НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___»_____20__р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи і методи штучного інтелекту»
спеціальності 122 «Комп'ютерні науки»

на тему: «Застосування методів інтелектуального аналізу даних для покращення точності розпізнавання об'єктів»

Виконала:
студентка IV курсу, групи КІ-01
Масюк Ольга Сергіївна _____

Керівник:
Доцент, Ph. D.
Гуськова Віра Геннадіївна _____

Консультант з економічного розділу:
д.е.н., професор
Шевчук О.А. _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри ШІ,
Кравець П.В. _____

Рецензент:
старший викладач, PhD

Левенчук Л.Б _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студентка _____

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«31» січня 2024 р.

ЗАВДАННЯ
на дипломну роботу студенту
Масюк Ользі Сергіївні

1. Тема роботи «Застосування методів інтелектуального аналізу даних для покращення точності розпізнавання об'єктів», керівник роботи Гуськова Віра Геннадіївна, Доцент, Ph. D., затверджені наказом по університету від «31» травня 2024р. №2240-С.

2. Термін подання студентом роботи «14» червня 2024 року.

3. Вихідні дані до роботи

4. Зміст роботи

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., доцент, к. е. н.		

7. Дата видачі завдання «05» лютого 2024 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Огляд літератури за темою	25.03.2024	Виконано
2	Підготовка першого розділу	10.04.2024	Виконано
3	Підготовка другого розділу	20.04.2024	Виконано
4	Розробка програмного продукту	15.05.2024	Виконано
5	Підготовка третього розділу	22.05.2024	Виконано
6	Підготовка економічної частини	28.05.2024	Виконано
7	Оформлення дипломної роботи	08.06.2024	Виконано
8	Підготовка презентації доповіді	13.06.2024	Виконано

Студент Ольга МАСЮК

Керівник Віра ГУСЬКОВА

РЕФЕРАТ

Дипломна робота: 103 с., 30 рис., 6 табл., 38 посилань, 1 додаток.

ВИЗНАЧЕННЯ КУТА ПРИБУТТЯ, ЛІНІЙНЕ ПЕРЕДБАЧЕННЯ, АВТОРЕГРЕСІЙНА МОДЕЛЬ, МОДЕЛЮВАННЯ СИГНАЛІВ, НЕЙРОННІ МЕРЕЖІ, ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ

Об’єкт дослідження – створення і застосування методів інтелектуального аналізу даних для покращення кута прибуття радару, дослідження існуючих алгоритмів та їх адаптація до конфігурації ULA.

Предмет дослідження – застосування лінійного передбачення до авторегресивної моделі сигналу для екстраполяції антенної решітки та дослідження і використання нейронних мереж для алгоритмів визначення кута прибуття.

Мета роботи – аналіз та реалізація методів інтелектуального аналізу даних для покращення якості розпізнавання об’єктів та вирішення проблем обчислювальної складності таких алгоритмів.

ABSTRACT

Bachelor’s thesis: 103 p., 30 figures, 6 tables, 38 references, 1 appendix.

DOA ESTIMATION, LINEAR PREDICTION, AUTOREGRESSIVE MODEL, SIGNAL MODELING, NEURAL NETWORKS, INTELLIGENT DATA ANALYSIS

The object of research is the creation and application of data analysis methods to improve the radar angle of arrival, evaluation of existing algorithms and their adaptation to the ULA configuration.

The subject of the research is the application of linear prediction to an autoregressive signal model for antenna array extrapolation, study and use of neural networks for algorithms for determining the angle of arrival.

The aim of the work is to analyze and implement data analysis methods to improve the quality of object recognition and solve the problems of computational complexity of such algorithms.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

DoA – Direction of Arrival, напрямок прибуття

SNR – Signal-to-Noise Ratio, відношення сигнал/шум

FFT – Fast Fourier Transform, швидке перетворення Фур'є

ULA – Uniform Linear Array, рівномірний лінійний масив

DFT – Discrete Fourier Transform, дискретне перетворення Фур'є

PSLR – Peak to SideLobe Ratio, коефіцієнт піку до бічного променю

AR – Autoregressive, авторегресивний

MSE – Mean Squared Error, середньоквадратична помилка

NN – Neural Network, нейронна мережа

DL – Deep Learning, глибоке навчання

CNN – Convolutional neural network, згорткові нейронні мережі

ВСТУП

Радіолокаційні системи є невід’ємною частиною технічних приладів та процесів у різноманітних галузях, таких як: автомобільна індустрія, системи протиповітряної оборони, метеорології, авіації, тощо.

Ідея цієї роботи була натхненною саме автомобільним радаром через ряд обмежень, які вводить ця предметна область, а саме: невеликий розмір радіолокаційної системи, оптимальна ціна виробництва, високоточне розпізнавання об’єктів та робота в різноманітних погодних умовах. Враховуючи перелічене, подібне може бути також застосовано для пристроїв оборонної галузі, як от дронів. Вважаючи на попит розробки подібних пристроїв, було вирішено працювати над вдосконаленням алгоритмів обробки сигналів радіолокаційних систем з використанням методів інтелектуального аналізу даних.

Використання методів моделювання часових рядів та нейронних мереж дозволить покращити розпізнавальні характеристики кута прибуття радару при цьому не підвищуючи витрат на його виробництво та не ускладнюючи систему. Це також надасть змогу скоректувати систему у разі виходу зі строю елементу антенного ряду, що є важливим фактором забезпечення коректної роботи пристрою.

Варто також зазначити, що подібні методи можуть мати ще більш широке використання, що виходить межі радіолокаційних технологій у загальніший випадок обробки інших видів сигналів. Це додає універсальності обраним методам дослідження та розширює область застосування.

Пояснювальна записка до дипломної роботи складається з чотирьох розділів. У першому розділі було проведено огляд предметної області, досліджено використані алгоритми обробки сигналів та визначено постановку задачі.

У другому розділі розглянуто різноманітні підходи інтелектуального аналізу даних, та проведено їх порівняння. До методів описаних у цьому розділі входить лінійне передбачення з моделюванням сигналу за допомогою AR-моделей та методів знаходження коефіцієнтів моделі, а також дослідження структур нейронних мереж.

Третій розділ містить демонстрацію реалізованих методів інтелектуального аналізу та результати експериментів на симульованих та реальних даних радару.

У четвертому розділі проведено економічний аналіз програмного продукту.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Принципи роботи радару

Основна теорія з принципу роботи радару ґрунтується на особливостях розповсюдження радіохвиль, а саме їх відбитті, постійній швидкості та прямолінійності розповсюдження. Опис цих базових принципів дасть зрозуміле та просте уявлення про радіолокацію.

Відбиття електромагнітних хвиль.

Цей процес дозволяє виявити наявність перешкоди на шляху розповсюдження електромагнітної хвилі – якщо на ньому трапляється певна відбиваюча поверхня, електромагнітна хвиля відбивається відіб'ється від неї. Якщо ж радіолокаційний пристрій зможе зареєструвати відбиту хвилю, це сигналізуватиме про можливий об'єкт (перешкоду) у напрямку розповсюдження.

Постійність швидкостей.

Електромагнітні хвилі розповсюджуються в просторі з постійною швидкістю, яка приблизно дорівнює швидкості світла – 300 000 км/с. Ця характеристика дозволяє виміряти відстань до об'єкта відносно радіолокатора враховуючи час затримки випроміненого сигналу та прийнятого (відбитої електромагнітної хвилі).

Прямолінійність траєкторії розповсюдження.

Енергія електромагнітної хвилі розповсюджується по прямій траєкторії, і загалом не піддатна впливу атмосферних чи погодних умов. Це надає змогу спрямовувати сигнал у певному напрямку для визначення азимуту та кута місця (elevation) відносно відбиваючого об'єкту.

1.2 Виміри радару

Параметрами вимірюваного об’єкту для радіолокаційної системи є: відстань, відносна швидкість та кут в азимуті або висоті. Ці метрики повністю визначають ціль у просторі, і їх детальний розрахунок можна побачити в роботі Kronauge, M. Waveform «Design for Continuous Wave Radars» [1].

Так як до розгляду в цій роботі було обрано покращення якості виміру кута, надалі буде розглянуто вимір та обчислення саме цього параметру, проте зазначені методи інтелектуального аналізу можуть також застосовуватися для покращення інших характеристик.

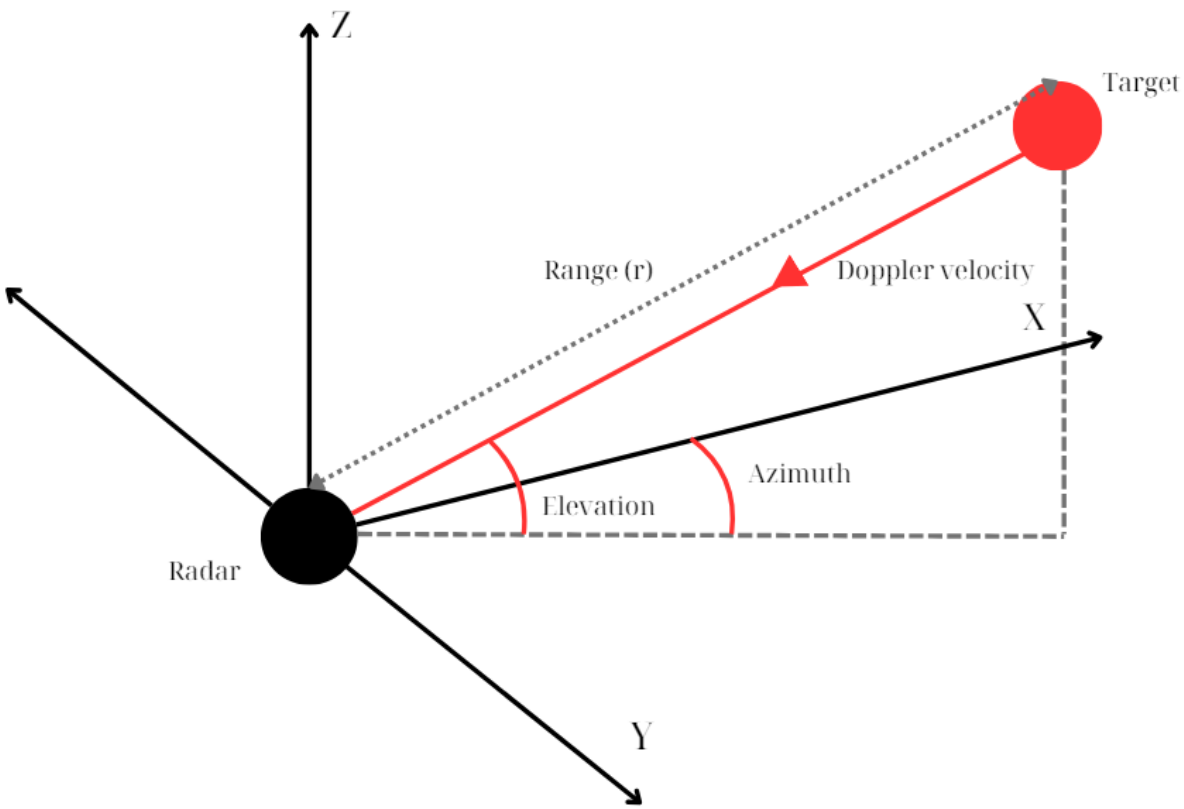


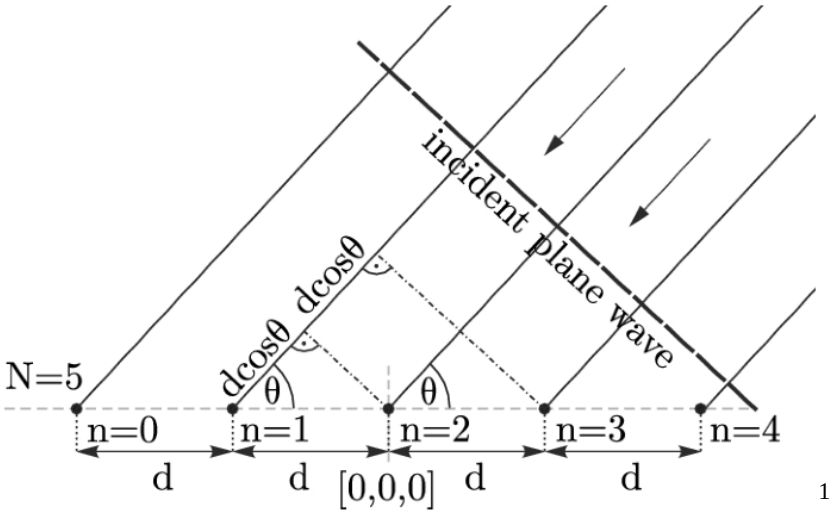
Рисунок 1.1 – Вимір параметрів радіолокаційної системи

1.2.1 Вимір кута

Для виміру кута радіолокаційною системою існує два основних методи. Перший полягає [1] у використанні антени, що має здатність механічно обертатися – таким чином кут визначається напрямом з якого надходить найбільша амплітуда відбитого сигналу. Цей спосіб застосовується у авіаційних радіолокаторах та очевидно не є оптимальним у випадку автомобільного (компактного) радару вважаючи на форму та габарити такої системи.

Другий спосіб [1] використовує масив антен, що за певною геометрією розташовані одна біля одної. У цьому разі кут прибуття визначається з різниці шляху відбитого сигналу при надходженні до кількох просторово розподілених антен. Цей метод також називається променеформування (eng. Beamforming), адже дозволяє не механічно, а обертати промінь.

У цій роботі буде розглянуто останній метод вимірювання кута у випадку лінійного антенного ряду (ULA – Uniform Linear Array).



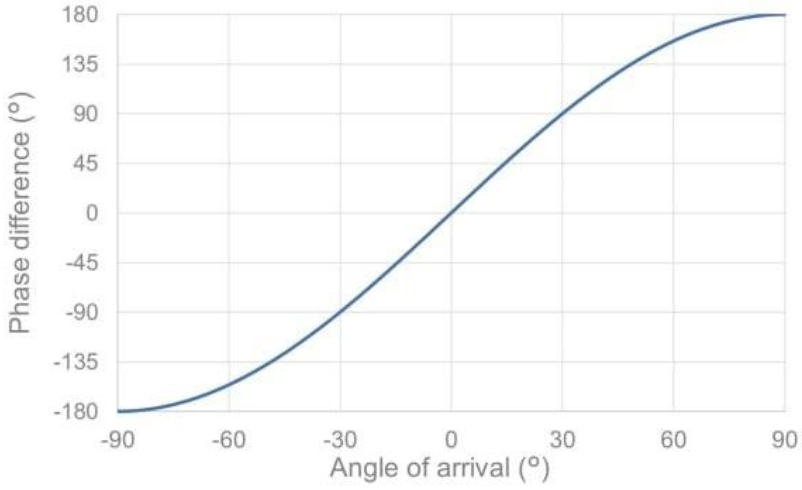
[1]

Рисунок 1.2 – Вимір кута

[1] Джерело зображення: https://www.researchgate.net/publication/362299892/figure/fig1/AS:11431281092181307@1666751426866/Uniform-linear-array-with-an-incident-plane-wave.png

Кут прибуття сигналу можна розрахувати [4], порівнюючи фазовий зсув сигналу, прийнятого кожною з антен. Фазовий зсув прийнятого сигналу залежить від частоти (довжини хвилі) і відстані між антенами, як показано на Рисунку 1.3.

$$\omega = \frac{2\pi \Delta d}{\lambda} = \frac{2\pi\, d \sin\theta}{\lambda}.$$



[2]

Рисунок 1.3 – Різниця фаз двох антен на відстані d = λ/2

1.2.2 Променеформування та ULA

Як вже було зазначено, визначення кута можливе за допомогою масиву антен. Вони можуть бути розташованими у ряд, або планарно. У цій роботі розглядатиметься саме перший варіант, що відповідає ULA радару.

ULA (Uniform Linear Array) є одним з найпоширеніших типів антенних решіток. В ULA всі антени розташовані на однаковій відстані одна від одної та розташовані вздовж прямої лінії. Така конфігурація забезпечує простоту в обробці сигналів та моделюванні [37].

[2] Джерело зображення: https://www.renesas.com/sites/default/files/media/images/phase-difference-receive-antennas.jpg

Основний принцип променеформування у ULA полягає у фазовій корекції прийнятих сигналів від кожної антени. Цей спосіб дозволяє зосередити енергію у визначеному напрямку, тим самим підвищуючи роздільну здатність системи. Чим тонший промінь зможе сформувати система – тим точніше будуть виміри. Також важливо контролювати зменшення побічних променів, адже це допомагає знизити інтерференцію від сторонніх сигналів та покращити визначення основного напрямку сигналу [37].

Існують такі методи променеформування:

- метод DFT (дискретне перетворення Фур'є): дозволяє ефективно обчислювати спектральну характеристику сигналу;
- метод MUSIC (Multiple Signal Classification): використовується для високоточної оцінки напрямків прибуття сигналів;
- метод ESPRIT (Estimation of Signal Parameters via Rotational Invariance Techniques): забезпечує високу точність при менших обчислювальних витратах у порівнянні з MUSIC.

Кількість антен у масиві значно впливає на ефективність променеформування, адже надає можливість спрямування вузького і потужного променя в одному напрямку. Це проілюстровано на Рисунку 1.4 з симуляції виконаної в MATLAB.

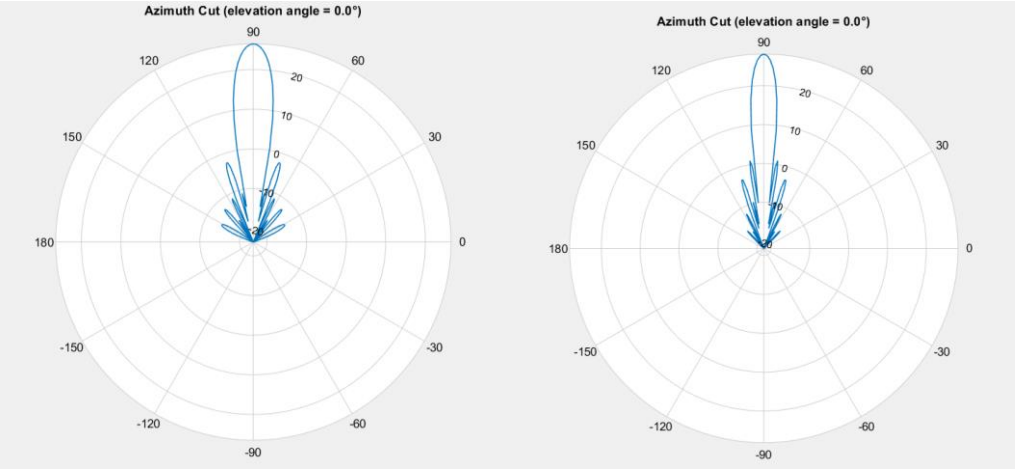


Рисунок 1.4 – Променеформування ULA системи з кількістю антен 8 та 16 відповідно

Дійсно, можна помітити, що більша кількість антен не лише покращує якість основного променя, а й знижує бокові промені, що є оптимальною ситуацією для радіолокаційної системи.

Переконаємося, що це дійсно вплине на здатність розпізнавання об’єктів виконавши нову симуляцію в MATLAB.

