля демонстрації можливостей сурогатних моделей розглянемо оптимізацію моделі лопатки (леза) турбіни. Було обрано саму цю модель оскільки вона є відносно складною в обчисленнях, що дасть змогу краще продемонструвати переваги сурогатних моделей.

Лопатка турбіни - це ключовий компонент газових і парових турбін, що використовуються в енергетиці, авіаційних двигунах і різних галузях промисловості. Лопатка призначена для ефективного вилучення енергії з рідини (зазвичай повітря або пари), що проходить над нею, перетворюючи кінетичну енергію в механічну роботу.

Коротко опишемо дану модель. Об'єктом оптимізації є лезо турбіни реактивного двигуна, яка контактує з гарячими газами з камери згорання, за допомогою чого і відбувається обертання ротора. А оскільки відбувається взаємодія між газом та лезом, то є ймовірність її деформації, що може призвести до пошкодження всієї системи, таким чином є важливим проаналізувати при яких умовах вхідних змінних деформація та навантаження перестануть бути для нас прийнятними.

Модель турбіни представлена у форматі .stl. Він зазвичай використовується для представлення тривимірної геометрії поверхні та розшифровується як "стереолітографія" і широко підтримується програмним забезпеченням для 3D-моделювання та 3D-принтерами. Модель лопатки турбіни у форматі .stl представляє її зовнішню поверхню у вигляді набору трикутних граней. Ці грані апроксимують гладку поверхню лопатки, а роздільна здатність трикутників визначає рівень деталізації моделі.

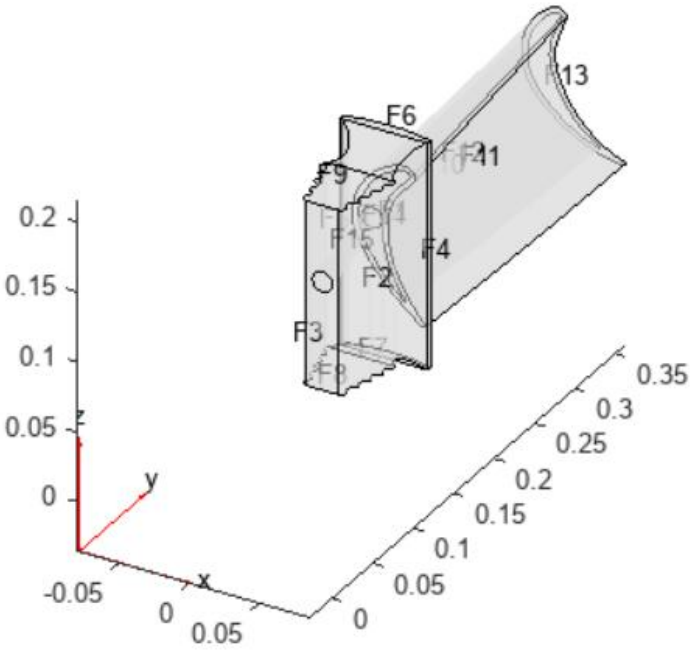

Рис. 6.1 Креслення досліджуваного леза турбіни [28]

Оскільки необхідно врахувати вплив температури та тиску, вхідними змінними в нашому випадку є:

- Температура продуктів горіння($T_{горіння}$)
- Температура повітря, яке охолоджує лезо($T_{повітря}$)
- Коефіцієнт теплопередачі продуктів горіння($h_{горіння}$)
- Коефіцієнт теплопередачі повітря($h_{повітря}$)
- Тиск на частину леза, на яку тиснуть гази($P_{газу}$)
- Тиск на частину леза, де відбувається всмоктування газів($P_{всмок}$)

На виході отримуємо два значення:

- Максимальна деформація(D)
- Максимальна напруга фон Мізеса (S). Напруга фон Мізеса є рушійною силою пошкодження багатьох пластичних інженерних матеріалів.

Таким чином, функції, які описують модель:

$$f_1\left(T_{горіння}, T_{повітря}, h_{горіння}, h_{повітря}, P_{газу}, P_{всмок}\right) = S$$

$$f_2\left(T_{горіння}, T_{повітря}, h_{горіння}, h_{повітря}, P_{газу}, P_{всмок}\right) = D$$

Задача полягає в мінімізації максимальної деформації та напруги.

Лістинг коду, який моделює лопатку турбіни, знаходиться в додатку Б.

Таким чином маємо задачу багатокритеріальної оптимізації.

6.2. Створення сурогатних моделей

Тренування моделей відбувалося протягом 20-и ітерацій (окрім QP). В початковій вибірці було взято 5 точок, після цього на кожній ітерації додавалося по 5 нових точок. Для роботи методу QP потрібно мінімум 28 точок, оскільки це мінімальна кількість точок, яка необхідна, щоб однозначно визначити параболу в шестивимірному просторі. Тому для методу QP тренування починалося з 30 точок.

На рис. 6.2-6.4. зображено історію тренування цих моделей. Щоб порівняти різні моделі, по осі абсцис відкладено кількість точок, використаних для тренування, оскільки ця метрика має більше сенсу, ніж кількість ітерацій.

У табл. 6.1 наведено фінальні результати тренування моделей на повній вибірці. Найкращі результати показали моделі кригінгу та KPLSK. Для розв’язання поставленої задачі будемо використовувати кригінг.

Табл. 6.1. Фінальні результати сурогатного моделювання

	MSE	MAE	MAPE
Kriging	0.021	0.105	0.607 %
KPLS	0.273	0.348	2.047 %
KPLSK	0.021	0.105	0.607 %
QP	0.219	0.319	1.841 %

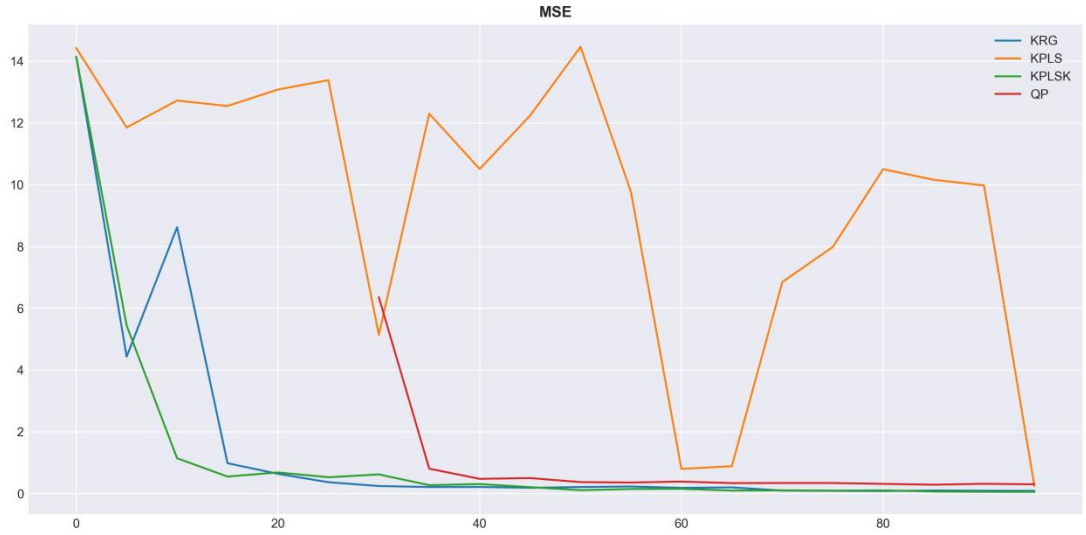

Рис. 6.2. Графік середньоквадратичної помилки (MSE) в залежності від кількості точок, використаних для побудови сурогатної моделі

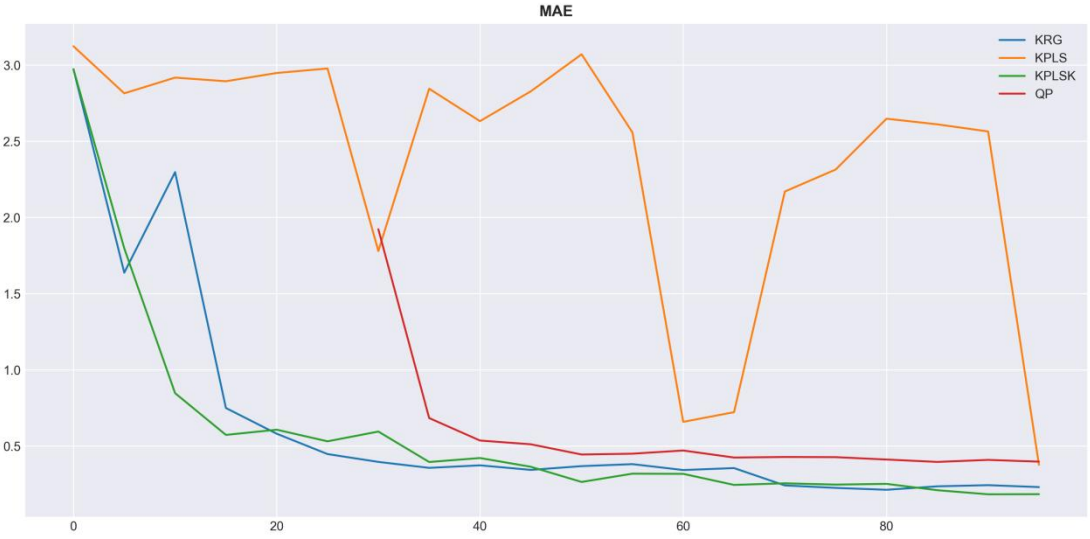

Рис. 6.3. Графік середньої абсолютної помилки (MAE) в залежності від кількості точок, використаних для побудови сурогатної моделі

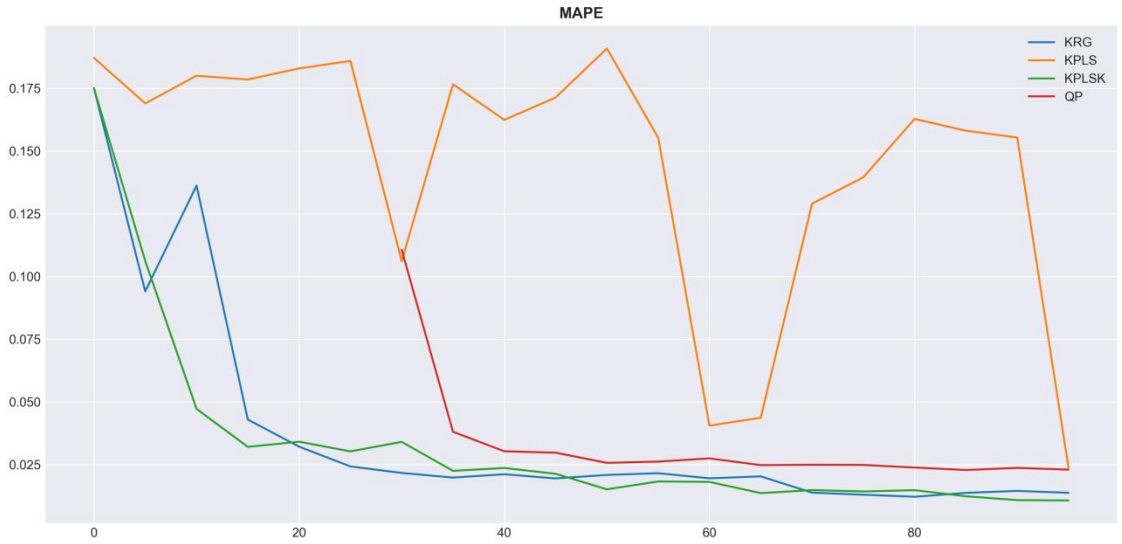

Рис. 6.4. Графік середньої абсолютної відсоткової помилки (MAPE) в залежності від кількості точок, використаних для побудови сурогатної моделі

6.3. Оптимізація моделі

Для розв'язку поставленої задачі багатокритеріальної мінімізації було використано метод лінійної скаляризації. Для того, щоб обидва критерії мали однаковий масштаб, було застосовано нормалізацію за допомогою функції MinMaxScaler бібліотеки scikit-learn. Таким чином була мінімізована функція

$$f(x) = w_1\tilde{f}_1(x) + (1 - w_1)\tilde{f}_2(x)$$

де $\tilde{f}_1(x)$ та $\tilde{f}_2(x)$ - нормалізовані функції.

Для побудови Парето фронту було взято 100 значень w_1 рівномірно розподілених на відрізку [0, 1].

Отримані точки зображено на рис. 6.5. Результати показали, що в даному випадку Парето фронт вироджується в одну точку. Це можна перевірити, обчисливши максимальне відносне відхилення кожної координати:

$$\frac{f_1^{max} - f_1^{min}}{f_1^{avg}} = 3.4 * 10^{-5}$$

$$\frac{f_2^{max} - f_2^{min}}{f_2^{avg}} = 6.1 * 10^{-5}$$

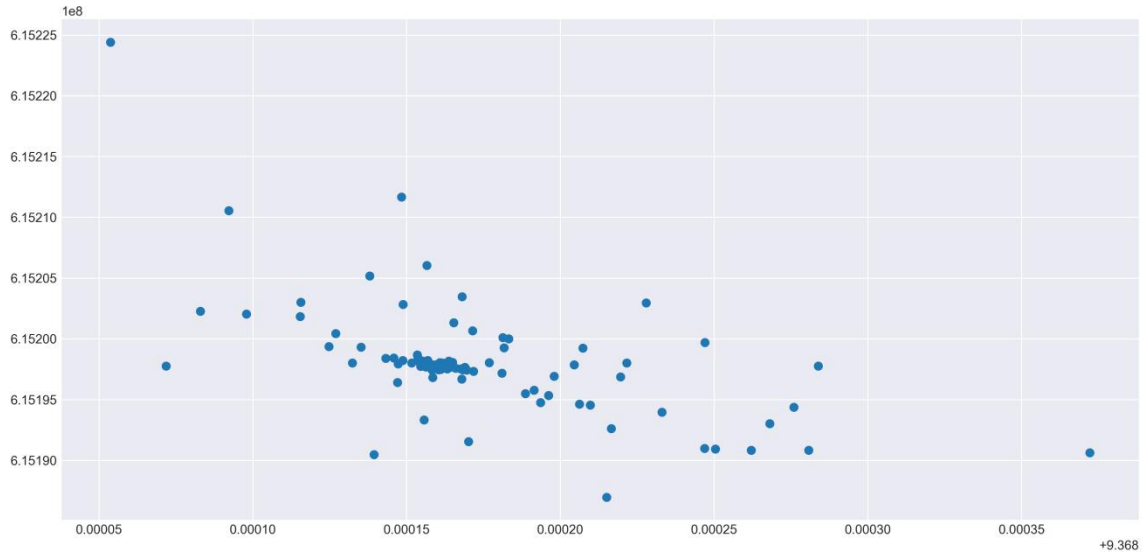

Рис. 6.5. Точки отримані в результаті оптимізації сурогатної моделі

З точністю до чисельних похибок маємо одну точку. Тобто маємо одну Парето-оптимальну точку, яка є оптимальною за обома критеріями.

7. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

7.1. Постановка задачі проектування

У даному розділі проводиться оцінка основних характеристик програмного продукту, розробленого для вирішення задачі “Використання сурогатних моделей для оптимізації систем керування складними об’єктами”.

Функціонально-вартісний аналіз (ФВА) - це методологія, що використовується для вивчення і вдосконалення продуктів або систем з точки зору їх функціональності та вартості. Вона включає в себе аналіз функцій, які виконує продукт або система, і вартості, пов'язаної з цими функціями.

У ФВА розглядаються різні аспекти продукту або системи, такі як функції, які вони виконують, елементи, з яких вони складаються, взаємозв'язки між цими елементами, а також вартість, пов'язана з кожною функцією або елементом. Метою ФВА є забезпечення оптимального співвідношення між функціональністю і вартістю продукту або системи.

ФВА може бути застосований у різних галузях, включаючи розробку нових продуктів, оптимізацію виробничих процесів, підвищення якості продукції і зниження витрат. Він дозволяє виявляти недоліки і можливості для вдосконалення продукту або системи, сприяє прийняттю обґрунтованих рішень щодо його подальшого розвитку і допомагає знижувати витрати без втрати функціональності.

Технічні вимоги до продукту, які були визначені під час виконання роботи:

- Здатність моделювати складні системи або процеси, відображаючи їх поведінку, взаємодію та динаміку.
- Здатність інтегруватися з існуючими системами або програмними рішеннями для обміну даними та інформацією.

- Забезпечення можливості моделювання великих обсягів даних та комплексних систем без значного зниження продуктивності.
- Можливість використовувати розподілені обчислювання та запускати на обчислювальних кластерах

7.2 Обґрунтування функцій та параметрів програмного продукту

Головною функцією F0 є розробка програмного продукту, для вирішення задачі “Використання сурогатних моделей для оптимізації систем керування складними об’єктами”.

Основні функції програмного продукту:

- F_1 – вибір мови програмування
- F_2 – вибір бібліотек для чисельних методів
- F_3 – вибір середовища розробки

Варіанти реалізації кожної з функцій:

- Функція F_1: а) Python, б) Matlab, в) Wolfram Language
- Функція F_2: а) NumPy, б) Partial Differential Equation Toolbox, в) Mathematica
- Функція F_3: а) Jupyter Notebook, б) PyCharm, в) Matlab, г) Wolfram Mathematica

Функціонал для роботи з чисельними методами вже вбудований в Matlab та Wolfram.

На рис. 7.1 наведено морфологічну карту варіантів реалізації основних функцій.

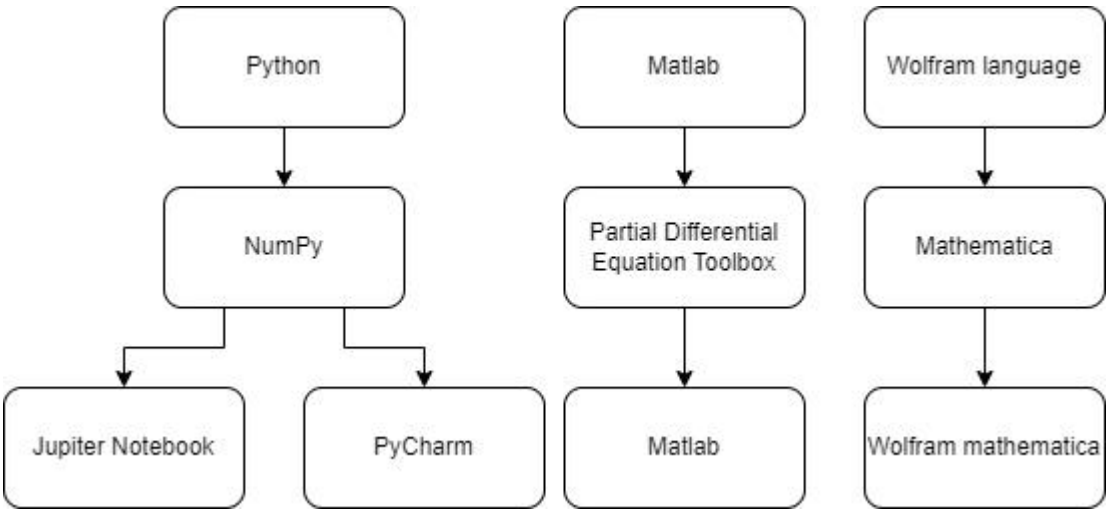

Рис. 7.1. Морфологічна карта варіантів реалізації функцій

У табл. 7.1 наведено позитивно-негативну матрицю варіантів основних функцій.

Табл. 7.1. Позитивно-негативна матриця

Функція	Варіант реалізації	Переваги	Недоліки
F_1	А	Кросплатформеність, безкоштовність, можливість інтеграції з іншими мовами та фреймворками	Низька швідкість роботи, проблеми з паралельним програмуванням
	Б	Кросплатформеність, оптимізований під чисельні розрахунки, можливість інтеграції з іншими мовами та	Ціна, обмеженість доступних бібліотек

		фреймворками	
	В	Кросплатформеність, символьні розрахунки	Ціна, низька швідкість роботи, ускладнена інтеграція з іншими мовами
F_2	А	Швидкодія, можливість простого аналізу даних та візуалізації, надійність	Підтримується лише мовою Python
	Б	Надійність	Підтримується лише мовою Matlab
	В	Надійність	Підтримується лише мовою Wolfram
F_3	А	Простота встановлення та налаштування, можливість виконання на віддаленому сервері	Мала кількість підтримуваних мов
	Б	Зручний інтерфейс, багато інструментів,	Обмежений функціонал в

		підтримує багато мов програмування	безкоштовній версії
	В	Можливість інтерактивної розробки	Підтримує лише мову Matlab, ціна
	Г	Можливість інтерактивної розробки	Підтримує лише мову Wolfram, ціна

Аналізуючи позитивно-негативну матрицю, можемо зробити висновок, що деякі з варіантів реалізацій функцій можна відкинути, бо вони не задовольняють задачам.

Функція F_1:

Можна використовувати усі варіанти.

Функція F_2:

Можна використовувати усі варіанти.

Функція F_3:

Оскільки PyCharm містить набагато більше функціоналу, ніж Jupyter Notebook, то відкидаємо варіант А (Jupyter Notebook).

Таким чином, будемо розглядати такі варіанти реалізації:

- F_1а - F_2а - F_3б
- F_1б - F_2б - F_3в
- F_1в - F_2в - F_3г

Для оцінювання якості розглянутих функцій далі розглядаємо вибір системи параметрів.

Для характеристики програмного продукту використаємо наступні параметри:

- X_1 - потенційний об’єм програмного коду
- X_2 - швидкодія операцій мови програмування
- X_3 - споживання пам’яті під час роботи програми

Кращі, середні та гірші значення параметрів вибираються на основі умов, що характеризують експлуатацію програмного продукта та вимог замовника, як наведено у табл. 7.2.

Табл. 7.2. Основні параметри програмного продукту

Опис параметру	Умовні позначення	Одиниці виміру	Значення параметру		
			гірші	середні	кращі
Потенційний об’єм коду	X_1	Рядки коду	700	500	300
Швидкодія мови програмування	X_2	оп/с	10^6	10^7	10^8
Об’єм пам’яті	X_3	Мб	500	300	100

За даними табл. 7.2 будуємо графічні характеристики параметрів (рис. 7.2-7.4).