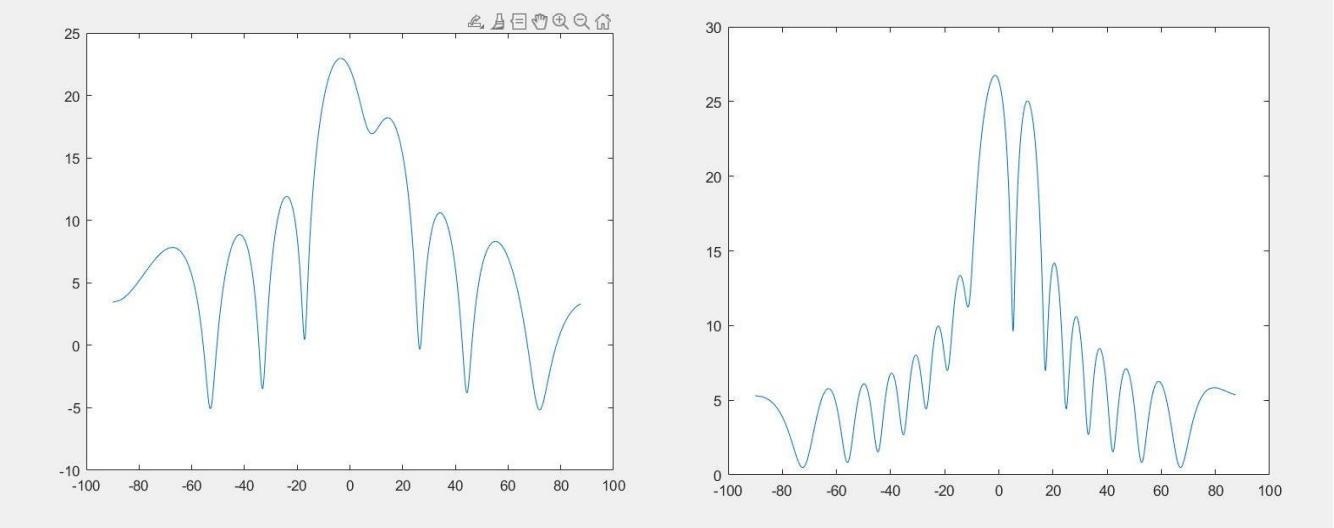


Рисунок 1.5 – Розпізнавання двох об’єктів на куті 3.5 градуси ULA системою з кількістю антен 8 та 16 відповідно

Справді, система з кількістю антен 8 не змогла розпізнати два різних об’єкти що знаходяться близько за кутами прибуття сигналу. А система з 16 антенами чітко розпізнала дві окремі цілі. Варто зауважити, що симуляція виконана без додавання шуму до сигналу. У реальній ситуації роботи радіолокаційної системи шум значно ускладнює коректне розпізнавання об’єкта.

Проте, збільшення кількості антен також підвищує складність системи та витрати на виробництво. Тому використання методів інтелектуального аналізу даних для збільшення «віртуального» ряду антен є оптимальним інструментом розв’язку цієї проблеми.

1.3 Роздільна здатність (Resolution)

Два розсіювачі однакової сили вважаються розв'язаними, якщо вони виробляють два окремо ідентифікованих сигнали на виході системи, на відміну від об'єднання в один недиференційований вихід. [4]

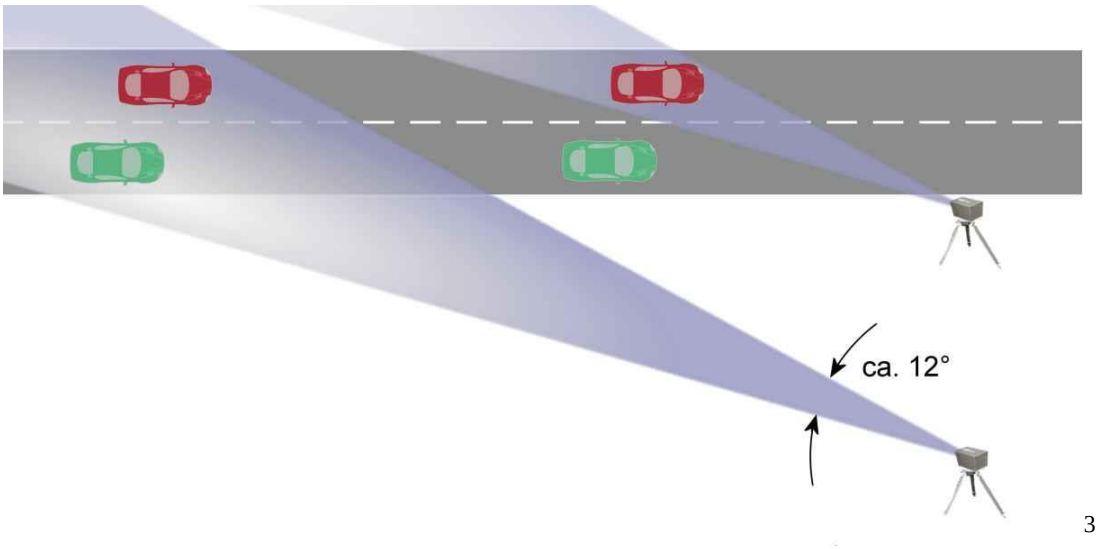
Роздільна здатність може застосовуватися, до швидкості, відстані чи доплерівському зсуві, а також у куті прибуття. Дві або більше цілей можуть бути одночасно розв’язані, наприклад, у відстані та бути нерозв'язаними для кута прибуття.

Покращення роздільної здатності є важливою проблемою для радіолокаційних систем. Особливо це є необхідністю для розробки автономних пристроїв, що базуються на радарі, адже підвищена роздільна здатність означає кращу безпеку та працездатність такої системи.

Як вже було зазначено, саме покращення кутової роздільної здатності становитиме цікавість для цієї роботи.

1.3.1 Кутова роздільна здатність (Angular Resolution)

Визначення кута прибуття є важливим завданням у багатоцільому сценарії. За наявності двох або більше цілей чи перешкод в однаковому діапазоні дальності, виникає необхідність коректно визначити усі з них. Кутова роздільна здатність θ_{res} вказує на мінімальну різницю між кутами, що може бути розв’язаною.



[3]

Рисунок 1.6 – Кутова роздільна здатність дорожнього радару

Для однієї антени його можна розрахувати [4] як:

$$\theta_{\text{res}} > \frac{\lambda}{N\, d \cos\theta},$$

де N – кількість приймальних антен,

θ – кут прибуття.

Кутова роздільна здатність при є максимальною для напрямків, близьких до напрямку наведення (θ = 0°):

$$\theta_{\text{res}} > \frac{\lambda}{N\, d}.$$

[3]Джерело зображення: https://www.radartutorial.eu/01.basics/pic/Radarfalle.print.jpg

Вважаючи, що антени знаходяться на відстані $d = \frac{\lambda}{2}$, кутова роздільна здатність не залежатиме від частоти і визначатиметься лише кількістю приймальних антен.

$$\theta_{\text{res}} > \frac{2}{N}.$$

Отже, відповідно, чим більше кількість антен – тим краще кутова роздільна здатність радіолокаційної системи.

1.4 Вибір середовища розробки та мови програмування

Мова програмування Python [23] є однією з найпопулярніших для роботи з даними та інтелектуального аналізу. Це пояснюється багатим набором бібліотек, таких як Pandas, NumPy, Scikit-learn, TensorFlow, Keras та PyTorch, що надають широкий спектр інструментів для попередньої обробки даних, побудови та оцінки моделей [9]. Простота та зручність синтаксису Python дозволяють швидко розробляти та тестувати моделі. Python також легко інтегрується з іншими мовами програмування, що дозволяє використовувати його в поєднанні з іншими інструментами та технологіями. Також вибір цієї мови пов’язаний з програмним забезпеченням приладу RadarLog для експериментального збору даних та їх обробки.

MATLAB [24], в свою чергу, є потужним інструментом для математичного моделювання, симуляцій та обробки даних. Він забезпечує високу продуктивність при обробці великих обсягів даних та проведенні чисельних симуляцій [10]. Інтерактивне середовище MATLAB дозволяє швидко вносити зміни та аналізувати результати, що сприяє більш ефективному та зрозумілому

користуванню. MATLAB також має вбудовані функції та інструменти для обробки сигналів та додатків що дозволяють працювати в області радіолокаційних систем. Крім того, MATLAB підтримує експорт даних у різні формати, що дозволяє легко інтегрувати результати симуляцій з іншими інструментами, такими як Python [11].

Середовище розробки Visual Studio Code (VS Code) [38] є популярним редактором коду серед розробників завдяки своїй багатоплатформеності. VS Code підтримує велику кількість розширень для різних мов програмування та інструментів, включаючи розширення для Python та MATLAB, що спрощує розробку та відлагодження коду. Інтерфейс VS Code є інтуїтивно зрозумілим та зручним, а вбудований термінал, система контролю версій Git та інші корисні функції роблять роботу з кодом більш ефективною [11].

Отже, використання Python, MATLAB та Visual Studio Code є обґрунтованим вибором для виконання дипломної роботи. Ці інструменти забезпечують необхідні функціональні можливості та зручність для ефективної реалізації поставлених задач, що сприятиме досягненню високих результатів у дослідженні та розробці.

1.5 Постановка задачі

Враховуючи виконане дослідження предметної області та визначену проблему, поставлену за мету вирішення у цій дипломній роботі, конкретизуємо задачі для ефективного прогнозування сигналу.

Постановку задачі прогнозування можна сформулювати та визначити таким чином.

1. Виконати симуляцію надходження сигналу, моделюючи певний сценарій для отримання даних для обробки.
2. Виконати візуалізацію даних після первинної обробки.

3. Дослідити методи інтелектуального аналізу даних для моделювання та прогнозування даних.
4. Розглянути існуючий інструментарій для роботи з даними та здійснення інтелектуального аналізу даних.
5. Обґрунтувати вибір середовища розробки та мови програмування для написання програмного забезпечення та супроводжуючих маніпуляцій з даними.
6. Розробити математичні моделі сигналу.
7. Застосувати методи інтелектуального аналізу до отриманої моделі.
8. Проаналізувати отримані результати моделювання та прогнозування даних.

Висновки до розділу 1

У першому розділі було проведено дослідження предметної області, а саме теоретичних основ та алгоритмів обробки сигналів, що були застосовані у роботі.

Було визначено постановку завдання покращення якості розпізнавання кута, в основі якого лежить використання лінійного передбачення за допомогою моделювання сигналу на «віртуальні» елементи антенного ряду. Відповідні методи розглянуто у Розділі 2.

Також було обґрунтовано доцільність використання конкретного інструментарію для його реалізації, а саме мов програмування Python, Matlab та середовища Visual Studio Code.

РОЗДІЛ 2. ОГЛЯД ОБРАНИХ МЕТОДІВ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДАНИХ

Отже, постає задача змоделювати сигнал що надійшов на лінійно розташовані антени з метою виявлення лінійного зв'язку між ними та передбачення сигналу поза межами фізичного ряду сенсорів.

Для такого моделювання було розглянуто декілька методів та порівняно їх результати. Вони визначатимуть лінійну залежність між прийнятими сигналами, а надалі екстрапольовані сигнали генеруватимуться за межами областей, що покриваються фактичною антеною решітки.

2.1 Лінійне прогнозування

Маючи дискретний набір вихідних значень, ми можемо використати коефіцієнти для наближення вихідних значень для того, що називається прямим або зворотнім лінійним передбаченням [15].

$$\mathrm{N}(x_n)_{n\in[\,[0,N]\,]}\mathrm{k}(a_n)_{n\in[\,[1,k]\,]}y_n = -\sum_{i=1}^{k} a_i x_{n-i} z_n = -\sum_{i=1}^{k} a_i x_{n+i}.$$

Прямо передбачене значення є лінійною зваженою комбінацією попередніх відомих значень, а зворотнє – лінійно-зваженою комбінацією наступних відомих значень.

Стандартний спосіб вибору коефіцієнтів полягає в тому, щоб мінімізувати суму квадратів похибки між вихідним і наближеним значеннями. Наприклад, для прямого лінійного передбачення:

$$F_k = \sum_{n=k}^{N} (x_n - y_n)^2 = \sum_{n=k}^{N} \left(x_n - \left(-\sum_{i=1}^{k} a_i x_{n-i}\right)\right)^2 \quad (2.1)$$

Для зворотного лінійного передбачення:

$$F_k = \sum_{n=k}^{N} (x_n - y_n)^2 = \sum_{n=k}^{N} \left(x_n - \left(-\sum_{i=1}^{k} a_i x_{n-i}\right)\right)^2. \quad (2.2)$$

Проблема, пов'язана з розв'язанням шляхом мінімізації полягає в тому, що метод використовує коефіцієнти коваріації або автокореляції, які погано підходять для чисельних обчислень, а також тому, що моделі не завжди є стабільними моделями [3]. Це означає, що повернуті алгоритмом не є практично корисними і погано апроксимують вихідні значення. Тому бажаним є більш надійне та стабільне рішення. Для цього буде розглянуто методи розв’язання рівнянь Юла-Уолкера, рекурсію Левінсона-Дурбіна та метод Бурга

2.2 Регресійний аналіз

Регресійний аналіз – це статистичний метод для виявлення та кількісної оцінки зв'язку між залежною змінною та однією або кількома незалежними змінними [12]. Регресійна модель представляє собою функцію незалежної змінної та параметрів з додаванням випадкових змінних. Ці параметри моделі коригуються для найкращого відповідності вихідних даних. Якість такої апроксимації (цільова функція) зазвичай визначається середньоквадратичною помилкою: сумою квадратів різниці між значеннями моделі та залежною змінною для всіх значень незалежної змінної як аргументами.

Регресійний аналіз використовується для прогнозування, аналізу часових

рядів, тестування гіпотез та виявлення прихованих зв'язків у даних.

2.3 Авторегресійна модель

Авторегресійна модель (англ. Autoregressive model, AR) [14] – модель часових рядів, в якій значення часового ряду у відповідний момент часу лінійно залежить від його попередніх значень. За своєю суттю цей підхід до моделювання передбачає, що інформаційний вміст найновіших точок даних робить найбільший внесок у значення даних x(n). Порядок авторегресійної моделі відповідає за кількість попередніх значень, що використані для визначення відповідного значення. Завдяки цій залежності для встановлення зв'язку між автокореляційною послідовністю процесу і модельними коефіцієнтами $a_1, \dots, a_i$ можуть бути використані так звані рівняння Юла-Уокера [13].

Зважаючи на те, що часткова автокореляція процесу AR(p) дорівнює нулю на часових відліках, більших за p, тож обрати максимальним часовим відліком p потрібно той, після якого всі часткові автокореляції дорівнюють нулю.

Авторегресійний процес порядку p визначається так:

$$x_t = c + \sum_{i=1}^{p} a_i x_{t-1} + \varepsilon_t,$$

де x_t – це значення часового ряду у період t,

c – константа, що може бути опущена,

a_i – коефіцієнти авторегресії,

ε_t – шум, що передбачається нормально розподіленим з нульовим середнім та сталою дисперсією.

2.3.1 Знаходження коефіцієнтів розв'язанням рівнянням Юла-Уолкера

Рівняння авторегресії базується на параметрах a_i – коефіцієнтах авторегресії. Існує пряма відповідність між цими коефіцієнтами та коваріаційною функцією процесу, причому цю відповідність можна обернути для визначення параметрів з автокореляційної функції (яка, в свою чергу, виводиться з коваріацій). Для цього використовуються рівняння Юла-Уокера [19]:

$$\gamma_m = \sum_{k=1}^{P} a_k \gamma_{m-k} + \sigma_\varepsilon^2 \delta_{m,0},$$

де γ – автоковаріаційна функція x_t,

σ_ε – середньоквадратичне відхилення процесу вхідного шуму, а

$\delta_{m,0}$ – дельта-функція Кронекера.

Ці рівняння зачасту відтворюють у матричній формі:

$$\begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \vdots \\ \gamma_p \end{bmatrix} = \begin{bmatrix} \gamma_0 & \gamma_{-1} & \gamma_{-2} & \cdots \\ \gamma_1 & \gamma_0 & \gamma_{-1} & \cdots \\ \gamma_2 & \gamma_1 & \gamma_0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ \gamma_{p-1} & \gamma_{p-2} & \gamma_{p-3} & \cdots \end{bmatrix} \begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \vdots \\ \varphi_p \end{bmatrix}.$$

Їх надалі необхідно розв'язати для усіх значень коефіцієнтів.

Наведені вище рівняння пропонують кілька способів оцінки параметрів моделі AR(p), замінюючи теоретичні коваріації на їхні оцінені значення. Деякі з цих варіантів можна описати наступним чином.

1. Оцінка автоковаріацій або автокореляцій.
2. Формулювання задачі як задачі найменших квадратів.
3. Оцінювання максимальної правдоподібності.

2.2.2 Метод Бурга

У 1975 році при використанні рекурсії Левінсона-Дурбіна [19] було необхідно знайти рішення з аналогічними обчислювальними вимогами, але без нестабільності. Саме це зробив Бург [20], застосувавши рекурсію Левінсона-Дурбіна з іншим обмеженням.

У вихідній рекурсії Левінсона-Дурбіна коефіцієнти$(a_n)_{n\in[1,k]}$ зберігаються у векторі $A_k = \begin{bmatrix} 1 \\ a_1 \\ a_2 \\ \vdots \\ a_k \end{bmatrix}$ та векторі перевернутого порядку $V_k = \begin{bmatrix} 0 \\ a_k \\ \vdots \\ a_2 \\ a_1 \\ 1 \end{bmatrix}$.

Отже, формула рекурсії виглядає наступним чином:

$$A_{k+1} = A_k + \mu N_k \ . \tag{2.3}$$

З обчисленим μ таким чином, щоб враховувати початкові умови задачі, як детально описано в [19]. Ідея Бурга полягала в тому, щоб просто змінити спосіб обчислень μ, щоб він відповідав не початковим умовам задачі, а замість цього мінімізував загальну суму $F_k + B_k$ введених в (2.1) і (2.2).

Для цього виведемо алгоритм Бурга з вже відомих формул лінійного передбачення.