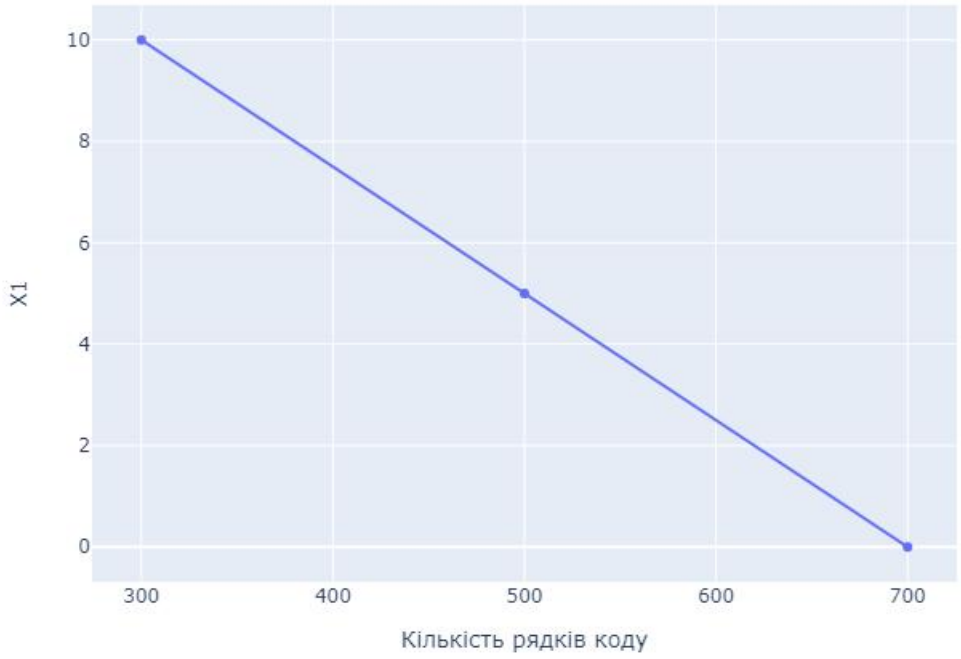

Рис. 7.2. Потенційний об’єм коду

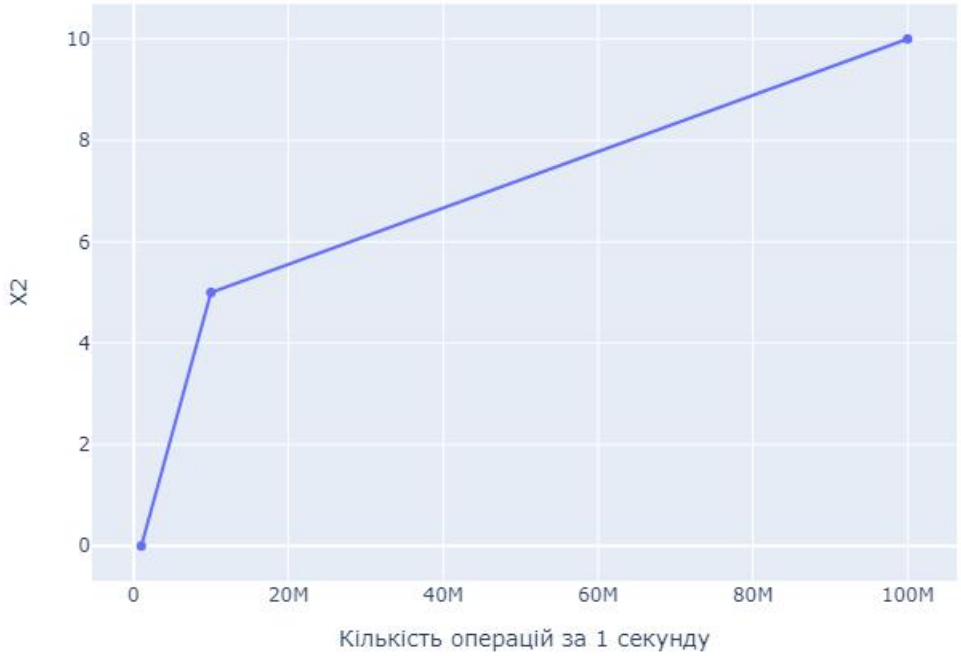

Рис. 7.3. Швидкодія мови програмування

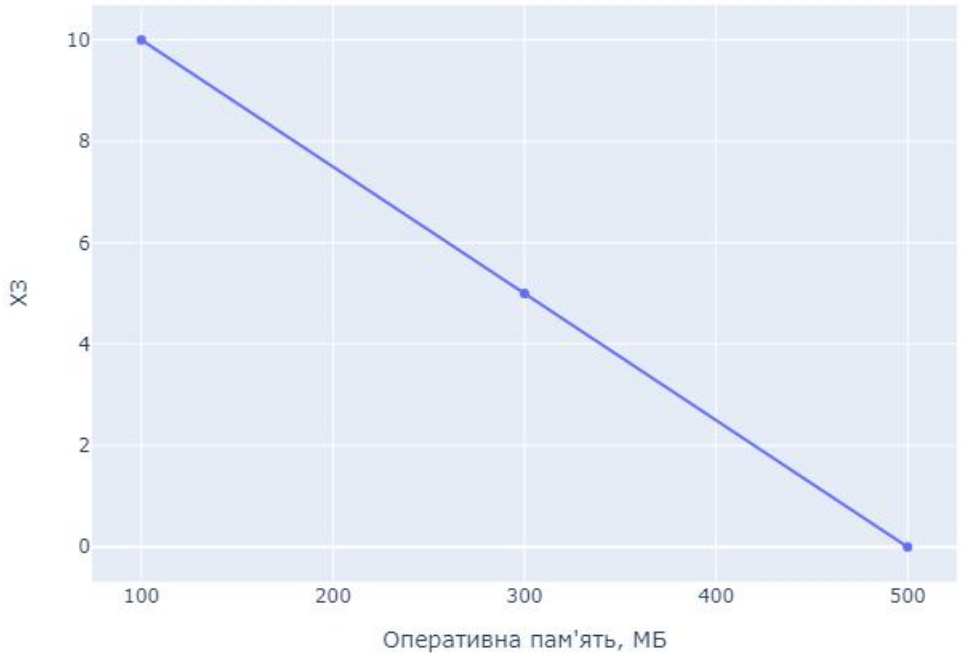

Рис. 7.4. Об’єм пам’яті

Після визначення основних параметрів, проводиться їх ранжування за допомогою методу попарного порівняння комісією з 7 людей. У результаті детального обговорення та аналізу кожний експерт оцінює ступінь важливості кожного параметру, беручи до уваги мету проекту.

Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів
- перевірку придатності експертних оцінок для подальшого використання
- визначення оцінки попарного пріоритету параметрів
- обробку результатів та визначення коефіцієнту значимості

Результати експертного ранжування представлено в табл. 7.3.

Табл. 7.3. Результати експертного ранжування показників

Парамет		Сума	Відхилення	Δ_i^2

р	1	2	3	4	5	6	7	рангів	Δ_i	
X_1	2	2	1	2	1	2	2	12	-2	4
X_2	3	3	3	3	3	3	3	21	7	49
X_3	1	1	2	1	2	1	1	9	-5	25
Разом	6	6	6	6	6	6	6	42	0	78

Для перевірки ступеню достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} = 42,$$

де r_{ij} – ранг *i*-го параметра, визначений *j*-м експертом;

N – число експертів.