Переробивши рівняння лінійного передбачення (2.1) шляхом присвоєння a_0 значення 1:

$$F_k = \sum_{n=k}^{N} \left(a_0 x_n + \sum_{i=1}^{k} a_i x_{n-1}\right)^2 = \sum_{n=k}^{N} \left(\sum_{i=0}^{k} a_i x_{n-1}\right)^2 = \\ = \sum_{n=k}^{N} (f_k(n))^2, \quad (2.4)$$

де

$$f_k(n) = \sum_{i=0}^{k} a_i x_{n-i}. \quad (2.5)$$

Аналогічно, з рівняння (2.5) маємо:

$$B_k = \sum_{n=0}^{N-k} \left(a_0 x_n + \sum_{i=1}^{k} a_i x_{n+i}\right)^2 = \sum_{n=0}^{N-k} \left(\sum_{i=0}^{k} a_i x_{n-i}\right)^2 = \sum_{n=0}^{N-k} \left(b_k(n)\right)^2, \quad (2.6)$$

де

$$b_k(n) = \sum_{i=0}^{k} a_i x_{n+i} \quad (2.7)$$

Отже, записавши A_{k+1} як вектор коефіцієнтів $(a'_n)_{n\in[\,[1,k+]\,]}$ та використовуючи рівняння (2.3) і той факт, що V_k є оберненим $\mathrm{A_k}$, та $a_{k+1} = 0$, отримаємо:

. $$a'_n = a_n + \mu a_{k+1-n} \quad (2.8)$$

Припускаючи, що A_k знайдено, для того, щоб знайти μ, потрібно використовувати (2.4) і (2.6), і мінімізувати:

. $F_{k+1} + B_{k+1} = \sum_{n=k+1}^{N} (f_{k+1}(n))^2 + \sum_{n=0}^{N-k-1} (b_{k+1}(n))^2$ (2.9)

З (2.5) і (2.8) маємо:

$$\begin{aligned} f_{k+1}(n) &= \sum_{i=1}^{k+1} a_i' x_{n-1} \\ &= \sum_{i=0}^{k+1} a_i x_{n-i} + \mu \sum_{i=0}^{k+1} a_{k+1-i} x_{n-i} \\ &= f_k(n) + \mu \sum_{j=0}^{k+1} a_j x_{n-k-1+j}. \end{aligned}$$

Отже,

$$f_{k+1}(n) = f_k(n) + \mu b_k(n-k-1). \quad (2.10)$$

З (2.7) і (2.8):

$$\begin{aligned} b_{k+1}(n) &= \sum_{i=0}^{k+1} a_i' x_{n+i} \\ &= \sum_{i=0}^{k+1} a_i x_{n+i} + \mu \sum_{i=0}^{k+1} a_{k+1-i} x_{n+1}. \end{aligned} \quad (2.11)$$

Отже, $b_{k+1}(n) = b_k(n) + \mu f_k(n+k+1)$.

Додаючи (2.9) та (2.10):

$$F_{k+1} + B_{k+1} = \sum_{n=k+1}^{N} \left(f_k(n) + \mu b_k(n-k-1)\right)^2 +$$

$$+ \sum_{n=0}^{N-k-1} (b_k(n) + \mu f_k(n+k+1))^2. \quad (2.12)$$

Це можна мінімізувати, знайшовши, коли похідна змінної μ дорівнює нулю:

$$\frac{\partial(F_{k+1}+B_{k+1})}{\partial\mu}=0. \tag{2.13}$$

З (2.12) та (2.13) отримаємо:

$$0=\frac{\partial\left(\sum_{n=k+1}^{N}\left(f_k(n)+\mu b_k(n-k-1)\right)^2+\sum_{n=0}^{N-k-1}\left(b_k(n)+\mu f_k(n+k+1)\right)^2\right)}{\partial\mu}$$

$$=2\sum_{n=k+1}^{N} b_k(n-k-1)\left(f_k(n)+\mu b_k(n-k-1)\right)$$

$$+2\sum_{n=0}^{N-k-1} f_k(n+k+1)\left(b_k(n)+\mu f_k(n+k+1)\right).$$

Тому

$$0=\sum_{n=k+1}^{N} b_k(n-k-1)(f_k(n)+\mu b_k(n-k-1))$$

$$+\sum_{n=0}^{N-k-1} f_k(n+k+1)(b_k(n)+\mu f_k(n+k+1)).$$

Таким чином:

$$0=\sum_{n=k+1}^{N} f_k(n)b_k(n-k-1)+\mu\sum_{n=k+1}^{N} b_k(n-k-1)^2$$

$$+\sum_{n=0}^{N-k-1} f_k(n+k+1)b_k(n)+\mu\sum_{n=0}^{N-k-1} f_k(n+k+1)^2.$$

Звідки можемо дістати μ:

$$\mu = -\frac{\sum_{n=k+1}^{N} f_k(n)b_k(n-k-1)+\sum_{n=0}^{N-k-1} f_k(n+k+1)b_k(n)}{\sum_{n=0}^{N-k-1} f_k(n+k+1)^2+\sum_{n=k+1}^{N} b_k(n-k-1)^2}.$$

Спрощення шляхом коригування індексів і меж призводить до

$$\mu = \frac{-2\sum_{n=0}^{N-k-1} f_k(n+k+1)b_k(n)}{\sum_{n=k+1}^{N} f_k(n)^2 + \sum_{n=0}^{N-k-1} b_k(n)^2}.$$

На цьому етапі достатньо реалізувати першу версію алгоритму, дотримуючись наведених нижче кроків.

i. Вибір m, кількість бажаних коефіцієнтів

ii. Ініціалізація $A_0 = [1]$

iii. Використовуючи (2.5) і (2.7), ініціалізація всіх $f_0(n) = b_0(n) = x_n$

iv. Для k від 0 до $m - 1$:

- обчислення μ за допомогою (2.14);
- оновлення A_{k+1} за допомогою (2.8);
- оновлення $(f_{k+1}(n))_{n\notin[\,[k+,N]\,]}$ за допомогою (2.5);
- оновлення $(b_{k+1}(n))_{n\in[\,[0,N-k-]\,]]}$ за допомогою (2.7).

Цей алгоритм можна покращити [17] використовуючи математичні спрощення. Обчислення μ може бути зроблено простіше і привести до поліпшеної версії, яка в літературі називається рекурсією Бурга.

Перш за все, варто зазначити, що знаменник D_k :

$$D_k = F_k - f_k(k)^2 + B_k - b_k(N-k)^2. \qquad (2.15)$$

Отже, необхідно знайти рекурсивну формулу, щоб можна було обчислити

$$D_{k+1} = F_{k+1} - f_{k+1}(k+1)^2 + B_{k+1} - b_{k+1}(N-k-1)^2.$$

$$F_{k+1} + B_{k+1} = \sum_{n=k+1}^{N} (f_k(n)^2 + 2\mu b_k(n-k-1) f_k(n) + \mu^2 b_k(n-k-1)^2)$$
$$+ \sum_{n=0}^{N-k-1} (b_k(n)^2 + 2\mu b_k(n) f_k(n+k+1) + \mu^2 f_k(n+k+1)^2)$$
$$= \sum_{n=k+1}^{N} f_k(n)^2 + \sum_{n=0}^{N-k-1} b_k(n)^2 \quad .$$
$$+2\mu \left(\sum_{n=k+1}^{N} b_k(n-k-1) f_k(n) + \sum_{n=0}^{N-k-1} b_k(n) f_k(n+k+1) \right)$$
$$+\mu^2 \left(\sum_{n=k+1}^{N} b_k(n-k-1)^2 + \sum_{n=0}^{N-k-1} f_k(n+k+1)^2 \right)$$

Використовуючи (2.4) і (2.6), а також оновлюючи індекси, отримаємо

$$F_{k+1} + B_{k+1} = F_k - f_k(k)^2 + B_k - b_k(N-k)^2$$
$$+2\mu \left(\sum_{j=0}^{N-k-1} b_k(j) f_k(j+k+1) + \sum_{n=0}^{N-k-1} b_k(n) f_k(n+k+1) \right).$$
$$+\mu^2 \left(\sum_{n=0}^{N-k-1} b_k(n)^2 + \sum_{n=k+1}^{N} f_k(n)^2 \right)$$

Тому:

$$F_{k+1} + B_{k+1} = F_k - f_k(k)^2 + B_k - b_k(N-k)^2$$
$$+4\mu \left(\sum_{n=0}^{N-k-1} b_k(n) f_k(n+k+1) \right) \quad .$$
$$+\mu^2 (F_k - f_k(k)^2 + B_k - b_k(N-k)^2)$$

Розкладання на множники та використання (2.14) дає:

$$\begin{aligned} F_{k+1} + B_{k+1} &= (1+\mu^2)(F_k - f_k(k)^2 + B_k - b_k(N-k)^2) \\ &\quad +4\mu\left(-\frac{\mu}{2}\left(\sum_{n=k+1}^{N} f_k(n)^2 + \sum_{n=0}^{N-k-1} b_k(n)^2\right)\right) \\ &= (1+\mu^2)(F_k - f_k(k)^2 + B_k - b_k(N-k)^2) \\ &\quad -2\mu^2(F_k - f_k(k)^2 + B_k - b_k(N-k)^2) \\ &= (1-\mu^2)(F_k - f_k(k)^2 + B_k - b_k(N-k)^2) \\ &= (1-\mu^2)D_k \end{aligned} .$$

Отже, маємо:

$$D_{k+1} = (1-\mu^2)D_k - f_{k+1}(k+1)^2 - b_{k+1}(N-k-1)^2. \qquad (2.16)$$

Остаточна версія покращеного алгоритма Бурга:

i. Вибір кількості бажаних коефіцієнтів m

ii. Ініціалізація $A_0 = [1]$

iii. Використовуючи (5) і (7), ініціалізація всіх $f_0(n) = b_0(n) = x_n$

iv. Використовуючи (4) і (6), обчислення F_0 і B_0

v. Використовуючи (15) обчислення D_0

vi. Для k від 0 до $m-1$:

- обчислення μ за допомогою $\mu = \frac{-2\sum_{n=0}^{N-k-1} f_k(n+k+1)b_k(n)}{D_k}$;
- оновлення A_{k+1} за допомогою (8);
- оновлення $(f_{k+1}(n))_{n\notin[\,[k+,N]\,]}$ за допомогою (10);
- оновлення $(b_{k+1}(n))_{n\notin[\,[0,N-k-1]\,]}$ за допомогою (11);
- оновлення D_k за допомогою (16).

2.4 Глибоке машинне навчання

Глибоке навчання відноситься до класу методів машинного навчання, що використовують багатошарові ієрархічні архітектури для неконтрольованого навчання ознак і класифікації шаблонів [29]. Це напрямок, який об'єднує дослідження нейронних мереж, графічного моделювання, оптимізації, розпізнавання образів та обробки сигналів [30].

Дві ключові причини популярності глибокого навчання сьогодні — це значне зниження вартості обчислювального обладнання та різке зростання обчислювальних можливостей мікросхем, таких як графічні процесори. Починаючи з 2006 року, дослідники досягли значних успіхів у застосуванні глибокого навчання до різних задач, таких як комп'ютерний зір, фонетичне розпізнавання, голосовий пошук, спонтанне розпізнавання мови, кодування ознак мови та зображень, класифікація семантичних висловлювань, розпізнавання рукописного тексту, обробка звуку, пошук інформації та робототехніка [16].

Генеративне навчання

Генеративне навчання та дискримінаційне навчання є двома поширеними підходами в машинному навчанні, які використовуються в різних сферах, включаючи обробку сигналів. Генеративне навчання полягає в побудові моделі, яка може генерувати нові дані, схожі на навчальні приклади. Цей підхід має такі характеристики:

- використання генеративної моделі;
- цільова функція навчання базується на спільній ймовірності, визначеній генеративною моделлю.

Генеративні моделі намагаються навчитися повному розподілу даних, включаючи залежність між різними змінними. Це дозволяє їм генерувати нові зразки даних, що схожі на вихідні навчальні дані. У обробці сигналів такі моделі можуть бути корисними для відновлення або синтезу сигналів, які мають подібні статистичні характеристики до оригінальних сигналів [16].

Дискримінаційне навчання

Дискримінаційне навчання, з іншого боку, фокусується на розрізненні між різними класами даних. Воно використовує моделі, які прогнозують ймовірність належності вхідного сигналу до певного класу. Характеристиками є:

- використання дискримінаційної моделі;
- використання дискримінаційної цільової функції навчання, такої як крос-ентропія, яка мінімізує різницю між передбаченими та фактичними класами.

Дискримінаційні моделі намагаються знайти кордони між класами даних, не турбуючись про моделювання всього розподілу даних. Це дозволяє їм бути більш ефективними у задачах класифікації сигналів. Популярні приклади дискримінаційних моделей включають логістичну регресію, підтримуючі векторні машини (SVM) та нейронні мережі [16].

Гібридні підходи

У обробці сигналів також стає популярним застосування дискримінаційного навчання до генеративних моделей. У цьому підході генеративна модель створює ймовірнісні уявлення даних, тоді як дискримінаційне навчання оптимізує цільову функцію на основі цих уявлень. Це поєднання дозволяє отримати переваги обох підходів, підвищуючи точність і надійність моделей обробки сигналів.

Дискримінаційні моделі безпосередньо використовують умовне відношення міток заданих вхідних векторів. Одна з основних таких моделей називається баєсовими класифікаторами мінімального ризику (BMR):

$$f(\mathbf{x};\lambda) = -\arg\min_{y'} \sum_{y} \Delta(y', y) p(y \mid \mathbf{x};\lambda).$$

2.5 Алгоритми навчання

Кожна модель має свої цілі і повинна навчатися по-різному. У цьому розділі описано кілька методів навчання, які поділяються на три основні категорії:

Кероване навчання

Кероване навчання використовує навчальну вибірку, що складається з пар вхідних даних і бажаних результатів [29]. Метою цього алгоритму є вивчення відображення між входом і виходом. Надалі це відображення використовується для оцінки результатів на нових вхідних даних. Якщо вихід приймає скінченний набір дискретних значень, що відповідають міткам класів вхідних даних, то йдеться про класифікацію. Якщо вихідні значення неперервні, то нейронна мережа виконує регресію [24].

Навчання без нагляду

Навчання без нагляду [32] використовує навчальну множину, що складається з вхідних даних без жодних призначених бажаних результатів. Мета цього алгоритму полягає в тому, щоб згрупувати вхідні точки, які є близькими одна до одної, присвоюючи мітку кожному входу. Типові завдання для неконтрольованого навчання включають кластеризацію і зменшення розмірності.

Навчання з підкріпленням

Навчання з підкріпленням [32] не має фіксованого набору навчальних даних і отримує зворотний зв'язок від середовища лише після вибору виходу для певного входу або спостереження. Зворотний зв'язок показує, наскільки вихід, відомий як дія в навчанні з підкріпленням, відповідає цілям агента.

2.6 Техніки регуляризації

Dropout

Dropout [31] – це техніка регуляризації, яка запобігає перенавчанню та забезпечує спосіб ефективного поєднання експоненціально великої кількості різних архітектур нейронних мереж. Термін "dropout" означає випадкове відключення певної кількості нейронів разом з усіма їх вхідними та вихідними з'єднаннями. Під час тренування створюється експоненціально велика кількість "розріджених" нейронних мереж, які мають спільні ваги. Під час тестування спільні ваги всіх "розріджених" нейронних мереж усереднюються і масштабуються для використання в "нерозрідженій" нейронній мережі без dropout.

Batch Normalization

Нормалізація пакета [31] – це техніка, що зменшує внутрішній ковариатний зсув. Ковариатний зсув виникає, коли розподіл вхідних даних кожного шару постійно змінюється, що призводить до зміни параметрів у попередньому шарі. Це досягається за допомогою кроку нормалізації, який нормалізує кожен елемент шару до нульового середнього та одиничної дисперсії, а потім виконує масштабування γ і зміщення β. Ці кроки значно підвищують швидкість навчання.

$$\hat{x}_i = \gamma \cdot \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta.$$

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i.$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2.$$

Early Stopping

Раннє зупинення [31] – це техніка регуляризації, яка зупиняє тренування, коли продуктивність моделі перестає покращуватися на валідаційному наборі даних. Ця техніка регуляризації запобігає перенавчанню та зберігає найкращі параметри моделі.

Weight Decay

Спад ваги [31] – це техніка регуляризації, яка штрафує великі ваги, додаючи регуляризаційний член до функції втрат. Як регуляризаційний член використовують L2-норму, яка штрафує великі ваги. Це запобігає перенавчанню та утримує ваги малими.

$$L(\mathbf{w}) = L_0(\mathbf{w}) + \frac{1}{2}\lambda \sum_{i=1} \|w_i\|^2.$$

2.7 Нейронні мережі

Штучні нейронні мережі (ШНМ) – це математичні моделі, а також їх програмні або апаратні реалізації, побудовані за принципами подання й обробки інформації у біологічних нейронних мережах – мережах нервових клітин живого організму [8].

Штучний нейрон (формальний нейрон, нейроподібний елемент) – це примітивний обчислювальний пристрій (або його модель), що має кілька входів і один вихід, і є основним обчислювальним елементом ШНМ. Математична модель функціонування штучного нейрона описується співвідношенням:

$$y = \psi(\varphi(w, x)),$$

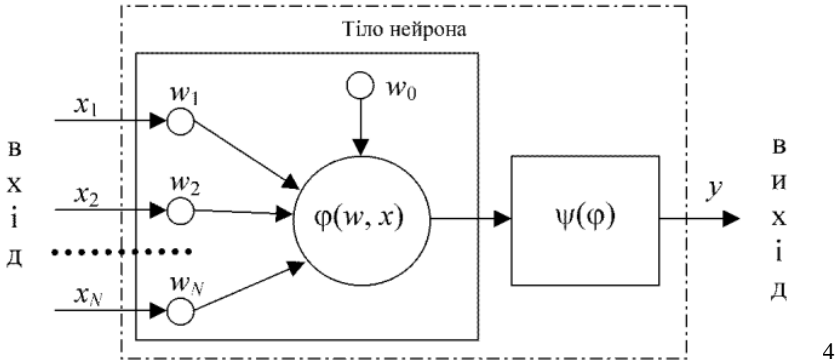
де x – вектор вхідних аргументів (сигналів),

y – значення на виході нейрона,

ψ – функція активації,

φ – дискримінантна функція,

w={wj}– вектор, що містить значення вагових коефіцієнтів wj і значення зсуву (порогове значення) $w0$ [8].



[4]

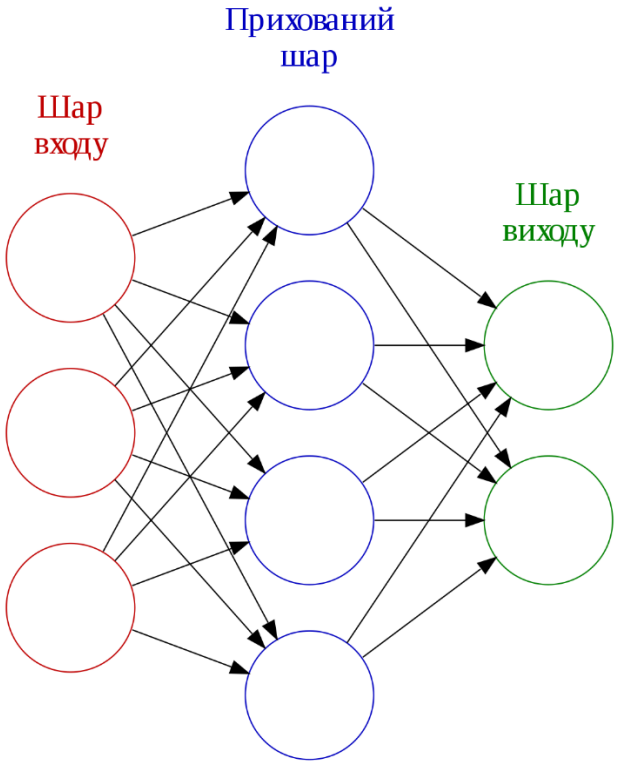
Рисунок 2.1 – Схема штучного нейрона

Нейронні мережі складаються з великої кількості взаємопов'язаних штучних нейронів, які працюють разом для вирішення конкретних завдань.

Основними складовими ШНМ є:

- вхідний шар: складається з нейронів, які отримують дані ззовні.
- приховані шари: один або більше шарів нейронів, що обробляють вхідні дані і передають їх далі по мережі;
- вихідний шар: нейрони цього шару надають кінцевий результат обробки.