б) середня сума рангів *T*:

$$T = \frac{1}{n} R_i = 14.$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T.$$

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{n} \Delta_i^2 = 78.$$

д) коефіцієнт узгодженості (конкордації):

$$\mathrm{W} = \frac{12S}{\mathrm{N}^2(\mathrm{n}^3 - \mathrm{n})} = \frac{12 \cdot 78}{7^2(3^3 - 3)} = 0.796 > \mathrm{W_k} = 0{,}67.$$

Ранжирування можна вважати достовірним, оскільки знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Далі проводимо попарне порівняння всіх параметрів, результати якого наведені у таблиці 6.4. Числове значення, що визначає ступінь переваги *i*-го параметра над *j*-тим, a_{ij} визначається за формулою:

$$a_{ij} = \begin{cases} 1{,}5 & x_i > x_j \\ 1{,}0 & x_i = x_j \\ 0{,}5 & x_i < x_j \end{cases}.$$

Табл. 7.4. Результати ранжування параметрів

Параметри	Експерти							Підсумкова оцінка	Числове значення коефіцієнтів переваги
	1	2	3	4	5	6	7		
$X_1,\ X_2$	<	<	<	<	<	<	<	<	0.5
$X_1,\ X_3$	>	>	<	>	<	>	>	>	1.5
$X_2,\ X_3$	>	>	>	>	>	>	>	>	1.5

З отриманих числових оцінок переваги складаємо матрицю $A = \|a_{ij}\|$. Для кожного параметра розрахунок вагомості K_{B_i} проводиться за наступною формулою:

$$K_{B_i} = \frac{b_i}{\sum_{i=1}^{n} b_i},$$

де $b_i = \sum_{j=1}^{N} a_{ij}$ – вагомість i-го параметра за результатами оцінок всіх експертів;

a_{ij} –коефіцієнт переваги i-го над j-тим параметром.

Відносні оцінки розраховуються декілька разів до тих пір, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступною формулою:

$$K_{B_i} = \frac{b_i^{'}}{\sum_{i=1}^{n} b_i^{'}},$$

де $b_i^{'} = \sum_{j=1}^{N} a_{ij} b_j$.

Табл. 7.5. Розрахунок вагомості параметрів

i	j			Перша ітерація		Друга ітерація	
	X_1	X_2	X_3	B_i	K_{B_i}	B_i^1	$K_{B_i}^1$
X_1	1	0.5	1.5	3	0.333	8	0.32
X_2	1.5	1	1.5	4	0.444	11.5	0.46
X_3	0.5	0.5	1	2	0.222	5.5	0.22
Разом				9	1	25	1

Різниця значень коефіцієнтів вагомості після другої ітерації не перевищує 2%, тому додаткові ітерації не потрібні.

7.3 Економічний аналіз варіантів розробки програмного продукту

Табл. 7.6. Розрахунок показників якості

Варіант реалізації	Бальна оцінка параметру			Коефіцієнт вагомості параметру			Коефіцієнт якості параметру			Коефіцієнт якості
	X_1	X_2	X_3	X_1	X_2	X_3	X_1	X_2	X_3	
F_1а - F_2а - F_3б	10	5	5	0.32	0.46	0.22	3.2	2.3	1.1	6.6
F_1б - F_2б - F_3в	5	10	10				1.6	4.6	2.2	8.8
F_1в - F_2в - F_3г	0	0	0				0	0	0	0

Найкращий коефіцієнт якості має варіант 2 (Matlab), за ним з невеликим відривом йде варіант 1 (Python) і значно відстає варіант 3 (Wolfram).

7.4 Економічний аналіз варіантів розробки програмного продукту

Задля оцінки вартості розробки продукту необхідно провести розрахунки трудомісткості.

Виділимо два завдання на які можна поділити розробку

1. Опис та моделювання об’єкту оптимізації
2. Розробка алгоритмів сурогатного моделювання

Визначимо ступінь новизни для цих завдань. Перше, у зв’язку з необхідністю моделювання нового об’єкта проте за допомогою вже відомих методів має групу Б. Друге ж, оскільки необхідно реалізувати вже відносно відомі алгоритми проте з певними модифікаціями, отримує групу В. Оцінка складності для цих завдань наступна: перше має групу 1, друге - третю.

Для реалізації завдання 1 використовується інформація у вигляді складної фізичної моделі, а завдання 2 використовує стандартні методи сурогатної. Загальна трудомісткість обчислюється за формулою.

$$\mathrm{Т_О} = \mathrm{Т_Р} \cdot \mathrm{К_П} \cdot \mathrm{К_{СК}} \cdot \mathrm{К_М} \cdot \mathrm{К_{СТ}} \cdot \mathrm{К_{СТ.М}}$$

де

- $Т_Р$ – трудомісткість розробки програмного продукту;
- $К_П$ – поправочний коефіцієнт;
- $К_{СК}$ – коефіцієнт на складність вхідної інформації;
- $К_М$ – коефіцієнт рівня мови програмування;
- $К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;
- $К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни Б та групи складності алгоритму 1, трудомісткість дорівнює: $T_P = 64$ людино-днів. Поправочний коефіцієнт, який враховує вид вхідної інформації для першого завдання: $K_П = 1{,}021$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації рівний $K_{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ} = 0{,}8$. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 64 \cdot 1.021 \cdot 0.8 = 52{,}2752 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для другого завдання, в якому використовується алгоритм третьої групи складності зі ступенем новизни В, тобто $T_P = 20$ людино-днів, $K_П = 0{,}6$, $K_{СК} = 1$, $K_{СТ} = 0{,}8$:

$$T_2 = 20 \cdot 0.6 \cdot 0.8 = 9{,}6 \text{ людино-днів.}$$

Оскільки загальна трудомісткість усіх варіантів реалізації збігаються, їх можна об’єднати в одну групу.

Загальна трудомісткість складає:

$$T_O = (52{,}2752 + 9.6) \cdot 2 = 123{,}75 \text{ людино-годин;}$$

В розробці беруть участь два програмісти з окладом 30000 грн., та один спеціаліст з прикладної фізики з окладом 22000 грн. Визначимо середню зарплату за годину за формулою:

$$C_ч = \frac{M}{T_m \cdot t} \, грн.,$$

де M – місячний оклад працівників;

T_m – кількість робочих днів на місяць;

t – кількість робочих годин в день.

$$C_ч = \frac{30000 + 30000 + 22000}{3 \cdot 20 \cdot 8} = 170{,}83 \, грн.$$

Тоді, розрахуємо заробітну плату за формулою:

$$C_{ЗП} = C_ч \cdot T_i \cdot K_Д,$$

де $C_ч$ – величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

$K_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$C_{ЗП} = 170{,}83 \cdot 123{,}75 \cdot 1{,}2 = 21\,140{,}21 \, грн.$$

Відрахування на соціальний внесок становить 22%:

$$C_{СВ} = C_{ЗП} \cdot 0{,}22 = 21\,140{,}21 \cdot 0{,}22 = 4\,650 \, грн.$$

Тепер визначимо витрати на оплату однієї машино-години. Оскільки одна ЕОМ обслуговується одним інженером апаратного забезпечення з окладом 15000 грн. та коефіцієнтом зайнятості $К_З = 0{,}2$ то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 15\,000 \cdot 0{,}2 = 36\,000\ грн.$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot \left(1 + К_З\right) = 36\,000 \cdot (1 + 0{,}2) = 43\,200\ грн.$$

Відрахування на соціальний внесок становить 22%:

$$С_{СВ} = С_{ЗП} \cdot 0{,}22 = 43200 \cdot 0{,}22 = 9\,504\ грн.$$

Амортизаційні відрахування розраховуємо за формулою при амортизації 25% та вартості ЕОМ – 35 000 грн.:

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1{,}15 \cdot 0{,}25 \cdot 35000 = 10\,062{,}5\ грн.$$