[4] Джерело зображення: https://lpnu.ua/sites/default/files/2023/radaphd/23528/2-phdthesispukachpp1705.pdf



[5]

Рисунок 2.2 – Складові нейронної мережі

2.7.1 Згорткові нейронні мережі

Згорткові нейронні мережі (CNN) – це спеціалізована архітектура нейронних мереж, яка добре підходить для обробки даних з топологічною структурою, таких як зображення. CNN знайшли широке застосування в завданнях комп'ютерного зору, включаючи розпізнавання об'єктів, класифікацію зображень, сегментацію та інші [7].

Особливістю CNN є використання спеціальної операції - згортки, лінійної

[5] Джерело зображення: https://upload.wikimedia.org/wikipedia/commons/thumb/6/61/Colored_neural_network_uk.svg/345px-Colored_neural_network_uk.svg.png?20180112235712

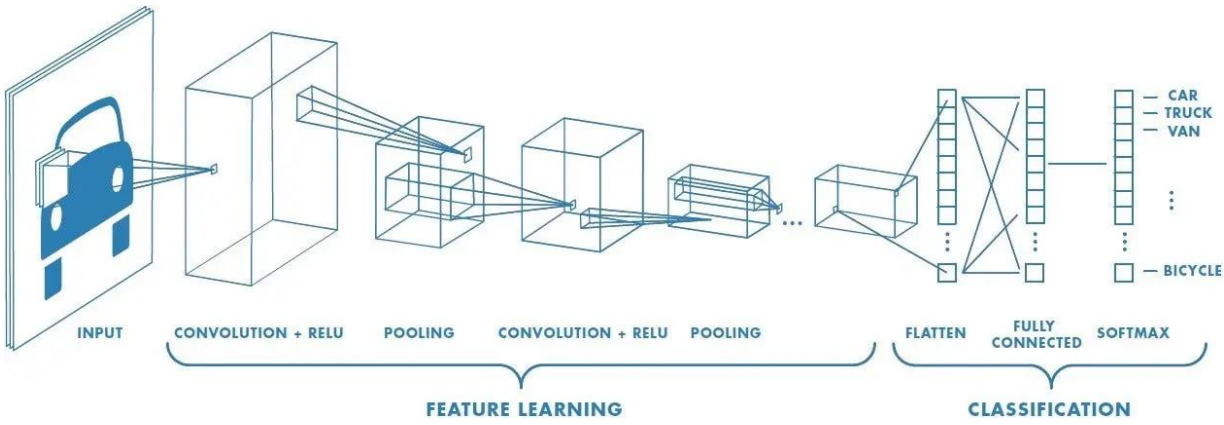
операції на матриці, яка полягає у сумуванні добутків ваг ядра та вхідної матриці для кожної клітинки вихідної. Математично це описується формулою:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n),$$

де * – операція згортки,

I – вхідна матриця,

S – вихідна матриця, K – ядро згортки.



[6]

Рисунок 2.3 – Типова архітектура згорткової нейромережі

2.8 Метрики якості

Визначимо метрики, які використовуватимуться для оцінювання ефективності створених моделей. Ці конкретні метрики зосереджені на вимірюванні точності класифікації. Оскільки завдання класифікації є

[6] Джерело зображення: https://www.researchgate.net/profile/Zuherman-Rustam-2/publication/342072604/figure/fig1/AS:904152184025088@1592578087236/Illustration-of-Convolutional-Neural-Network.jpg

багатокласовим, метрики будуть описані у відповідному багатокласовому форматі.

Перш ніж перейти до опису метрик, введемо поняття істинно позитивних, істинно негативних, хибно позитивних та хибно негативних значень у контексті багатокласового завдання [6].

1. Істинно позитивне (True Positive, TP): для певного класу «i» екземпляр вважається істинно позитивним, якщо він належить до класу «i» і правильно визначений моделлю як клас «i».
2. Істинно негативне (True Negative, TN: для певного класу «i» екземпляр вважається істинно негативним, якщо він не належить до класу «i» і правильно не віднесений моделлю до класу «i».
3. Хибно позитивне (False Positive, FP): для певного класу «i» екземпляр вважається хибно позитивним, якщо він не належить до класу «i», але модель помилково віднесла його до класу «i».
4. Хибно негативне (False Negative, FN): для певного класу «i» екземпляр вважається хибно негативним, якщо він належить до класу «i», але модель помилково не віднесла його до класу «i».

Для обчислення цих значень розглядається кожен клас окремо, трактуючи його як «позитивний», в той час як інші класи розглядаються як «негативні». Такий підхід називається «Один проти всіх» (One-vs-All) [6].

Accuracy

У багатокласовій класифікації точність визначається як відношення кількості правильних прогнозів моделі до загальної кількості прогнозів [9]. Математично це виражається так:

$$accuracy = \frac{number\ of\ true\ predictions}{total\ number\ of\ predictions} = \frac{TP+TN}{TP+TN+FP+FN}.$$

Показник влучності варіюється в інтервалі від 0 до 1, де значення 1 відповідає ідеальній точності, а значення 0 свідчить про повну відсутність

правильних прогнозів [9].

Precision

Точність можна розглядати як здатність моделі правильно ідентифікувати тільки правильні екземпляри кожного класу. Іншими словами, це метрика, яка показує, скільки з усіх екземплярів, віднесених до певного класу, насправді належать цьому класу.

У випадку багатокласової класифікації точність зазвичай обчислюється для кожного класу окремо, а потім усереднюється по всіх класах. Існує два підходи до обчислення точності в багатокласових задачах [9].

1. Мікро-усереднення (micro-average)**.** У цьому підході всі класи розглядаються разом. Обчислення агрегується для всіх класів, підсумовуючи загальну кількість істинно позитивних і хибно позитивних результатів у всіх класах. Формула для обчислення точності виглядає так:

$$\text{precision} = \frac{TP}{TP+FP}.$$

2. Макро-усереднення (macro-average). У цьому підході класи розглядаються послідовно і незалежно, після чого усереднюється значення точності по всіх класах:

$$\text{precision} = \text{mean}\left\{\frac{TP_i}{TP_i + FP_i}\right\} \text{ for every } i.$$

Recall

Повнота (recall) – це метрика, що показує здатність моделі правильно ідентифікувати істинні екземпляри певного класу з усіх екземплярів, що належать цьому класу. Математично повноту можна обчислити за формулою:

$$\text{recall} = \frac{TP}{TP + FN}.$$

Як і точність, повноту можна обчислювати двома способами: мікроусередненням і макро-усередненням. Повнота є важливим показником, особливо у сценаріях, де критично важливо виявити істинні випадки, і де помилки мають високу ціну. Ця метрика дозволяє оцінити здатність моделі мінімізувати кількість хибно негативних випадків [9].

F1 Score

У багатокласовій класифікації міра F1 є показником загальної ефективності моделі класифікації. Вона об'єднує точність і повноту в одну оцінку, враховуючи як хибно позитивні, так і хибно негативні прогнози. F1-міра також може обчислюватися в контексті мікро-усереднення та макро-усереднення. Формула для обчислення F1-міри є гармонійним середнім між точністю та повнотою:

$$F1\ score = \frac{2 \cdot precision \cdot recall}{precision + recall}.$$

Як і повнота та точність, значення F1-міри знаходиться в інтервалі від 0 до 1, де 1 вказує на найкращий можливий результат моделі [9].

Висновки до розділу 2

У другому розділі було проведено огляд методів інтелектуального аналізу для використання в задачі покращення розпізнавання об’єктів радару. Було розглянуто авторегресійний аналіз для побудови моделі сигналу та розглянуто

методи визначення коефіцієнтів цієї моделі. Також було досліджено структуру нейронних мереж.

РОЗДІЛ 3 ОПИС ПРОВЕДЕНИХ ДОСЛІДЖЕНЬ

3.1 Генерація сигналів

Для проведення роботи з даними, необхідно їх спочатку згенерувати. Для цього розглянемо процес генерації сигналів та деякі фундаментальні концепції, які лежать в основі цього процесу. Сюди входить генерація керуючих матриць, отримання сигналів з шумом і без нього, а також огляд співвідношення сигнал/шум (SNR). Розуміння цих концепцій має вирішальне значення для розробки передових радіолокаційних систем і алгоритмів обробки сигналів.

Радіолокаційний сигнал у нашій симуляції працює на частоті f = 77 ГГц, яка зазвичай використовується в автомобільних радарах завдяки своїй високій роздільній здатності та здатності виявляти невеликі об'єкти. Лямбда довжини хвилі радіолокаційного сигналу розраховується за формулою:

$$\lambda = c/f,$$

де c – швидкість світла.

Довжина хвилі є важливим параметром, оскільки впливає на конструкцію антенної решітки і відстань між елементами.

Антенна решітка складається з декількох антенних елементів, розташованих у певній конфігурації для прийому сигналів з різних напрямків. У нашому моделюванні ми використовуємо рівномірну лінійну антенну решітку (ULA) з M = 16 елементів. Відстань між сусідніми елементами d дорівнює половині довжини хвилі λ.

$$d = \frac{\lambda}{2}.$$

Така конфігурація гарантує, що масив може ефективно дискретизувати вхідні хвильові фронти без просторового зміщення.

Матриця керування A являє собою відгук антенної решітки на сигнал, що надходить з певного напрямку. Матриця керування для антени з N елементами і цілями, що приходять під кутами θ, має вигляд:

$$A = e^{i \cdot 2\pi \frac{d}{\lambda} sin(\theta)(0:N-1)},$$

де θ – це вектор кутів прибуття.

Керуюча матриця ефективно відображає просторові характеристики вхідних сигналів на елементи масиву.

Отримані сигнали від цілей моделюються як складний гаусівський шум. Для L=3 цілей та K=1361 часових відліків сигнали s генеруються наступним чином:

$$s = randn\ (L, K) + i\ \cdot randn\ (L, K).$$

Це випадкові сигнали від кожної цілі, що включають як реальні, так і уявні компоненти для імітації варіацій фази та амплітуди радіолокаційних сигналів.

Прийняті сигнали на антенній решітці без шуму x отримуються шляхом комбінування матриці управління A з сигналами цілей s:

$$x = A^T \cdot\ s.$$

Ця операція проектує цільові сигнали на елементи антени відповідно до напрямку приходу.

Для імітації реалістичного радіолокаційного середовища ми додаємо гаусівський шум до отриманих сигналів. Потужність шуму визначається на основі заданого відношення сигнал/шум (SNR) в децибелах (дБ):

$$NoisePower = \frac{SignalPower}{10^{SNR\_dB/10}}.$$

Шум генерується як складний гаусівський шум:

$$Noise = \sqrt{\frac{NoisePower}{2}} \cdot (randn(size(x)) + i \cdot randn(size(x))).$$

Прийняті сигнали з шумом x_noisy отримуються шляхом додавання згенерованого шуму до початкових прийнятих сигналів:

$$x_{noisy} = x + noise.$$

3.2 Збір експериментальних даних

Після успішного моделювання симульованих даних було вирішено провести дослідження зі справжніми сигналами та порівняти результати.

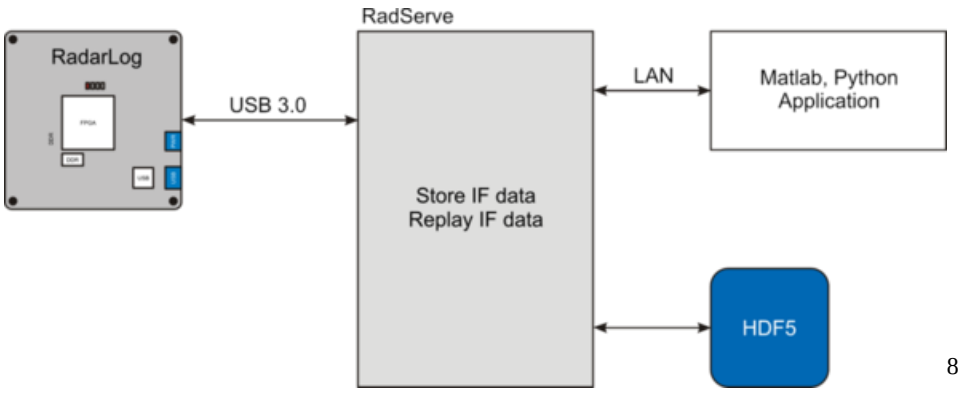
Для цього було обрано пристрій Radarlog – це середовище збору даних мікрохвильових радарів для комплексного аналізу даних і розробки алгоритмів.

[7]

Рисунок 3.1 – Зображення сенсорів використаного для збору даних пристрою Radarlog (праворуч)

За допомогою цього пристрою можна дискретизувати до 16 аналогових каналів прийому зі швидкістю 65 MSPS на канал, обробляти їх у потоковому режимі і зберігати через інтерфейс USB 3.0 зі швидкістю передачі даних до 1,6 Гбіт/с. Крім того, Radarlog може працювати з USB-сервером, який дозволяє записувати дані у файл у форматі HDF5 і одночасно візуалізувати отримані або оброблені дані [32].

На рис. 3.2 зображено схему роботи цього приладу.



[8]

Рисунок 3.2 – Схема роботи та передачі даних приладом Radarlog

[7] Джерело зображення: https://d3i71xaburhd42.cloudfront.net/3d5b57d924632332cdc02cfe3330e87ced6512b7/2-Figure2-1.png

[8] Джерело зображення: https://inras.at/wp-content/uploads/2021/11/radserve-01-a6.png

Для роботи з пристроєм було обрано Python інтерфейс. Код програми для конфігурації пристрою та збору даних знаходиться в додатку А.

Рисунок 3.3 – Проведення збору експериментальних даних

3.3 Моделювання симульованого сигналу за допомогою авторегресійних коефіцієнтів

Для наглядного розуміння змодельованого сигналу було побудовано графік з порівнянням оригінального та змодельованого за допомогою авторегресійних коефіцієнтів сигналів (рис. 3.1).

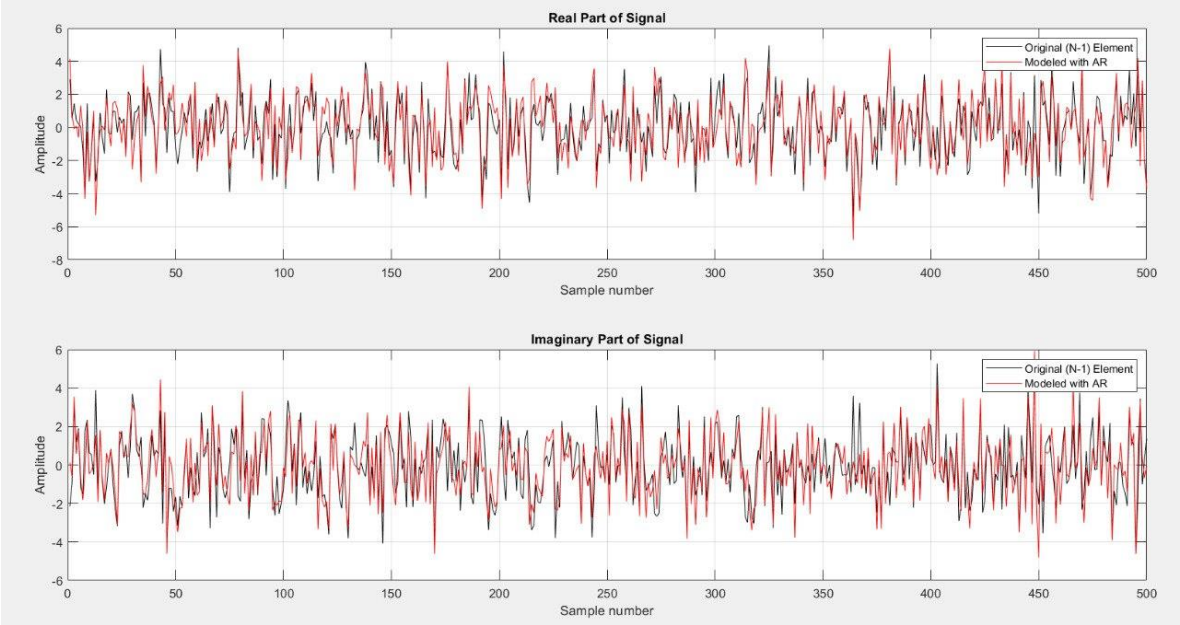


Рисунок 3.4 – Порівняння оригінального сигналу із змодельованим

Для подальшої валідації моделі було обраховано метрику MSE (Mean Squared Error):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

де y_i – оригінальні значення сигналу,

$\hat{y}_i$ – змодельовані значення.

Результат обрахунку метрики MSE:

- MSE (Real Part): 0.0001
- MSE (Imaginary Part): 0.0001

Інтерпретуючи результати, можна сказати, що значення середньої квадратичної помилки є дуже низькими, що вказує, що змодельовані сигнали близько співпадають з оригінальними на обох реальних та уявних частинах.

3.4 Порівняння методів

У цій главі представлено детальний аналіз оцінки напрямку прибуття (DoA) з використанням антенних решіток із застосуванням і без застосування екстраполяції апертури антени. Метрики ефективності аналізуються за допомогою методів Бартлетта і FFT для різних порядків авторегресійної моделі (4, 8, 12 і 16). Основні показники, що оцінюються, включають спектр Бартлетта, відношення піків до бічних сторін (PSLR).

3.2.1 Аналіз методу Бартлета

На рис. 3.4 показано спектр Бартлетта для вихідного масиву та з AR моделями 8, 12 та 16 порядків.

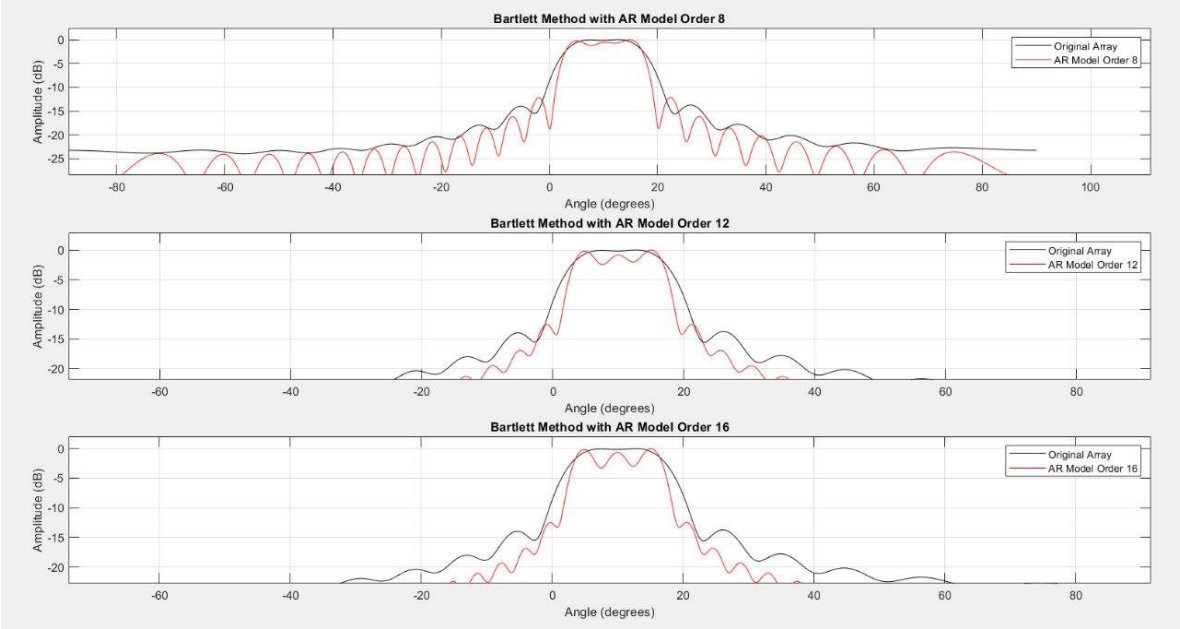


Рисунок 3.5 – Метод Бартлетта для вихідного та екстрапольованого масиву, згенеровані кути – 5°, 10°, 15°

Метод Бартлетта з AR-моделлю порядку 8:

- головна пелюстка є відносно гострою як для оригінального, так і для екстрапольованого спектрів;
- екстраполяція вводить вищі бічні пелюстки, що призводить до помірного покращення різкості головної пелюстки, але з підвищенням рівнів бічних пелюсток.

Метод Бартлетта з AR-моделлю порядку 12:

- головна пелюстка стає дещо гострішою порівняно з вихідним масивом;
- рівні бічних пелюсток зменшуються порівняно з 8-м порядком, що свідчить про кращу продуктивність.

Метод Бартлетта з AR-моделлю порядку 16:

- спостерігається найгостріша головна пелюстка, що значно покращує роздільну здатність;
- бічні пелюстки ще більше пригнічуються, що вказує на найкращу продуктивність серед оцінених порядків.

3.2.2 Аналіз методу ШПФ

На рис. 3.5 показано спектр FFT для вихідного масиву та з порядками AR моделі 4, 8, 16

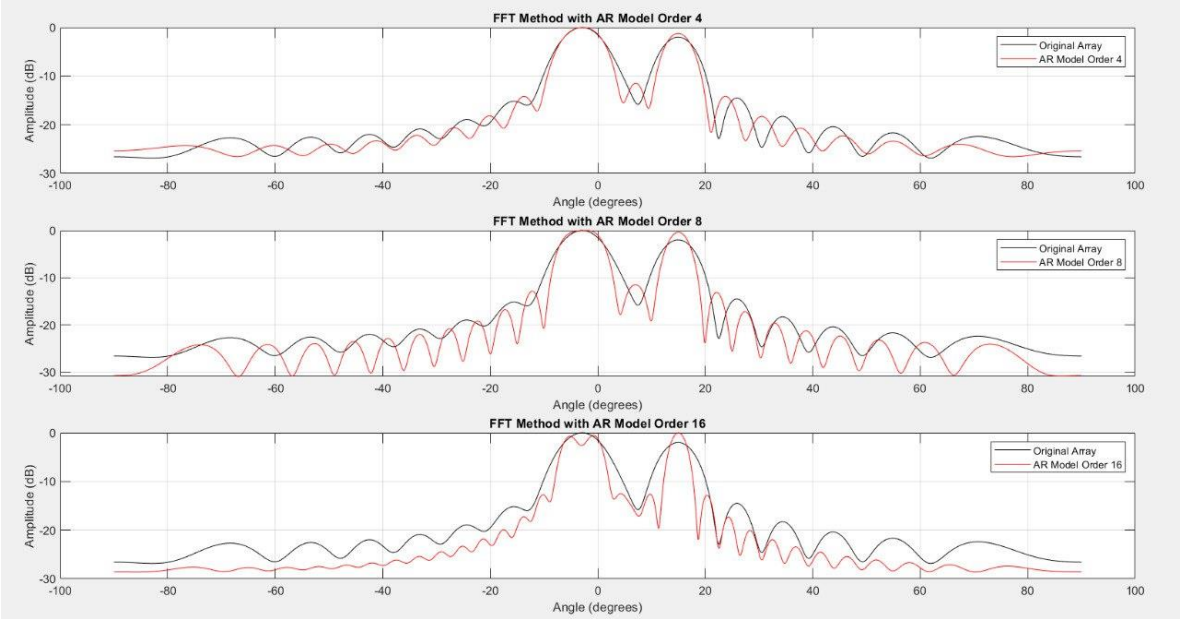


Рисунок 3.6 – Метод FFT для вихідного та екстрапольованого масиву, згенеровані кути – -5°, -1°, 15°

Метод ШПФ з AR-моделлю порядку 4:

- основна пелюстка ширша, а бічні пелюстки вищі, що вказує на менш точну оцінку DoA;
- екстраполяція покращує різкість головної пелюстки, але несуттєво.

Метод ШПФ з AR-моделлю порядку 8:

- основна частка ще більше загострюється, з помітним придушенням бічних часток порівняно з 4-м порядком;
- екстраполяція показує значне покращення, але бічні пелюстки все ще присутні.

Метод ШПФ з AR-моделлю 16-го порядку:

- головна пелюстка є найгострішою, забезпечуючи найкращу роздільну здатність;
- бічні пелюстки зведені до мінімуму, що вказує на найточнішу оцінку DoA серед оцінених порядків.

3.2.3 Показник ефективності PSLR

Відношення піку до бічної пелюстки (PSLR) - важливий показник в обробці сигналів і радіолокації, який використовується для вимірювання якості спрямованого спектра. Він визначається як відношення пікового значення головної пелюстки до найвищого рівня бічної пелюстки, зазвичай виражається в децибелах (дБ).

Вищий показник PSLR свідчить про кращу продуктивність. Зазвичай для оцінки PSLR використовують наступні рекомендації:

- PSLR > 20 дБ: це, як правило, вважається дуже хорошим показником. Це означає, що бічні пелюстки значно нижчі за основну пелюстку, що свідчить про чітке і ясне виявлення цілі з мінімальними перешкодами від бічних пелюсток;
- PSLR між 10 дБ і 20 дБ: цей діапазон прийнятний для багатьох застосувань. Бічні пелюстки присутні, але не створюють значних перешкод для основної пелюстки, що дозволяє досить точно виявляти та оцінювати цілі;
- PSLR < 10 дБ: зазвичай вважається поганим. Бічні пелюстки відносно високі, що може призвести до труднощів у розрізненні основної пелюстки і бічних пелюсток, що призводить до менш точного виявлення та оцінки цілі.

На рис. 3.6 показано порівняння PSLR без екстраполяції та з екстраполяцією 16-го порядку. Значне покращення PSLR спостерігається при екстраполяції 16-го порядку, що свідчить про кращу ефективність у розрізненні головної пелюстки від бічних. Проте варто зауважити, що при високому рівні шуму, моделювання сигналу за допомогою коефіцієнтів авторегресії буде також моделювати шум, відповідно посилюючи його в більш довготривалих

передбаченнях. Саме тому важливо правильно підібрати порядок моделі, враховуючи SNR (Sound to Noise Ratio) конкретної системи. Також методи позбавлення від шуму будуть вагомим доповненням до методу авторегресійного моделювання, так як дозволять прогнозувати в більшості змістовну частину сигналу, замість шуму.

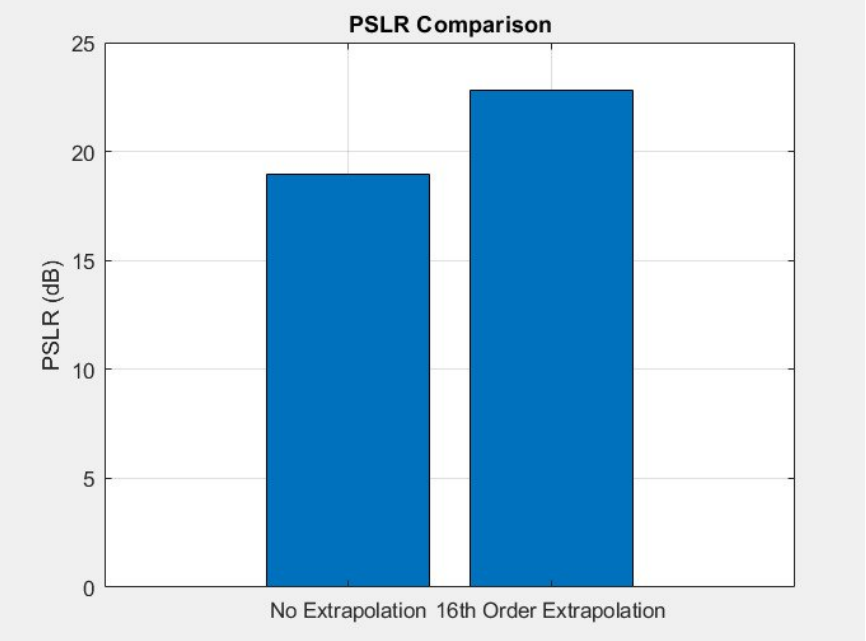


Рисунок 3.7 – Порівняння PSLR без екстраполяції та з екстраполяцією 16-го порядку

PSLR без екстраполяції: ~18 дБ

PSLR з екстраполяцією 16-го порядку: ~22 дБ

На рис.3.7 показано значення PSLR для різних порядків моделі AR. Очевидною є тенденція до збільшення PSLR з вищими порядками моделі AP, що вказує на те, що вищі порядки моделі AP забезпечують краще придушення бічних пелюсток.

Порядок моделі AP 8: ~22 дБ

Порядок моделі AP 12: ~22,5 дБ

Порядок моделі AP 16: ~23 дБ

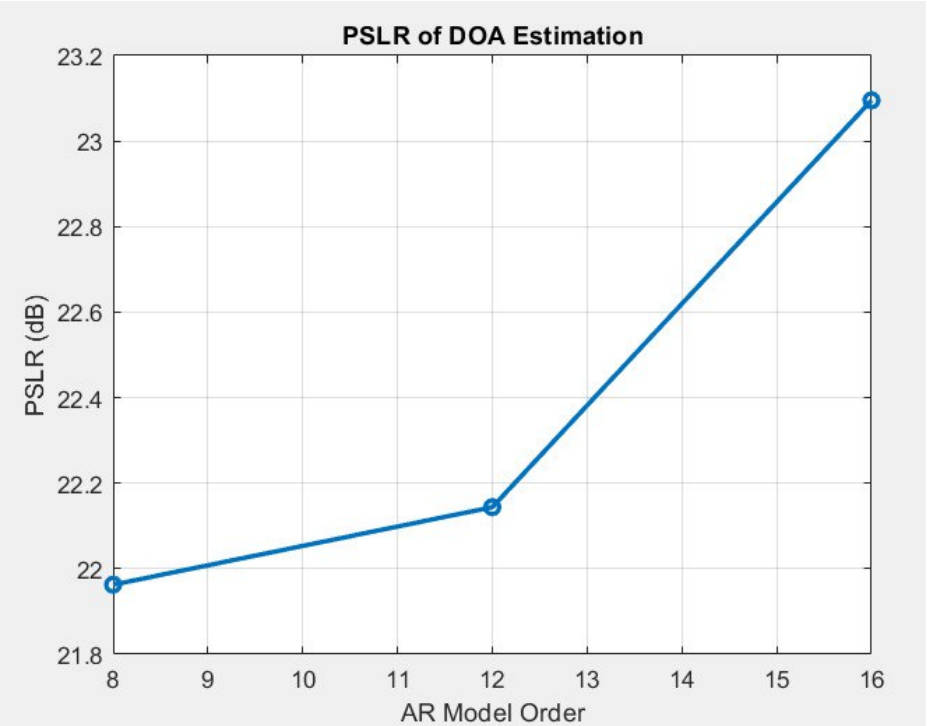


Рисунок 3.8 – Порівняння PSLR в залежності від порядку моделі

Отримавши позитивні результати лінійного передбачення на симульованих даних було проведено експеримент з використанням пристрою Radarlog. Під час цього було використано два відбивачі, розташовані на відстані менше 10 сантиметрів одне від одного. Пристрій було розташовано на 2 метрах від відбивачів. Підрахувавши кут за теоремою косинусу з трикутника 2м, 2м, 0,1м, знаходимо, що кут між відбивачами складає 3,6°. На рисунку 3.8 зображено цю конфігурацію та відповідь радару. Як бачимо, система з 16 антенами не може достатньо чітко розпізнати два об’єкти, розташовані на такому куті.

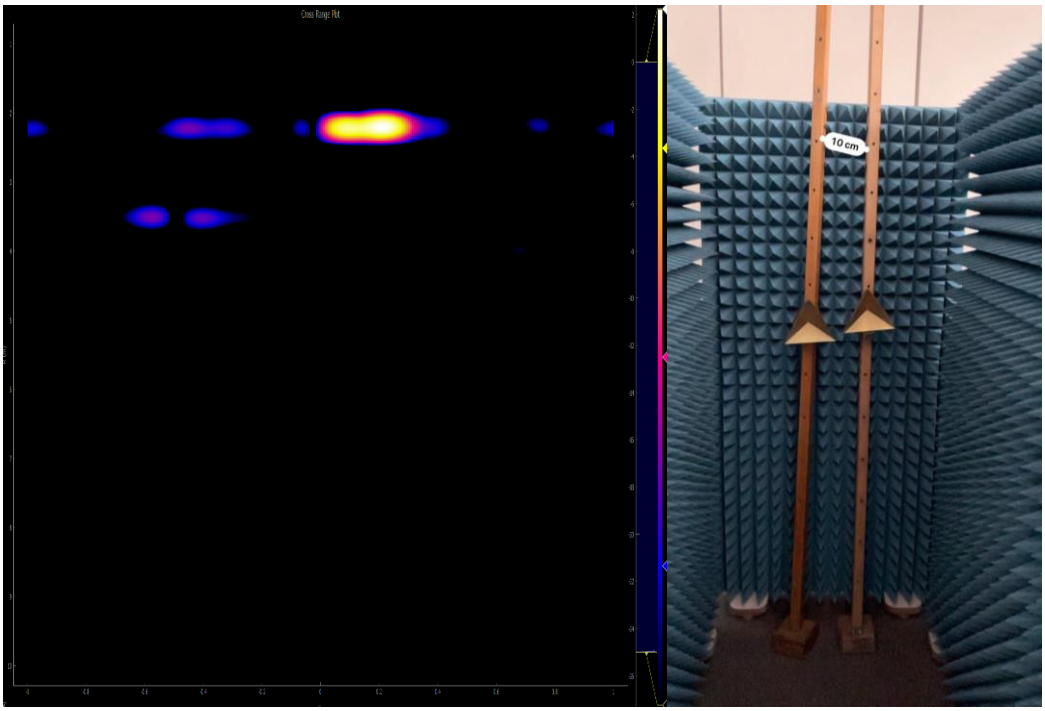


Рисунок 3.9 – Вимір проведений системою з 16 антенами

Враховуючи ту ж конфігурацію рефлекторів та відстані між ними та пристроєм, отримуємо результат реалізованого методу на рисунку 3.11. Як можна побачити, система чітко розпізнає дві окремі цілі.

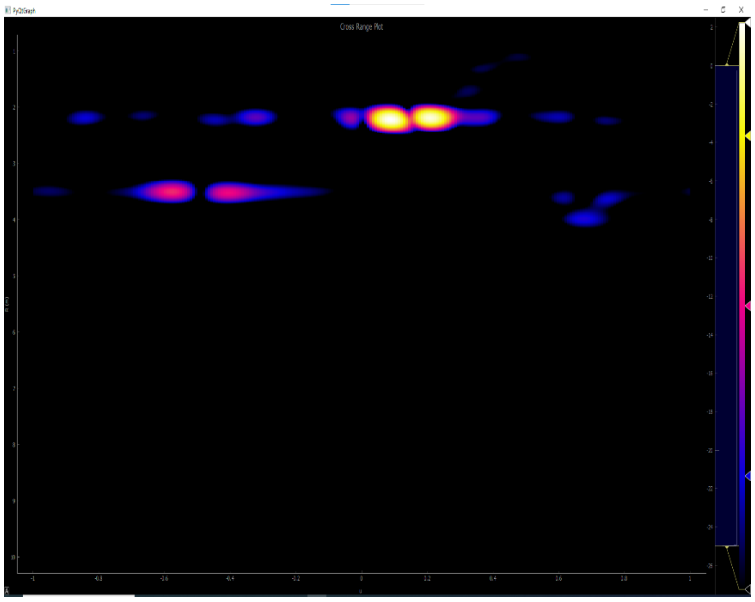


Рисунок 3.10 – Вимір, проведений системою з використанням екстраполяції антенної решітки

3.5 Оцінка напрямку прибуття за допомогою глибокого навчання

Окрім необхідності у високій роздільній здатності для точного визначення кута, сучасним радіолокаційним системам необхідно мати швидке та надійне програмне забезпечення, що має бути придатним до застосування в реальному часі. Насамперед, це передбачає оптимізацію алгоритмів обробки сигналів та зменшення обчислювальної складності. Для цього у роботі було реалізовано нейронну мережу за статтею Керема Мадена та Ішина Ерера «Оцінка напрямку прибуття в MIMO-радарах за допомогою глибокого навчання» [36].

У статті [36] розглядається складна проблема оцінки напрямку прибуття (DOA) в MIMO-радарах. Було визначено, що традиційні методи, такі як MUSIC і ESPRIT, вимагають великих обчислювальних ресурсів і погано працюють в умовах низького співвідношення сигнал/шум (SNR). Для подолання цих проблем у статті пропонується нова архітектура, яка поєднує в собі згорткові автокодери (DCAE) і згорткові нейронні мережі (CNN), що затушовують шум. Цей підхід покращує кутову роздільну здатність, зменшує час обробки та краще працює в зашумленому середовищі, що робить його придатним для застосувань у реальному часі.

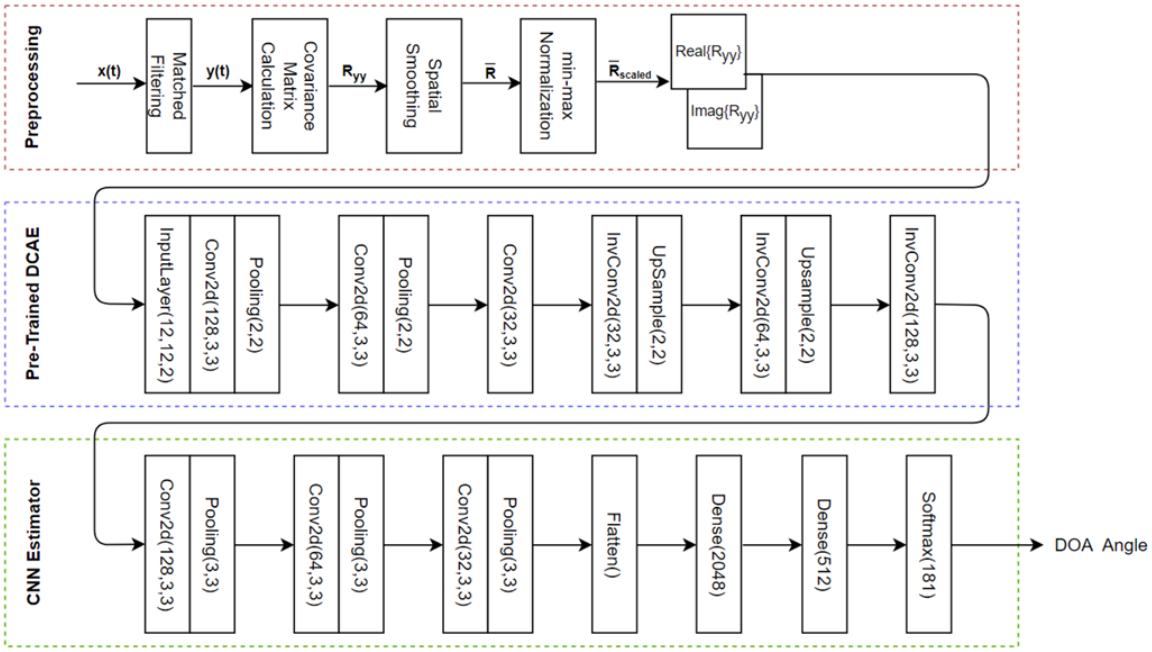


Рисунок 3.11 – Запропонована архітектура [36]

3.5.1 Структура та методологія запропонованої нейронної мережі

Попередня обробка

Етап попередньої обробки передбачає обчислення коваріаційної матриці вибірки з віртуального вектора даних, отриманого MIMO-радаром. Дійсна та уявна частини цієї коваріаційної матриці нормалізуються окремо. Просторове згладжування застосовується для зменшення розміру матриці і підвищення стійкості до шуму. Ці попередньо оброблені дані слугують вхідними даними для наступних нейронних мереж.