де $К_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$ – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо за формулою:

$$С_Р = К_{ТМ} \cdot К_Р \cdot Ц_{ПР} = 1{,}15 \cdot 0{,}05 \cdot 35000 = 2\,012{,}5\ грн.$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = \left(Д_К - Д_В - Д_С - Д_Р\right) \cdot t \cdot К_В,\ Т_{ЕФ} = (365 - 104 - 12 - 16) \cdot 8 \cdot 0{,}9 = 1\,667{,}6\ год.$$

де $Д_К$ – календарна кількість днів у році;

$Д_В, Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

$К_В$ – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕЛ} = 1\,677{,}6 \cdot 0{,}65 \cdot 0{,}2 \cdot 4{,}79 = 1\,044{,}64\ грн.$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$ – коефіцієнтом зайнятості приладу;

$Ц_{ЕЛ}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0{,}67 = 35\,000 \cdot 0{,}67 = 23\,450\ грн.$$

Тоді, річні експлуатаційні витрати будуть складати:

$$С_{ЕК} = С_{ЗП} + С_{СВ} + С_А + С_Р + С_{ЕЛ} + С_Н,$$

$$С_{ЕК} = 43\,200 + 9\,504 + 10\,062{,}5 + 2\,012{,}5 + 1\,044{,}64 + 23\,450 = 89\,273{,}64\ грн.$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{МГ} = \frac{С_{ЕК}}{Т_{ЕФ}} = \frac{89\,273{,}64}{1\,677{,}6} = 53{,}53\,грн/год.$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу складають:

$$С_М = С_{МГ} \cdot Т = 53{,}53 \cdot 123{,}75 = 6\,624{,}85\ грн.$$

Накладні витрати складають 67% від заробітної плати:

$$C_Н = C_{ЗП} \cdot 0{,}67 = 21\,140{,}21 \cdot 0{,}67 = 14\,163{,}8\ грн.$$

Отже, вартість розробки програмного продукту за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{СВ} + C_М + C_Н,$$

$$C_{ПП} = 21\,140{,}21 + 9\,504 + 6\,624{,}85 + 14\,163{,}8 = 51\,432{,}86\ грн.$$

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{ТЕР} = \frac{K_К}{C_{ПП}} = \frac{7{,}16}{51\,432{,}86} = 0{,}139 \cdot 10^{-5}$$

ВИСНОВКИ

У даній роботі було розглянуто різні типи сурогатних моделей та їх застосування для оптимізації складних систем. Була розроблена програмна реалізація розглянутих методів та протестована на конкретному прикладі – моделі лопатки турбіни. Для цього були використані мови програмування Python та MATLAB і бібліотеку SMT.

У першому розділі було проведено огляд існуючої літератури та доведено актуальність теми сурогатних моделей.

В другому розділі була розглянута проблема однокритеріальної оптимізації, основні поняття пов’язані з цією проблемою та методи її розв'язання.

У третьому розділі було визначено поняття парето-оптимальності і представлені основні методи розв'язання багатокритеріальних задач: різні види скаляризації, генетичні алгоритми та багатокритеріальні градієнтні методи.

У четвертому розділі було представлено та детально проаналізовано загальний алгоритм оптимізації складних систем із використанням сурогатних моделей. Показана важливість попереднього дослідження об’єкту оптимізації, правильного підходу до створення початкової вибірки та вибору сурогатної моделі.

У п'ятому розділі було розглянуто деталі побудови сурогатних моделей. Було обґрунтовано вибір мов програмування та бібліотек, наведено опис програмної реалізації.

У шостому розділі було протестовано створену програмну реалізацію на прикладі багатокритеріальної оптимізації моделі лопаток турбіни.

У сьомому розділі було проведено функціонально-вартісний аналіз розробленого програмного продукту.

Побудована система демонструє високий рівень точності моделювання та значний приріст у швидкості та може бути використана як для подальших наукових досліджень так і для бізнес потреб.

Основними напрямками до розвитку продукту можуть слугувати впровадження більш широкого кола сурогатних моделей (наприклад, із використанням штучних нейронних мереж) та більш ефективне використання можливості паралельних обчислень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Jin, H., Chen, X., & Wang, L. (2018). A comprehensive survey of surrogate modeling for optimization. Structural and Multidisciplinary Optimization, 57(6), 2353-2383. [DOI: 10.1007/s00158-017-1799-9]

[2] Forrester, A., Sobester, A., & Keane, A. (2008). Engineering design via surrogate modelling: a practical guide. John Wiley & Sons.

[3] Queipo, N. V., Haftka, R. T., Shyy, W., Goel, T., Vaidyanathan, R., & Tucker, P. K. (2005). Surrogate-based analysis and optimization. Progress in Aerospace Sciences, 41(1), 1-28. [DOI: 10.1016/j.paerosci.2005.02.001]

[4] Queipo, N. V., Chick, J. H., Goh, C. J., & Venter, G. (2008). Surrogate-based analysis and optimization of systems with high-dimensional stochastic input variables. Engineering Optimization, 40(6), 559-578.

[5] Koziel, S., & Leifsson, L. (2013). Surrogate-based Modeling and Optimization: Applications in Engineering. Springer.

[6] Simpson, T. W., Poplinski, J. D., & Koch, P. N. (2001). Metamodels for computer-based engineering design: Survey and recommendations. Engineering with Computers, 17(2), 129-150.

[7] Rasmussen, C. E., & Williams, C. K. (2006). Gaussian Processes for Machine Learning. MIT Press.

[8] Seeger, M. (2004). Gaussian Processes for Machine Learning. International Journal of Neural Systems, 14(2), 69-106.

[9] Williams, C. K., & Rasmussen, C. E. (1996). Gaussian Processes for Regression. In Advances in Neural Information Processing Systems.

[10] Chiles, J. P., & Delfiner, P. (2012). Geostatistics: Modeling Spatial Uncertainty (2nd ed.). John Wiley & Sons.

[11] Cressie, N. (1993). Statistics for Spatial Data (Revised ed.). Wiley Series in Probability and Statistics.

[12] Stein, M. L. (1999). Interpolation of Spatial Data: Some Theory for Kriging. Springer.

[13] Montgomery, D. C., Peck, E. A., & Vining, G. G. (2012). Introduction to Linear Regression Analysis (5th ed.). John Wiley & Sons.

[14] Draper, N. R., & Smith, H. (1998). Applied Regression Analysis (3rd ed.). John Wiley & Sons.

[15] Wampler, C. W. (2010). Regression modeling strategies: with applications to linear models, logistic and ordinal regression, and survival analysis. Springer.

[16] Jin, Y. (2005). A comprehensive survey of fitness approximation in evolutionary computation. Soft Computing, 9(1), 3-12.

[17] Lim, M. H., & Lee, K. Y. (2008). Hybrid genetic algorithm-based optimization using a surrogate model for the design of a disc brake system. Structural and Multidisciplinary Optimization, 36(3), 233-243.

[18] Jin, Y., & Sendhoff, B. (2003). Pareto-based multi-objective machine learning: An overview and case studies. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 33(2), 123-139.

[19] Bingham, D., & Ranjan, P. (2018). Surrogate models in engineering design: A review. Journal of Mechanical Design, 140(7), 070801.

[20] Keane, A. J., & Nair, P. B. (2006). Computational approaches for aerospace design: the pursuit of excellence. John Wiley & Sons.

[21] Zhang, Y., & Zhang, J. (2018). Multi-objective aerodynamic shape optimization using improved surrogate models. Aerospace Science and Technology, 74, 422-431. DOI: 10.1016/j.ast.2018.01.045

[22] Queipo, N. V., Haftka, R. T., Shyy, W., Goel, T., Vaidyanathan, R., & Tucker, P. K. (2005). Surrogate-based analysis and optimization. Progress in Aerospace Sciences, 41(1), 1-28.

[23] Chang, J., Zhao, Q., Shen, W., Wu, C., & Jiang, W. (2017). Multi-objective optimization design of automotive body in white considering crashworthiness and lightweight. Structural and Multidisciplinary Optimization, 56(5), 1031-1052. DOI: 10.1007/s00158-017-1689-1

[24] Ding, Y., & Liao, W. (2014). Wind turbine design optimization under multiple operating conditions using a surrogate-assisted evolutionary algorithm. Renewable Energy, 63, 659-668. DOI: 10.1016/j.renene.2013.10.036

[25] Chu, H., Lin, H., Liu, Y., Wang, G., & Su, W. (2017). Surrogate modeling assisted multi-objective optimization of an industrial hydrocracking process. Computers & Chemical Engineering, 97, 53-64. DOI: 10.1016/j.compchemeng.2016.10.008

[26] Zhang, W., Zhang, Y., & Liu, C. (2016). Surrogate modeling for air quality prediction. Environmental Modelling & Software, 85, 339-349. DOI: 10.1016/j.envsoft.2016.09.016

[27] Brian Kolo (2011). Single and Multiple Objective Optimization. Weatherford Press.

[28] Miettinen, K. (2012). Nonlinear Multiobjective Optimization. Springer.

[29] McKay, M. D., Beckman, R. J., & Conover, W. J. (1979). Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. Technometrics, 21(2), 239-245.

[30] Buhmann, M. D. (2003). Radial Basis Functions: Theory and Implementations. Cambridge University Press.

[31] Rosipal, R., & Trejo, L. J. (2001). Kernel partial least squares regression in reproducing kernel Hilbert space. Journal of Machine Learning Research, 2(1), 97-123.

[32] Numerical Python. [Електронний ресурс] - Режим доступу: https://numpy.org/

[33] Surrogate Modeling Toolbox [Електроний ресурс] - Режим доступу: https://smt.readthedocs.io/en/latest/

ДОДАТОК А

Лістинг програми сурогатного моделювання та оптимізації

```
import matlab.engine
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error, mean_absolute_percentage_error
from sklearn.preprocessing import MinMaxScaler
from smt.sampling_methods import LHS
from smt.surrogate_models import KRG, KPLS, KPLSK, MGP, RBF, QP, IDW, LS
from tqdm.notebook import tqdm
from collections import defaultdict
import scipy
import time

plt.style.use('seaborn-v0_8-darkgrid')

eng = matlab.engine.start_matlab()
hard_model = eng.HardModel()

limits = np.array([[20.0, 40.0], [120.0, 180.0], [30.0, 70.0], [800.0, 1200.0], [1e5, 10e5], [1e5, 10e5]])

def f1(x):
    return eng.runModel(hard_model, *x, nargout=1)
```

```
def f2(x):
    return eng.runModel(hard_model, *x, nargout=2)[1]

def f12(x):
    return eng.runModel(hard_model, *x, nargout=2)

def predict(f, x, verbose=True):
    y = []
    for xi in tqdm(x, disable = not verbose):
        yi = f(xi)
        y.append(yi)
    return np.array(y)

def generate_points(f, sampling, num_points):
    x = sampling(num_points)
    y = predict(f, x)
    return x, y

def save_data(x: np.ndarray, y: np.ndarray, filename: str):
    data = np.concatenate([x, y.reshape((-1, 1))], axis=1)
    columns = [f'x_{i}' for i in range(x.shape[1])] + ['y']
    df = pd.DataFrame(data=data, columns=columns)
    df.to_csv(index=False, path_or_buf=f'data/{filename}')
    return df

def load_data(filename: str):
```

```
    data_df = pd.read_csv(f'data/{filename}')
    data_df = data_df.sample(frac = 1) # random shuffle
    return data_df.drop(columns=['y']).values, data_df['y'].values

def generate_and_save_data(train_size, validation_size):
    train_sampling = LHS(xlimits=limits)
    x_train, y_train = generate_points(f1, train_sampling, train_size)
    validation_sampling = LHS(xlimits=limits)
    x_validation, y_validation = generate_points(f1, validation_sampling,
validation_size)
    save_data(x_train, y_train, "train.csv")
    save_data(x_validation, y_validation, "validation.csv")

x_train, y_train = load_data("train.csv")
x_train_small, y_train_small = load_data("train_small.csv")
x_validation, y_validation = load_data("validation.csv")

# x_train_small, y_train_small = generate_points(f1, LHS(xlimits=limits), 5)
# save_data(x_train_small, y_train_small, "train_small.csv")

def train_model(model_class, x, y, model_params={}):
    sm = model_class(print_training=False, print_problem=False,
print_prediction=False,
                     print_solver=False, print_global=False,
**model_params)
    sm.set_training_values(x, y)
    sm.train()
```

```
    return sm

def train_and_test_model(model_name, x_train, y_train, x_validation, y_validation, model_params={}):
    model = train_model(model_name, x_train, y_train, model_params=model_params)
    return model, calculate_metrics(model, x_validation, y_validation)

def calculate_metrics(model, x_validation, y_validation):
    y_predicted = model.predict_values(x_validation)
    return {
        'mse': mean_squared_error(y_validation, y_predicted),
        'mae': mean_absolute_error(y_validation, y_predicted),
        'mape': mean_absolute_percentage_error(y_validation, y_predicted)
    }

def select_new_points(model, grid, n_points):
    variances = model.predict_variances(grid)
    max_n_indices = np.argsort(variances, axis=0)[-n_points:]
    new_points = grid[max_n_indices.flatten()]
    return new_points

from smt.surrogate_models import MGP
model = train_model(MGP, x_train, y_train)
calculate_metrics(model, x_validation, y_validation)

model.predict_variances(x_validation)
```

```
def train(model_name, f, x_train_initial, y_train_initial, x_validation, y_validation, limits, new_points_grid_n, epoch_new_points_n, epochs = None, mape_threshold = None):
    x_train = x_train_initial
    y_train = y_train_initial
    metrs = []
    global mape
    mape = 1
    epoch = 1
    model = None

    def iter():
        global mape
        if epochs is None:
            while mape >= mape_threshold:
                yield
        else:
            for _ in range(epochs):
                yield

    for _ in tqdm(iter(), total=epochs):
        if epoch != 1:
            new_points_grid = LHS(xlimits=limits)(new_points_grid_n)
            new_points = select_new_points(model, new_points_grid, epoch_new_points_n)
            new_points_y = predict(f, new_points, verbose=False)
```

```
            x_train = np.vstack((x_train, new_points))
            y_train = np.concatenate((y_train, new_points_y))

        model, metr = train_and_test_model(model_name, x_train, y_train, x_validation, y_validation)
        print(f'Epoch {epoch}, metrics: {metr}')
        metrs.append(metr)
        mape = metr["mape"]

        epoch += 1

    return metrs

result = train(KRG, f1, x_train_small, y_train_small, x_validation, y_validation, limits, 1000000, 5, epochs=4)

train_and_test_model(KRG, x_train, y_train, x_validation, y_validation)

model = train_model(KRG, x_train, y_train)

def train_compare(models, f, x_train_initial, y_train_initial, x_validation, y_validation, limits, new_points_grid_n, epoch_new_points_n, epochs=None, mape_threshold=None):
    results = {}
    for model_name_str, model_name in models.items():
```

```
print(f'**********')
        print(f'Model {model_name_str}')
        result = train(model_name, f, x_train_initial, y_train_initial, x_validation, y_validation, limits, new_points_grid_n, epoch_new_points_n, epochs, mape_threshold)
        results[model_name_str] = result
    return results

models = {
    'KRG': KRG,
    'KPLS': KPLS,
    'KPLSK': KPLSK,
}
comparison_result = train_compare(models, f1, x_train_small, y_train_small, x_validation, y_validation, limits, 1000000, 5, epochs=20)

def train_without_variance(model_name, x_train_full, y_train_full, x_validation, y_validation, epoch_new_points_n, n_epochs):
    initial_points_n = 30
    x_train = x_train_full[:initial_points_n]
    y_train = y_train_full[:initial_points_n]
    metrs = []
    for epoch in tqdm(range(n_epochs)):
        model, metr = train_and_test_model(model_name, x_train, y_train, x_validation, y_validation)
```

```
            print(f'Epoch {epoch}, metrics: {metr}')
            metrs.append(metr)

            first_new_point = initial_points_n + epoch * epoch_new_points_n
            last_new_point = initial_points_n + (epoch + 1) * epoch_new_points_n
            x_train = np.vstack((x_train, x_train_full[first_new_point:last_new_point]))
            y_train = np.concatenate((y_train, y_train_full[first_new_point:last_new_point]))
        return metrs

qp_results = train_without_variance(QP, x_train, y_train, x_validation, y_validation, 5, 15)

comparison_result['QP'] = qp_results

result_df = pd.DataFrame()
for model_name, metrics in comparison_result.items():
    metrics_df = pd.DataFrame(metrics)
    mape_df = metrics_df['mape']
    result_df[model_name] = mape_df
result_df['QP'] = result_df['QP'].shift(5)
result_df.index *= 5
print(result_df.head(10))

metrics_dict = defaultdict(pd.DataFrame)
```

```
for model_name, metrics in comparison_result.items():
    metrics_df = pd.DataFrame(metrics)
    if model_name == 'QP':
        nans = pd.DataFrame(np.nan, index=pd.RangeIndex(6),
columns=metrics_df.columns)
        metrics_df = pd.concat((nans, metrics_df)).reset_index(drop=True)
    metrics_df.index *= 5
    for metric_name in metrics[0].keys():
        metrics_dict[metric_name][model_name] = metrics_df[metric_name]

for metric_name, metric_df in metrics_dict.items():
    ax = metric_df.plot(figsize=(15, 7))
    ax.set_title(f'{metric_name.upper()}', fontweight='bold')
    plt.savefig(f'{metric_name.upper()}.png', dpi=300)

models = {
    'KRG': KRG,
    'KPLS': KPLS,
    'KPLSK': KPLSK,
    'QP': QP
}
time_results = {}
for model_name, model_class in models.items():
    model = train_model(model_class, x_train, y_train)
    start_time = time.time()
    model.predict_values(x_validation)
```

```
        end_time = time.time()
        time_results[model_name] = (end_time - start_time) / x_train.size
time_results

models = {
        'KRG': KRG,
        'KPLS': KPLS,
        'KPLSK': KPLSK,
        'QP': QP
}
metric_results = {}
for model_name, model_class in models.items():
        _,    metric_results[model_name]    =    train_and_test_model(model_class,
x_train, y_train, x_validation, y_validation)
metric_results

import json

json.dump(comparison_result, open('data/comparison_result.json', 'w'))

x_train, y_train = load_data("train.csv")
scaler = MinMaxScaler().fit(y_train.reshape(-1, 1))
y_train = scaler.transform(y_train.reshape(-1, 1)).flatten()

x_train2, y_train2 = load_data("train2.csv")
scaler2 = MinMaxScaler().fit(y_train2.reshape(-1, 1))
y_train2 = scaler2.transform(y_train2.reshape(-1, 1)).flatten()
```

```
final_model_1 = train_model(KRG, x_train, y_train)
final_model_2 = train_model(KRG, x_train2, y_train2)

def h1(x):
    return final_model_1.predict_values(np.array([x]))[0, 0]

def h2(x):
    return final_model_2.predict_values(np.array([x]))[0, 0]

def h(x, w1, w2):
    return w1 * h1(x) + w2 * h2(x)

results = []
for w1 in tqdm(np.linspace(0, 1, 101)):
    w2 = 1 - w1
    optimization_result = scipy.optimize.minimize(lambda x: h(x, w1, w2), x0=np.array([(a + b) / 2 for [a, b] in limits]), bounds=limits)
    x = optimization_result['x']
    results.append((scaler.inverse_transform(h1(x).reshape(-1, 1))[0, 0], scaler2.inverse_transform(h2(x).reshape(-1, 1))[0, 0]))
results = np.array(results)

x_results = results[:, 0]
x_variation = (np.max(x_results) - np.min(x_results))/np.average(x_results)

y_results = results[:, 1]
```

```
y_variation = (np.max(y_results) - np.min(y_results))/np.average(y_results)
(x_variation, y_variation)

plt.figure(figsize=(15, 7))

plt.scatter(results[:, 0], results[:, 1])
plt.savefig('results.png', dpi=300)

f12(optimization_result.get('x'))

h(np.array([(a + b) / 2 for [a, b] in limits]))
```