Згортковий автокодер зі згладжуванням (Denoising Convolutional Autoencoder, DCAE)

Архітектура DCAE складається з кодера та декодера:

- кодер: використовує згорткові шари з активацією ReLU, шари MaxPooling для дискретизації та шари BatchNormalization для стабілізації та прискорення навчання;
- декодер: дзеркальне відображення структури кодера зі згортковими шарами та шарами UpSampling і BatchNormalization для реконструкції деномінованої коваріаційної матриці.

Згорткова нейронна мережа (CNN):

CNN призначена для класифікації деномінованої коваріаційної матриці з метою оцінювання DOA.

- архітектура: складається з декількох згорткових та об'єднуючих шарів для виділення ознак, за якими слідують щільні шари для класифікації;
- активація: використовує активацію ReLU як для згорткових, так і для повністю з'єднаних шарів, з фінальним шаром softmax для класифікації;
- оптимізація: для тренування моделі використовую оптимізатор Adam та функцію втрат sparse categorical cross-entropy.

```
def build_cnn(input_shape, num_classes):
    model = tf.keras.models.Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(MaxPooling2D((2, 2)))
    model.add(BatchNormalization())
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(BatchNormalization())
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

Рисунок 3.12 – Реалізована архітектура моделі

Для тренування моделі я генерую великий набір даних із різними значеннями SNR. Спочатку навчаю DCAE на зашумлених даних для відновлення

чистої матриці коваріації. Потім використовую ці очищені дані для навчання CNN на класифікацію напрямку приходу сигналу.

```
def generate_training_data(num_samples, M, v_th):
    X_train = []
    Y_train = []
    for _ in range(num_samples):
        theta = np.random.choice(v_th)
        s = np.sum([np.exp(2j * np.pi * d / l
                           * np.sin(np.deg2rad(theta)) * m)
        for theta in [theta]], axis=0)
        Ryy = np.outer(s, s.conj())
        R_smoothed = spatial_smoothing(Ryy, M)
        R_real = R_smoothed.real
        R_imag = R_smoothed.imag

        R_real_normalized = (R_real - R_real_min) / (R_real_max - R_real_min)
        R_imag_normalized = (R_imag - R_imag_min) / (R_imag_max - R_imag_min)

        R_combined = np.stack((R_real_normalized, R_imag_normalized), axis=-1)

        X_train.append(R_combined)
        Y_train.append(np.argmin(np.abs(v_th - theta)))