ДОДАТОК Б
Лістинг коду моделювання лопатки турбіни

```
classdef HardModel
    properties
        TemperatureModel
        StressModel
        msh
    end
    methods
        function obj=HardModel()
            obj.TemperatureModel = createpde("thermal","steadystate");
            importGeometry(obj.TemperatureModel,"turbineblade.stl");

            obj.StressModel = createpde("structural","static-solid");
            importGeometry(obj.StressModel,"turbineblade.stl");

            obj.msh = generateMesh(obj.TemperatureModel,"Hmax",0.01);
            obj.TemperatureModel.Mesh = obj.msh;
            obj.StressModel.Mesh = obj.msh;

            kapp = 11.5; % in W/m/K

thermalProperties(obj.TemperatureModel,"ThermalConductivity",kapp);

        end
        function [maxDisplacement,maxStress] = runModel(obj, hAir,tAir, ...
                hGas, tGas, pPress, pSuc)
```

```
E = 227E9; % in Pa
CTE = 12.7E-6; % in 1/K
nu = 0.27;

thermalBC(obj.TemperatureModel,"Face",[15 12 14], ...
        "ConvectionCoefficient",hAir, ...
        "AmbientTemperature",tAir);

thermalBC(obj.TemperatureModel,"Face",[11 10 13 1], ...
        "ConvectionCoefficient",hGas, ...
        "AmbientTemperature",tGas);

thermalBC(obj.TemperatureModel,"Face",[6 9 8 2 7], ...
        "ConvectionCoefficient",15, ...
        "AmbientTemperature",400);

thermalBC(obj.TemperatureModel,"Face",[3 4 5], ...
        "ConvectionCoefficient",1000, ...
        "AmbientTemperature",300);

Rt = solve(obj.TemperatureModel);


structuralProperties(obj.StressModel,"YoungsModulus",E, ...
        "PoissonsRatio",nu, ...
        "CTE",CTE);
```

```
                structuralBoundaryLoad(obj.StressModel,"Face",11, ...
                    "Pressure",pPress);

                structuralBoundaryLoad(obj.StressModel,"Face",10, ...
                    "Pressure",pSuc);
                obj.StressModel.ReferenceTemperature = 300; %in degrees C
                structuralBodyLoad(obj.StressModel,"Temperature",Rt);
                structuralBC(obj.StressModel,"Face",3,"Constraint","fixed");

                Rc = solve(obj.StressModel);

                maxStress = max(Rc.VonMisesStress);

                maxDisplacement = max(Rc.Displacement.Magnitude)*10000;

            end
        end
    end
```