    return np.array(X_train), np.array(Y_train)
```

Рисунок 3.13 – Генерація тренувальних даних

Результати навчання та валідації свідчать про успішне навчання моделі (рис. 3.11). Швидке зростання точності та зниження втрат на ранніх етапах, а також стабільність на пізніших етапах вказують на те, що модель ефективно навчилася виконувати завдання оцінки напрямку приходу сигналу. Висока точність на валідаційній вибірці свідчить про достатню генералізацію моделі на нових даних, що є важливим для реальних застосувань.

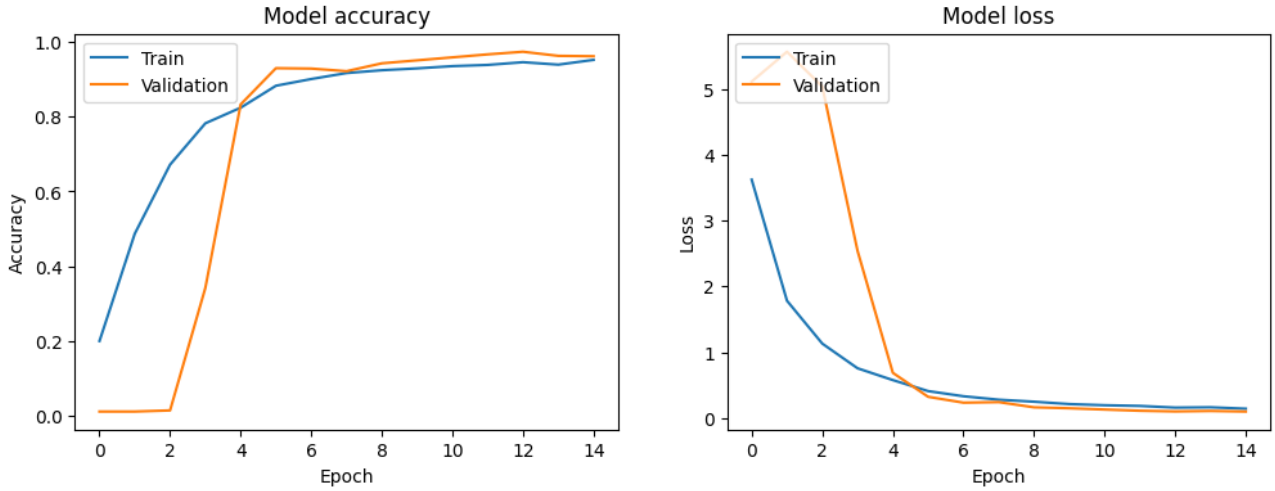


Рисунок 3.14 – Оцінка метрик імплементованої моделі

З рис. 3.12 бачимо, що три метрики – точність, повнота та F1-оцінка – мають значення близькі до 1.0. Це свідчить про те, що модель ефективно виконує своє завдання класифікації, з високою точністю передбачаючи позитивні приклади та з високою повнотою виявляючи всі позитивні приклади в наборі даних. Висока F1-оцінка додатково підтверджує, що модель має збалансовану продуктивність, забезпечуючи як точність, так і повноту на високому рівні. Ці результати демонструють, що модель є надійною та ефективною у виконанні поставленої задачі.

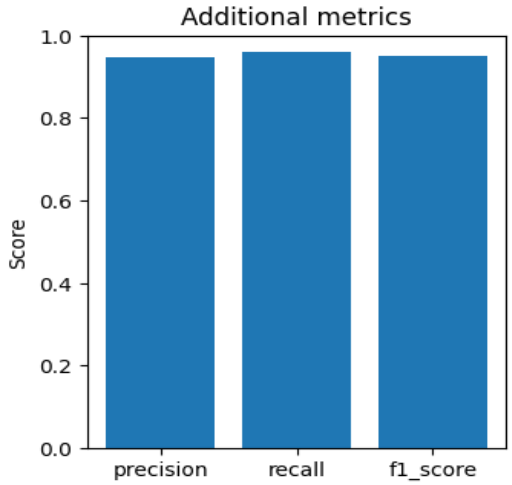


Рисунок 3.15 – Метрики Precision, Recall, F1 score

Висновки до розділу 3

У третьому розділі було продемонстровано реалізовані методи інтелектуального аналізу для покращення розпізнавання об’єктів. Було реалізовано екстраполяцію за допомогою авторегресивного моделювання та отримано результати, що вказують на покращення визначення кута. У другій частині розділу було досліджено наявну та імплементовано нейронну мережу для визначення кута прибуття. Було проаналізовано результати експериментів на симульованих та реальних даних радару.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У четвертому розділі ставиться завдання оцінити ключові характеристики майбутнього програмного продукту, який спеціалізується на покращенні розпізнаваня об’єктів радіолокаційними системами за допомогою методів інтелектуального аналізу даних. У дослідженні будуть представлені різні варіанти реалізації та підходи для забезпечення найоптимальнішої стратегії вибору, що впливатиме на економічні фактори та сумісність із майбутнім програмним продуктом. Для цього буде застосовано метод функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) є технологією, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Такий аналіз проводиться для виявлення резервів зниження витрат завдяки ефективнішим варіантам виробництва та кращому співвідношенню між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовуються економічні, технічні та конструкторські дані.

Алгоритм функціонально-вартісного аналізу включає визначення етапів розробки продукту, розрахунок повних річних витрат та кількості робочих годин, визначення джерел витрат і кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У цій роботі застосовується метод функціонально-вартісного аналізу (ФВА) для проведення техніко-економічного аналізу розробки системи методів

інтелектуального аналізу даних для покращення розпізнавання об’єктів. Оскільки рішення щодо проектування та реалізації компонентів впливають на всю систему, кожна окрема підсистема повинна відповідати її вимогам. Таким чином, фактичний аналіз охоплює аналіз функцій програмного продукту, призначеного для збору, обробки та аналізу даних компанії.

Можемо виділити такі вимоги до програмного продукту:

- висока точність результатів, що відповідає стандартам відповідної галузі;
- зручна та проста інтеграція в процес обробки сигналів;
- можливість ефективного масштабування;
- швидкодія у межах виконання в дійсному часі.

4.2 Обґрунтування функцій програмного продукту

Головна функція F0 – розробка програмного продукту, що розв’язує задачу покращення розпізнавання об’єктів у радіолокаційних системах. Ця функція визначає характеристики, що впливають на стабільність системи. На основі цієї функції, можна виділити наступні:

- F1 – вибір мови програмування;
- F2 – вибір реалізації методів інтелектуального аналізу;
- F3 – вибір середовища розробки.

Ці функції мають декілька варіантів реалізації:

Функція F1.

1. Python;
2. C++.

Функція F2.

1. Використання готових бібліотек для обраних методів роботи з даними та обробкою сигналів.
2. Розробка власних методів.

Функція F3.

1. Visual Studio Code;
2. Google Colab.

Варіанти реалізації основних функцій наведено у морфологічній карті системи (рис. 4.1).

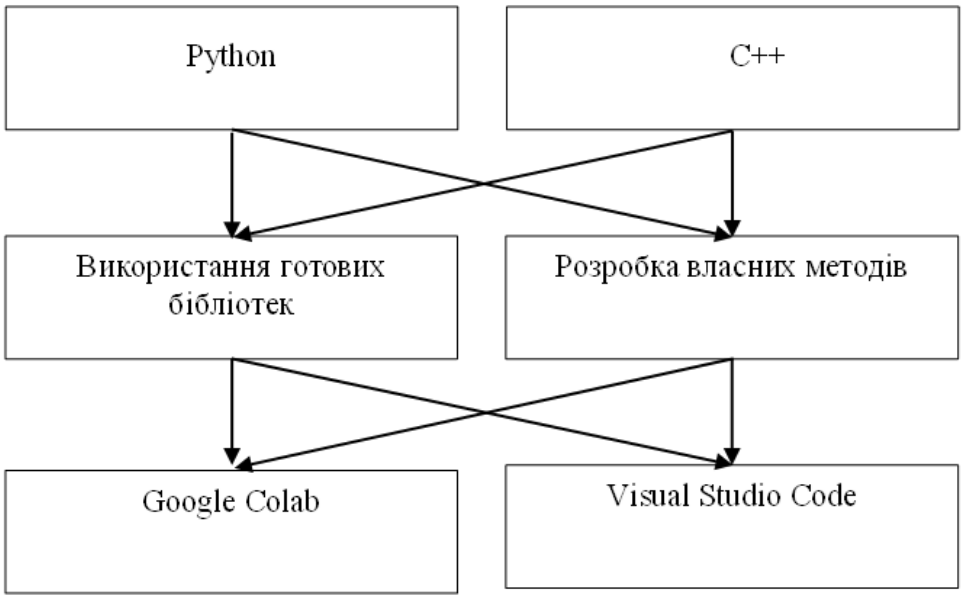


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину усіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 – Позитивно-негативна матриця

Функція	Варіант реалізації	Переваги	Недоліки
F_1	А	Велика кількість спеціально розроблених бібліотек і фреймворків для роботи моделюванням даних та створення нейромереж. Легкість у використанні, висока швидкість розробки.	Нижча швидкодія, обмежена керованість пам’яттю.
	Б	Висока продуктивність, краща керованість пам'яттю.	Відсутність необхідного інструментарію та бібліотек.
F_2	А	Швидка розробка, перевірені рішення, висока ефективність.	Обмежена гнучкість, залежність від сторонніх рішень та ліцензій.
	Б	Реалізація ПЗ з врахуванням особливостей поставленої задачі, додаткові можливості оптимізації.	Висока трудомісткість, необхідність глибоких знань алгоритмів.
F_3	А	Висока гнучкість, велика кількість плагінів, підтримка багатьох мов програмування.	Відсутність інтерактивних можливостей для роботи з даними.
	Б	Зручність для інтерактивної роботи з даними, підтримка багатьох мов, легкість у використанні, можливість використання ресурсів GPU.	Обмежена функціональність для великих проектів, вартість плану з використаннями більш потужних GPU.

Функція F1: перевагу надаємо загальнодоступності та швидкості реалізації; для спрощення написання коду варіант Б слід відхилити.

Функція F2: перевагу надаємо доступності в реалізації та зменшенню часових витрат на розробку програмного продукту; для спрощення написання коду варіант Б слід відхилити.

Функція F3: програма допускає вибір обох варіантів; можна використати варіанти А або Б.

Отже, проаналізувавши функції, будемо розглядати такі два варіанти реалізації ПП:

– $F_1(\text{А}) + F_2(\text{А}) + F_3(\text{А})$.

– $F_1(\text{А}) + F_2(\text{А}) + F_3(\text{Б})$.

Для оцінювання якості розглянутих функцій обрана система параметрів, яку описано нижче.

4.3 Обґрунтування системи параметрів програмного продукту

Виділивши два можливі варіанти реалізації, визначимо основні параметри вибору, які будуть використовуватись для розрахунку коефіцієнта технічного рівня. Для проектування характеристики програмного продукту будуть використані такі параметри:

- $X1$ – швидкодія мови програмування;
- $X2$ – об’єм пам’яті для обчислень та збереження даних;
- $X3$ – час обробки даних;
- $X4$ – потенційний об’єм програмного коду.

Визначення найгірших, середніх і найкращих значень параметрів проводиться з урахуванням вимог замовника та умов експлуатації програмного продукту, як зазначено в таблиці 4.2.

Таблиця 4.2 – Основні параметри програмного продукту

Найменування параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X_1	оп/мс	20	100	190
Об’єм пам’яті	X_2	Мб	2000	600	350
Час обробки даних	X_3	мс	100	50	20
Потенційний об’єм програмного коду	X_4	К-ть рядків коду	1000	500	350

За даними таблиці 4.2 побудовано графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

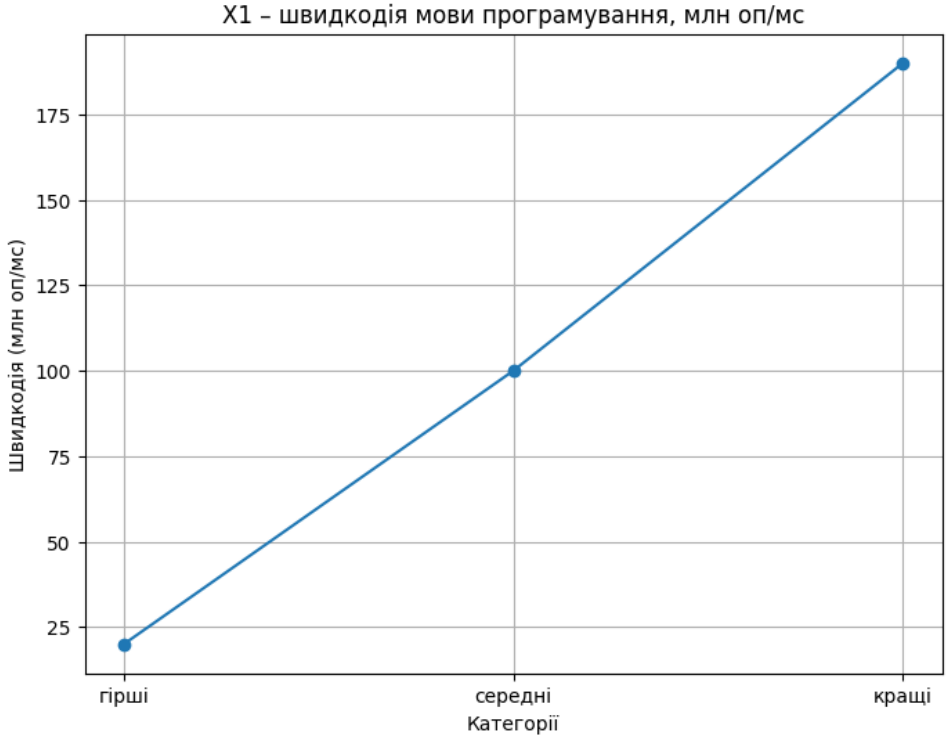


Рисунок 4.2 – Графік значень параметру X1 (швидкодія мови), млн оп/мс

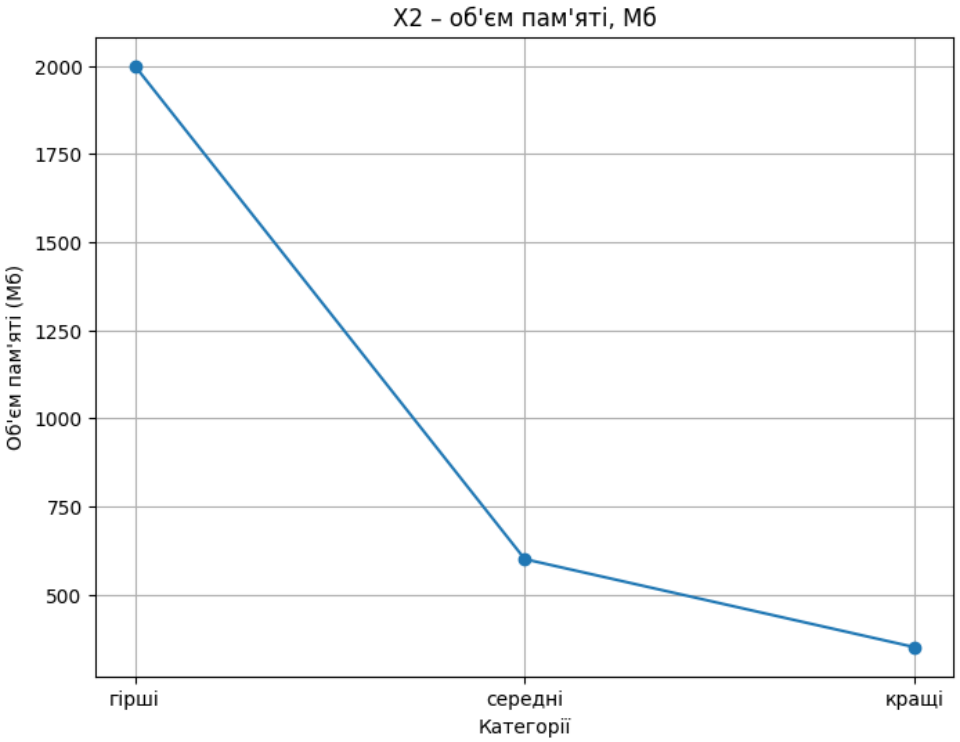


Рисунок 4.3 – Графік значень параметру X2 (об’єм пам’яті), Мб

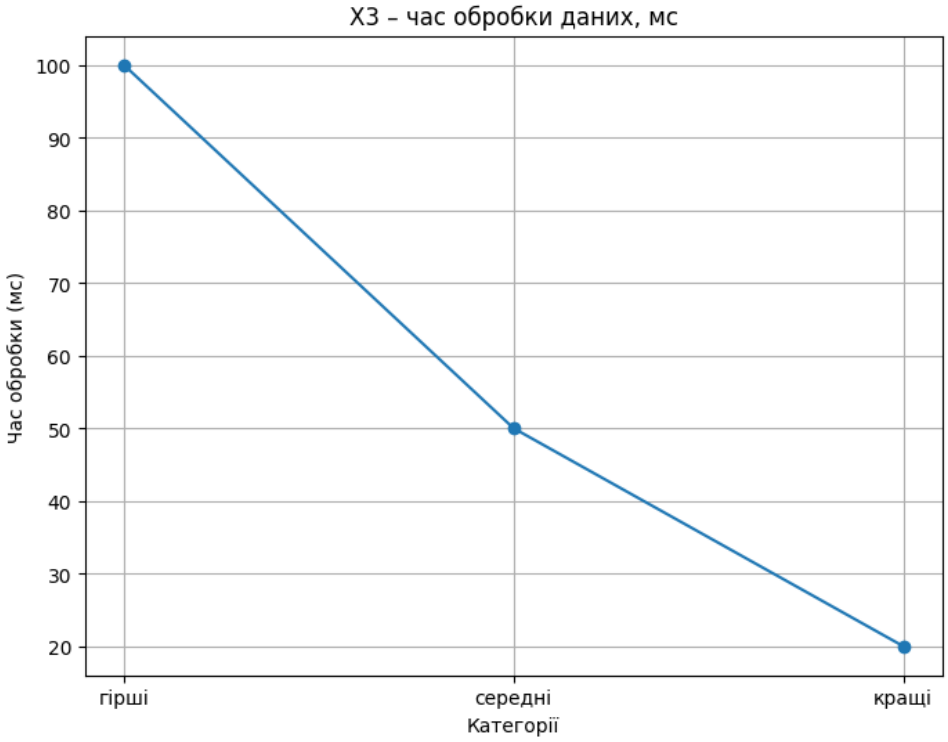


Рисунок 4.4 – Графік значень параметру X3 (час обробки даних), мс

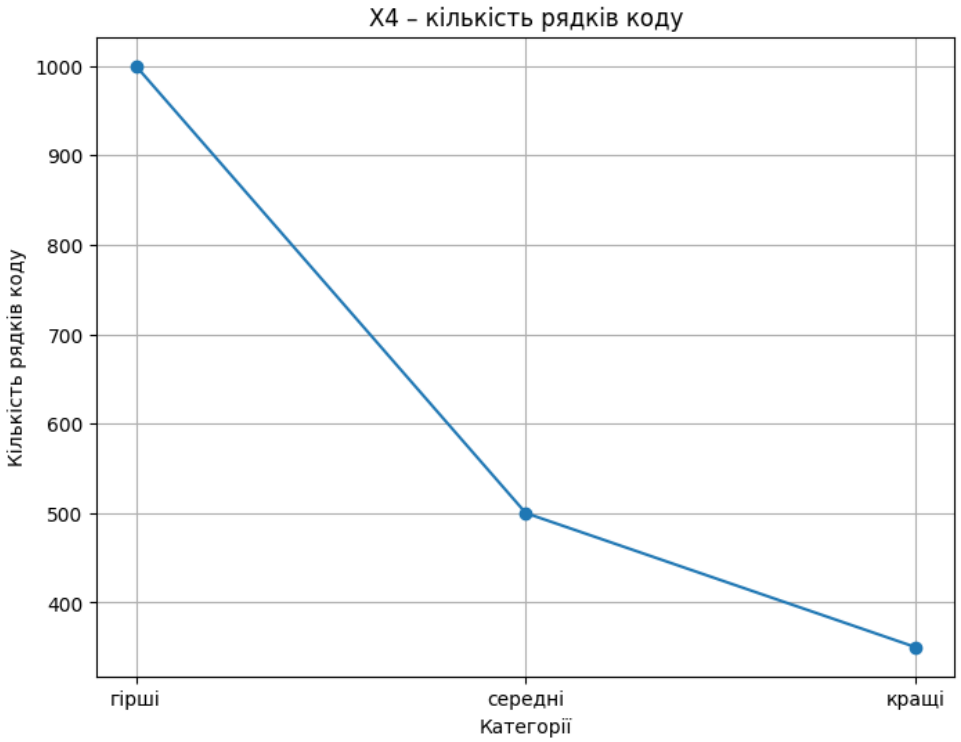


Рисунок 4.5 – Графік значень параметру X4 (кількість рядків коду)

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. 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 наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Найменування параметра	Умовне позначення	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
		1	2	3	4	5	6	7			
Швидкодія мови програмування	X_1	2	1	1	1	3	1	2	10	-7,5	56,25
Об’єм пам’яті	X_2	1	2	3	2	2	3	1	15	-2,5	6,25
Час обробки даних	X_3	3	3	2	3	1	2	3	17	-0,5	0,25
Потенційний об’єм програмного коду	X_4	4	4	4	4	4	4	4	28	10,5	110,25
Разом									70	0	173

Обрахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 173}{7^2(4^3 - 4)} = 0.706 > W_k = 0.67.$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0.67. Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 – Попарне порівняння параметрів.

Параметри	Експерт							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X_1 та X_2	>	<	<	>	<	>	<	>	1.5
X_1 та X_3	<	<	<	>	<	>	>	<	0.5
X_1 та X_4	<	<	<	<	<	<	<	<	0.5
X_2 та X_3	>	>	<	<	>	<	<	<	0.5
X_2 та X_4	<	<	<	<	<	<	<	<	0.5
X_3 та X_4	<	<	<	<	<	<	<	<	0.5

Числове значення, що визначає ступінь переваги i–го параметра над j–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j. \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

Розрахунок вагомості параметрів наведено в таблиці 4.5.

Таблиця 4.5 – Розрахунок вагомості параметрів

X_i	X_j				Ітерація					
					1		2		3	
	X_1	X_2	X_3	X_4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X_1	1	1,5	0,5	0,5	3,5	0.22	12.25	0,21	44,875	0,21
X_2	0,5	1	0,5	0,5	2,5	0.16	9.25	0,16	34,125	0,16
X_3	1,5	1,5	1	0,5	4,5	0.28	16.25	0,28	59,125	0,28
X_4	1,5	1,5	1,5	1	5,5	0.34	21.25	0,35	77,875	0,35
Всього					16	1	59	1	216	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X_1 (швидкодія мови програмування), X_3 (час обробки даних) та X_4 (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X_2 (об'єм пам'яті) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j},$$

де n – кількість параметрів;

$K_{вi}$– коефіцієнт вагомості i –го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F_1	А (X_1)	190	10	0,21	2,1
F_2	А (X_2)	600	5	0,16	0,8
F_3	А (X_4)	20	8	0,28	2,24
	Б (X_4)	350	4	0,35	1,4

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}].$$

визначаємо рівень якості кожного з варіантів:

$$\text{I}. K_K = 2{,}1 + 0{,}8 + 2{,}24 = 5{,}14.$$
$$\text{II}. K_K = 2{,}1 + 0{,}8 + 1{,}4 = 4{,}3.$$

Як видно з розрахунків, кращим є I варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки програмного продукту

Для визначення вартості розробки ПП спочатку необхідно провести розрахунок трудомісткості.

Всі варіанти включають два окремих завдання.

1. Розробка проекту програмного продукту.
2. Розробка програмної оболонки

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 та завдання 2 використовується довідкова інформація та інформація у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$\text{Т}_0 = \text{Т}_\text{Р} \cdot \text{К}_\text{П} \cdot \text{К}_\text{СК} \cdot \text{К}_\text{М} \cdot \text{К}_\text{СТ} \cdot \text{К}_\text{СТ.М},$$

де $Т_Р$ – трудомісткість розробки ПП,

$К_П$ – 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 А та групи складності алгоритму 1, трудомісткість дорівнює: $Т_Р$ =38 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх завдань рівний 1: $К_{СК}$ = 1. Враховуючи те, що при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.9. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 38 \cdot 1.7 \cdot 0.9 = 58{,}14 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =30 людино-днів, $К_П$ =0.8, $К_{СК}$ = 1, $К_{СТ}$ =0.8:

$$Т_2 = 30 \cdot 0.8 \cdot 0.8 = 19.2 \text{ людино-днів.}$$

Складемо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (58{,}14 + 19.2 + 4.8 + 19.2) \cdot 8 = 810.72 \quad \text{людино-годин.}$$

$$T_{II} = (58{,}14 + 19.2 + 7{,}6 + 19.2) \cdot 8 = 833.12 \quad \text{людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 23000 грн., та один аналітик даних з окладом 21000. Визначимо середню зарплату за годину за формулою:

$$С_ч = \frac{М}{Т_m \cdot t} \text{ грн,}$$

де М – місячний оклад працівників,

$Т_m$ – кількість робочих днів тиждень,

t – кількість робочих годин в день.

$$С_ч = \frac{23000+23000+21000}{3 \cdot 21 \cdot 8} = 132.936 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д,$$

де $С_ч$ – величина погодинної оплати праці програміста,

$Т_i$ – трудомісткість відповідного завдання,

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

Зарплата розробників за варіантами становить:

$$I.С_{ЗП} = 132.936 \cdot 810.72 \cdot 1.2 = 129328.6 \text{ грн.}$$

$$II.С_{ЗП} = 132.936 \cdot 833.12 \cdot 1.2 = 132901.9 \text{ грн.}$$

Відрахування на єдиний соціальний внесок становить 22%:

$$\text{I.}С_{ВІД} = С_{ЗП} \cdot 0.22 = 129328.6 \cdot 0.22 = 28452 \text{ грн.}$$

$$\text{II.}С_{ВІД} = С_{ЗП} \cdot 0.22 = 132901.9 \cdot 0.22 = 29238 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години ($С_М$).

Так як одна ЕОМ обслуговує одного програміста з окладом 23000 грн., з коефіцієнтом зайнятості 0.2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 23000 \cdot 0.2 = 55200 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г(1 + К_З) = 55200 \cdot (1 + 0.2) = 66240 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 66240 \cdot 0.22 = 13248 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 19000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.4 \cdot 0.25 \cdot 19000 = 6650 \text{ грн.}$$

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 як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.4 \cdot 19000 \cdot 0.08 = 2128 \text{ грн},$$

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) \cdot t \cdot К_В = (365 - 104 - 12 - 16) \cdot 8 \cdot 0.97 = \\ = 1808.08 \text{ годин},$$

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:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1808.08 \cdot 0.2 \cdot 0.8 \cdot 5.23 = 1513 \text{ грн},$$

де $N_С$ – середньо-споживча потужність приладу,

$К_З$ – коефіцієнтом зайнятості приладу,

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

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 19000 \cdot 0.67 = 12730 \text{ грн}.$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{ЕКС} = C_{ЗП} + C_{ВІД} + C_{А} + C_{Р} + C_{ЕЛ} + C_{Н} =$$

66240 + 29238 + 6650 + 2128 + 1513 + 12730 = 118499 грн

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{МГ} = \frac{C_{ЕКС}}{T_{ЕФ}} = \frac{118499}{1808.08} = 65.54 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_{М} = C_{МГ} \cdot T$$

$$I.C_{М} = 65.54 \cdot 810.72 = 53133.43 \text{ грн.}$$

$$II.C_{М} = 65.54 \cdot 833.12 = 54602.68 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$C_{Н} = C_{ЗП} \cdot 0.67$$

$$I.C_{Н} = 129328.6 \cdot 0.67 = 86650.16 \text{ грн.}$$

$$II.C_{Н} = 132901.9 \cdot 0.67 = 89044.27 \text{ грн.}$$

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_{М} + C_{Н}.$$

$$I.C_{ПП} = 129328.6 + 28452 + 53133.43 + 86650.16 = 297564.19 \text{ грн.}$$

$$II.C_{ПП} = 132901.9 + 29238 + 54602.68 + 89044.27 = 305786.85 \text{ грн.}$$

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР} = \frac{К_К}{С_{ПП}}$$

I. $$К_{ТЕР} = \frac{5,14}{297564.19} = 1.73 \cdot 10^{-5}.$$

II. $$К_{ТЕР} = \frac{4,3}{305786.85} = 1.41 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}$= 1,73 ·10^{-5}.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР}$ = 1,73 · 10^{-5}.

Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування – Python;
- використання готових бібліотек для обробки даних;
- інтерфейс розробки – Visual Studio Code.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку реалізацію програми та доступний функціонал для роботи.

Висновки до розділу 4

У цьому розділі було проведено повний функціонально-вартісний аналіз системи інтелектуального аналізу даних для покращення розпізнавання об’єктів, а також оцінено основні функції програмного продукту.

На першому етапі функціонально-вартісного аналізу було розглянуто можливі варіанти розробки системи. Проведено оцінку різних варіантів реалізації, що дозволило здійснити розрахунок коефіцієнта технічного рівня та обрати найкращу альтернативу.

Друга частина аналізу зосереджувалась на оцінці вартості реалізації програмного продукту. У межах цього етапу враховано заробітну плату працівників, витрати на електроенергію та обчислювальну техніку.

У результаті був обраний найбільш ефективний варіант, який вимагав найменших витрат.

ВИСНОВКИ

Дипломна робота являє собою дослідження методів інтелектуального аналізу даних для покращення алгоритмів розпізнавання об’єктів на прикладі радіолокаційних систем. У роботі було розглянуто два головних методи – на основі математичного моделювання та застосування нейронних мереж. Коректність роботи зазначених методів була підтверджена вдалим застосуванням на симульованих та експериментально згенерованих даних радару.

У першому розділі було проведено огляд предметної області, досліджено використані алгоритми обробки сигналів. Також було визначено потреби до покращення алгоритмів та відповідно сформовано постановку задачі.

У другому розділі розглянуто різноманітні підходи інтелектуального аналізу даних, та проведено їх порівняння. До методів описаних у цьому розділі входить лінійне передбачення з моделюванням сигналу за допомогою AR-моделей та методів знаходження коефіцієнтів моделі, а також дослідження структур нейронних мереж. У цьому розділі було детально досліджено математичну складову вище зазначених методів та розглянуто теоретичну базу глибокого навчання.

Третій розділ містить детальний опис та демонстрацію реалізованих методів інтелектуального аналізу та результати експериментів на симульованих та реальних даних радару. У цьому розділі міститься повна архітектура дослідженої нейронної мережі для визначення кута прибуття.

У четвертому розділі проведено функціонально-вартісний аналіз отриманої системи, визначено її основні функції та їх реалізації, виявлено коефіцієнт техніко-економічного рівня та вартість розробки повноцінної системи на основі представленої у роботі.

Підсумовуючи, робота виконала поставлені задачі і сформувала нові питання, що можуть і мають бути розглянуті у подальших дослідженнях.

ПЕРЕЛІК ПОСИЛАНЬ

1. Kronauge, M. (n.d.). *Waveform Design for Continuous Wave Radars*. [Thesis].
2. Swingler, D. N., & Walker, R. S. (1989). Line-array beamforming using linear prediction for aperture interpolation and extrapolation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(1), 16-22. doi: 10.1109/29.21701
3. Hayes, M. H. (2014). *Statistical Digital Signal Processing and Modeling*. New York: John Wiley & Sons.
4. Martínez Vázquez, M. (2022). Radar Resolution: How Accurate Can a Radar Be?. URL: https://www.renesas.com/us/en/blogs/radar-resolution-how-accurate-can-radar-be (дата звернення 19.04.24)
5. Triebe, O. J., Laptev, N., & Rajagopal, R. (2019). AR-Net: A Simple Auto-Regressive Neural Network for Time-Series. arXiv preprint arXiv:1911.12436.
6. Geoffrey Hinton, Oriol Vinyals, Jeff Dean. Distilling the Knowledge in a Neural Network. arXiv:1503.02531, 2015, DOI: https://doi.org/10.48550/arXiv.1503.02531 (дата звернення: 21.04.24).
7. Krose B., Van Der Smagt P. An Introduction to Neural Networks. Manuscript. The University of Amsterdam, 1996. 135 p.
8. Субботін, С. О., & Олійник, А. О. (2014). *Нейронні мережі*. Запоріжжя: Запорізький національний технічний університет.
9. VanderPlas, J. (2016). Python Data Science Handbook: Essential Tools for Working with Data. O'Reilly Media.
10. Palm, W. J. (2020). Introduction to MATLAB for Engineers. McGraw-Hill Education.
11. Gackenheimer, C. (2019). Exploring Visual Studio Code. Packt Publishing.
12. Регресійний аналіз: як застосовувати та інтерпретувати результати (2023). Mind the Graph. URL: https://mindthegraph.com/blog/uk/регресійний-аналіз/ (дата звернення: 28.04.24).

13. Gisder, T. (2021). *Methoden zur Integration eines automobilen Radars mit synthetischer Apertur* (Doctoral dissertation, Technische Universität München).
14. Autoregressive Models (2023). Penn State Eberly College of Science. URL: https://online.stat.psu.edu/stat501/lesson/t/t.2/t.2.1-autoregressive-models (дата звернення: 05.05.24).
15. Collomb, C. (2009). Burg's method, algorithm and recursion. *Comput. Sci*.
16. Mishra, C., & Gupta, D. L. (2017). Deep machine learning and neural networks: An overview. *IAES international journal of artificial intelligence*, *6*(2), 66.
17. Vos, K. (2013). A fast implementation of Burg's method. *OPUS codec*.
18. F. Li, H. Liu and R. J. Vaccaro, "Performance analysis for DOA estimation algorithms: unification, simplification, and observations," in IEEE Transactions on Aerospace and Electronic Systems, vol. 29, no. 4, pp. 1170-1184, Oct. 1993, doi: 10.1109/7.259520.
19. Eshel, G. (2003). The yule walker equations for the AR coefficients. *Internet resource*, *2*, 68-73.
20. Burg, J. P. (1975). *Maximum entropy spectral analysis*. Stanford University.
21. Sim, H., Lee, S., Kang, S., & Kim, S. C. (2019). Enhanced DOA estimation using linearly predicted array expansion for automotive radar systems. *IEEE Access*, *7*, 47714-47727.
22. Bhattacharya, R., & Waymire, E. C. (2022). *Stationary processes and discrete parameter Markov processes* (Vol. 293). Springer Nature.
23. Python. [Електронний ресурс]. — URL: https://www.python.org/
24. Matlab. [Електронний ресурс]. — URL: https://de.mathworks.com/products/matlab.html
25. Nielsen, M. A. (2015). *Neural networks and deep learning* (Vol. 25, pp. 15-24). San Francisco, CA, USA: Determination press.
26. NumPy: the fundamental package for scientific computing with Python. [Електронний ресурс]. — URL: https://numpy.org/.

27. Scikit-Learn. [Електронний ресурс]. — URL: https://scikitlearn.org/stable/index.html.
28. PyTorch. [Електронний ресурс]. — URL: https://pytorch.org/.
29. Dutt S. (2017). Deep Learning Architectures: A Comprehensive Guide. IBM Developer. [Електронний ресурс]. — URL: https://developer.ibm.com/articles/cc-machine-learning-deep-learningarchitectures/.
30. IBM. (н.д.). Deep Learning. [Електронний ресурс]. — URL: https://www.ibm.com/topics/deep-learning.
31. Müller A.C., Guido S. Introduction to Machine Learning with Python: A Guide for Data Scientists. - O'Reilly Media, 2018. — 394 с.
32. Goodfellow I., Bengio Y., Courville A. Deep Learning. — MIT Press, 2016. - 800 с.
33. Radarlog. [Електронний ресурс]. — URL: https://inras.at/en/radarlog/.
34. D. Swingler, R. Walker, and G. Lewis, ‘‘Linear prediction for aperture extrapolation in line array beamforming,’’ in Proc. IEEE Int. Conf. Acoust., Speech, Signal Process (ICASSP), Tokyo, Japan, Apr. 1986, pp. 2519–2522.
35. D. N. Swingler and R. S. Walker, ‘‘Line-array beamforming using linear prediction for aperture interpolation and extrapolation,’’ IEEE Trans. Acoust., Speech Signal Process., vol. 37, no. 1, pp. 16–30, Jan. 1989.
36. Maden, K., & Erer, I. (2022, November). DOA Estimation in MIMO Radars via Deep Learning. In *2022 30th Telecommunications Forum (TELFOR)* (pp. 1-4). IEEE.
37. Friedlander, B. (2021, October). The mythical uniform linear antenna array. In *2021 55th Asilomar Conference on Signals, Systems, and Computers* (pp. 221-225). IEEE.
38. Visual Studio Code. [Електронний ресурс]. — URL: https://code.visualstudio.com/.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

Симуляція сигналів та валідація методів в MATLAB

```
f = 77e9;
c = 299792458;
lambda = c / f;
M = 16;
d = lambda / 2;
N = M;
L = 3;
K = 1361;
SNR_dB = 40;

theta_sets = {
   [-1, 3],
   [-5, -1, 15],
   [5, 10, 15]
};

theta = theta_sets{3};

A = exp(2i * pi * d / lambda * sind(theta).' * (0:N-1));

s = randn(L, K) + 1i * randn(L, K);
x = A.' * s;

signal_power = mean(abs(x(:)).^2);
noise_power = signal_power / (10^(SNR_dB / 10));
```

```
noise = sqrt(noise_power/2) * (randn(size(x)) + 1i * randn(size(x)));
x_noisy = x + noise;

angles = -90:0.1:90;
angles_rad = deg2rad(angles);
steering_matrix = exp(1i * 2 * pi * d / lambda * (0:N-1).' * sin(angles_rad));

Rxx = x_noisy * x_noisy' / K;
bartlett_spectrum = zeros(length(angles), 1);
for i = 1:length(angles)
    bartlett_spectrum(i) = abs(steering_matrix(:, i)' * Rxx * steering_matrix(:, i));
end
bartlett_spectrum = 10 * log10(bartlett_spectrum / max(bartlett_spectrum));

[~, estimated_indices] = maxk(bartlett_spectrum, L);
estimated_angles = angles(estimated_indices);
rmse_no_extrap = sqrt(mean((sort(estimated_angles) - sort(theta)).^2));

peak_value = max(bartlett_spectrum);
sidelobe_value = mean(bartlett_spectrum(bartlett_spectrum < peak_value));
pslr_no_extrap = peak_value - sidelobe_value;

ar_orders = [8, 12, 16];
fft_spectra_extrap = zeros(length(angles), length(ar_orders));
bartlett_spectra_extrap = zeros(length(angles), length(ar_orders));

for o = 1:length(ar_orders)
    P = ar_orders(o);
    x_extrapolated = zeros(P, K);
```

```
    for k = 1:K
        ar_coeffs = arburg(x_noisy(:, k), N-1);

        for p = 1:P
            x_segment_ext = flipud([x_noisy(end-N+p:end, k); x_extrapolated(1:p-1,
k)]);
            x_extrapolated(p, k) = -ar_coeffs(2:end) * x_segment_ext(1:N-1);
        end
    end

    x_combined = [x_noisy; x_extrapolated];
    Rxx_combined = x_combined * x_combined' / K;

    steering_matrix_combined = exp(1i * 2 * pi * d / lambda * (0:N+P-1).' *
sin(angles_rad));

    bartlett_spectrum_extrap = zeros(length(angles), 1);
    for i = 1:length(angles)
        bartlett_spectrum_extrap(i) = abs(steering_matrix_combined(:, i)' *
Rxx_combined * steering_matrix_combined(:, i));
    end
    bartlett_spectrum_extrap = 10 * log10(bartlett_spectrum_extrap /
max(bartlett_spectrum_extrap));
    bartlett_spectra_extrap(:, o) = bartlett_spectrum_extrap;

    fft_spectrum_extrap = zeros(length(angles), 1);
    for i = 1:length(angles)
        fft_spectrum_extrap(i) = abs(steering_matrix_combined(:, i)' * Rxx_combined *
steering_matrix_combined(:, i));
    end
```

```
    fft_spectrum_extrap = 10 * log10(fft_spectrum_extrap /
max(fft_spectrum_extrap));
    fft_spectra_extrap(:, o) = fft_spectrum_extrap;
end

[~, estimated_indices_extrap] = maxk(bartlett_spectrum_extrap, L);
estimated_angles_extrap = angles(estimated_indices_extrap);
rmse_extrap = sqrt(mean((sort(estimated_angles_extrap) - sort(theta)).^2));

peak_value_extrap = max(bartlett_spectrum_extrap);
sidelobe_value_extrap = mean(bartlett_spectrum_extrap(bartlett_spectrum_extrap <
peak_value_extrap));
pslr_extrap = peak_value_extrap - sidelobe_value_extrap;

figure;
subplot(2, 1, 1);
plot(angles, bartlett_spectrum, 'k', 'DisplayName', 'Original Array');
hold on;
plot(angles, bartlett_spectrum_extrap, 'r', 'DisplayName', '16th Order AR Model');
xlabel('Angle (degrees)');
ylabel('Amplitude (dB)');
title('Bartlett Method Comparison');
legend('show');
grid on;

subplot(2, 1, 2);
bar([1, 2], [rmse_no_extrap, rmse_extrap]);
set(gca, 'xticklabel', {'No Extrapolation', '16th Order Extrapolation'});
ylabel('RMSE (degrees)');
title('RMSE Comparison');
```

```
grid on;

figure;
bar([1, 2], [pslr_no_extrap, pslr_extrap]);
set(gca, 'xticklabel', {'No Extrapolation', '16th Order Extrapolation'});
ylabel('PSLR (dB)');
title('PSLR Comparison');
grid on;

fprintf('RMSE without Extrapolation: %.2f degrees\n', rmse_no_extrap);
fprintf('RMSE with 16th Order Extrapolation: %.2f degrees\n', rmse_extrap);
fprintf('PSLR without Extrapolation: %.2f dB\n', pslr_no_extrap);
fprintf('PSLR with 16th Order Extrapolation: %.2f dB\n', pslr_extrap);

figure;
for o = 1:length(ar_orders)
    subplot(length(ar_orders)
```

Реалізація екстраполяції на реальних даних радару в Python

```
import sys, os
sys.path.append("../")
import src.cmd_modules.IfxTx2Rx16 as IfxTx2Rx16
import numpy as np
import numpy.matlib
from pyqtgraph.Qt import QtGui, QtCore
import pyqtgraph as pg
from PyQt5 import QtWidgets
from pyqtgraph.widgets.GraphicsView import GraphicsView

def burg_method(x, order):
    N = len(x)
    ar_coeffs = np.zeros(order + 1, dtype=complex)
    ef = np.zeros(N, dtype=complex)
    eb = np.zeros(N, dtype=complex)
    ar_coeffs[0] = 1
    ef[1:] = x[1:]
    eb[:-1] = x[:-1]
    E = np.dot(x, x.conj())

    for k in range(1, order + 1):
        num = -2 * np.dot(eb[:N-k], ef[:N-k].conj())
        denom = np.dot(ef[:N-k], ef[:N-k].conj()) + np.dot(eb[:N-k], eb[:N-k].conj())
        gamma = num / denom
        ar_coeffs[k] = gamma

        ef_old = ef[:N-k].copy()
        eb_old = eb[:N-k].copy()
```

```
        ef[:N-k] += gamma * eb_old
        eb[:N-k] = eb_old + gamma.conj() * ef_old

        ar_coeffs[1:k] += gamma * ar_coeffs[k-1:0:-1]

        E *= 1 - abs(gamma) ** 2

    return ar_coeffs

def extrapolate_signal(x, P, N, K):
    x_extrapolated = np.zeros((P, K), dtype=complex)
    for k in range(K):
        ar_coeffs = burg_method(x[:, k], N-1)
        for p in range(P):
            x_segment_ext = np.flipud(np.concatenate([x[-N+p:, k], x_extrapolated[:p,
k]]))
            x_extrapolated[p, k] = -np.dot(ar_coeffs[1:], x_segment_ext[:N-1])
    return np.concatenate([x, x_extrapolated], axis=0)

# Display configurations
Disp_FrmNr = 1
Disp_TimSig = 0  # display time signals
Disp_RP = 1  # display range profile
Disp_JOpt = 1  # display cost function for DBF

c0 = 1 / np.sqrt(8.85e-12 * 4 * np.pi * 1e-7)

if Disp_TimSig > 0 or Disp_JOpt > 0:
    App = QtWidgets.QApplication([])
```

```
if Disp_TimSig > 0:
    WinTim = pg.GraphicsWindow(title="Time signals")
    WinTim.setBackground((255, 255, 255))
    WinTim.resize(1000, 600)
    PltTim = WinTim.addPlot(title="TimSig", col=0, row=0)
    PltTim.showGrid(x=True, y=True)

if Disp_JOpt:
    View = pg.PlotItem(title='Cross Range Plot')
    View.setLabel('left', 'R', units='m')
    View.setLabel('bottom', 'u')
    Img = pg.ImageView(view=View)
    Img.show()
    Img.ui.roiBtn.hide()
    Img.ui.menuBtn.hide()
    Img.getHistogramWidget().gradient.loadPreset('flame')

# Setup Connection
Brd = IfxTx2Rx16.IfxTx2Rx16('RadServe', '127.0.0.1')

# Reset Board and Enable Power Supply
Brd.BrdRst()
Brd.BrdPwrEna()

# Software Version
Brd.BrdDispSwVers()

# Read Calibration data
dCalCfg = dict()
dCalCfg['Mask'] = 1
```

```
dCalCfg['Len'] = 64
CalData = Brd.BrdGetCalData(dCalCfg)

# Enable Receive Chips
Brd.RfRxEna()

# Enable Transmit Chips
TxChn = 1
TxPwr = 60
Brd.RfTxEna(TxChn, TxPwr)

# Configure Connection for RadarLog
Brd.ConSet('Mult', 2)
Brd.ConSet('BufSiz', 64)

# Use clock from frontend with 80 MHz for synchronous sampling
Brd.Set('ClkSrc', 2, 80e6)

# Configure AFE5801
Brd.Set('fAdc', 40e6)
Brd.Set('AfeLowNoise', 0)
Brd.Set('AfeIntDcCoupling', 0)
Brd.Set('AfePatRamp', 'Off')
Brd.Set('AfeGaindB', 20)

# Configure Up-Chirp
dCfg = dict()
dCfg['fStrt'] = 76e9
dCfg['fStop'] = 77e9
dCfg['TRampUp'] = 204.8e-6
```

```
dCfg['TRampDo'] = 64e-6
dCfg['TInt'] = 200e-3
dCfg['Tp'] = 500e-6
dCfg['N'] = 512
dCfg['NrFrms'] = 200
dCfg['TxPwr'] = 60

lRampCfg = list()
dRampCfg = dict()
dRampCfg['fStrt'] = dCfg['fStrt']
dRampCfg['fStop'] = dCfg['fStop']
dRampCfg['TRamp'] = dCfg['TRampUp']
dRampCfg['RampCfg'] = Brd.SampSeq + Brd.PaCtrl_Tx1
lRampCfg.append(dRampCfg)

dRampCfg = dict()
dRampCfg['fStrt'] = dCfg['fStop']
dRampCfg['fStop'] = dCfg['fStrt']
dRampCfg['TRamp'] = dCfg['TRampDo']
dRampCfg['TWait'] = dCfg['Tp'] - dCfg['TRampDo'] - dCfg['TRampUp']
dRampCfg['RampCfg'] = Brd.PaCtrl_Tx2
lRampCfg.append(dRampCfg)

dRampCfg = dict()
dRampCfg['fStrt'] = dCfg['fStrt']
dRampCfg['fStop'] = dCfg['fStop']
dRampCfg['TRamp'] = dCfg['TRampUp']
dRampCfg['RampCfg'] = Brd.SampSeq + Brd.PaCtrl_Tx2
lRampCfg.append(dRampCfg)
```

```
dRampCfg = dict()
dRampCfg['fStrt'] = dCfg['fStrt']
dRampCfg['fStop'] = dCfg['fStop']
dRampCfg['TRamp'] = dCfg['TRampUp']
dRampCfg['TWait'] = dCfg['Tp'] - dCfg['TRampDo'] - dCfg['TRampUp']
dRampCfg['RampCfg'] = Brd.PaCtrl_Tx1
lRampCfg.append(dRampCfg)

Brd.RfMeas('RccMs', lRampCfg, dCfg)

# Read Settings for N and fs
N = int(Brd.Get('N'))
fs = Brd.Get('fs')
NrChn = int(Brd.Get('NrChn'))

# Configure Signal Processing
NFFT = 2**12
Win2D = Brd.hanning(N-1, 2*NrChn-1)
ScaWin = np.sum(Win2D[:, 0])
kf = Brd.RfGet('kf')
vRange = np.arange(NFFT) / NFFT * fs * c0 / (2 * kf)

RMin = 1
RMax = 10
RMinIdx = np.argmin(np.abs(vRange - RMin))
RMaxIdx = np.argmin(np.abs(vRange - RMax))
vRangeExt = vRange[RMinIdx:RMaxIdx]

NFFTAnt = 256
WinAnt2D = Brd.hanning(2*NrChn-1, len(vRangeExt))
```

```
ScaWinAnt = np.sum(WinAnt2D[:, 0])
WinAnt2D = WinAnt2D.T
vAngDeg = np.arcsin(2 * np.arange(-NFFTAnt//2, NFFTAnt//2) / NFFTAnt) / np.pi
* 180

CalData = np.concatenate((CalData[0:NrChn], CalData[NrChn+1:]))
mCalData = np.matlib.repmat(CalData, N-1, 1)

if Disp_TimSig:
    n = np.arange(int(N))
    CurveTim = []
    for IdxChn in np.arange(2*NrChn-1):
        CurveTim.append(PltTim.plot(pen=pg.intColor(IdxChn, hues=2*NrChn-1)))

# Measure and calculate DBF
for Cycles in range(0, 1000):
    Data = Brd.BrdGetData(2)

    if Disp_FrmNr > 0:
        print(Data[0, :])

    DataV = np.concatenate((Data[1:N, :], Data[N+1:, 1:]), axis=1)
    np.save('angular_fft_data.npy', DataV)

    RP = 2 * np.fft.fft(DataV[:, :] * Win2D * mCalData, n=NFFT, axis=0) / ScaWin *
Brd.FuSca
    RP = RP[RMinIdx:RMaxIdx, :]

    # Extrapolate the signal
    ar_orders = [8, 12, 16]
```

```
    for P in ar_orders:
        DataV_extrapolated = extrapolate_signal(DataV, P, N, K=DataV.shape[1])
        RP_extrapolated = 2 * np.fft.fft(DataV_extrapolated[:, :] * Win2D * mCalData,
n=NFFT, axis=0) / ScaWin * Brd.FuSca
        RP_extrapolated = RP_extrapolated[RMinIdx:RMaxIdx, :]

        if Disp_JOpt > 0:
            JOpt = np.fft.fftshift(np.fft.fft(RP_extrapolated * WinAnt2D, NFFTAnt,
axis=1) / ScaWinAnt, axes=1)

            JdB = 20 * np.log10(np.abs(JOpt))
            JMax = np.max(JdB)
            JNorm = JdB - JMax
            JNorm[JNorm < -25] = -25

            Img.setImage(JNorm.T, pos=[-1, RMin], scale=[2.0 / NFFTAnt, (RMax -
RMin) / vRangeExt.shape[0]])
            View.setAspectLocked(False)

    pg.QtGui.QGuiApplication.processEvents()

Brd.CloseDataPort()
Brd.BrdRst()
Brd.BrdPwrDi()

del Brd
```

Реалізація та валідація моделі глибокого навчання

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D,
Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.models import Model
import matplotlib.pyplot as plt

f = 77e9
c = 299792458
M = 16
l = c / f
d = l / 2
v_th = np.linspace(-90, 90, 181)
v_th0 = np.array([-3.5])

m = np.arange(M)
s = np.sum([np.exp(2j * np.pi * d / l * np.sin(np.deg2rad(theta)) * m) for theta in
v_th0], axis=0)

Ryy = np.outer(s, s.conj())

def spatial_smoothing(Ryy, M):
    subarrays = [Ryy[i:i+M, i:i+M] for i in range(Ryy.shape[0] - M + 1)]
    R = np.sum(subarrays, axis=0)
    return R / (Ryy.shape[0] - M + 1)

R_smoothed = spatial_smoothing(Ryy, M)
```

```
R_real = R_smoothed.real
R_imag = R_smoothed.imag

R_real_min, R_real_max = R_real.min(), R_real.max()
R_imag_min, R_imag_max = R_imag.min(), R_imag.max()

R_real_normalized = (R_real - R_real_min) / (R_real_max - R_real_min)
R_imag_normalized = (R_imag - R_imag_min) / (R_imag_max - R_imag_min)

R_combined = np.stack((R_real_normalized, R_imag_normalized), axis=-1)
R_combined_reshaped = R_combined.reshape((1, *R_combined.shape))

def build_dcae(input_shape):
    input_img = Input(shape=input_shape)
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = BatchNormalization()(x)
    x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = BatchNormalization()(x)
    x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = UpSampling2D((2, 2))(x)
    x = BatchNormalization()(x)
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
    x = UpSampling2D((2, 2))(x)
    x = BatchNormalization()(x)
    decoded = Conv2D(2, (3, 3), activation='sigmoid', padding='same')(x)
    autoencoder = Model(input_img, decoded)
    autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

```
    return autoencoder

def generate_training_data(num_samples, M, v_th):
    X_train = []
    Y_train = []
    for _ in range(num_samples):
        theta = np.random.choice(v_th)
        s = np.sum([np.exp(2j * np.pi * d / l * np.sin(np.deg2rad(theta)) * m) for theta in
[theta]], axis=0)
        Ryy = np.outer(s, s.conj())
        R_smoothed = spatial_smoothing(Ryy, M)
        R_real = R_smoothed.real
        R_imag = R_smoothed.imag
        R_real_normalized = (R_real - R_real_min) / (R_real_max - R_real_min)
        R_imag_normalized = (R_imag - R_imag_min) / (R_imag_max - R_imag_min)
        R_combined = np.stack((R_real_normalized, R_imag_normalized), axis=-1)
        X_train.append(R_combined)
        Y_train.append(np.argmin(np.abs(v_th - theta)))
    return np.array(X_train), np.array(Y_train)

num_samples = 5000
X_train, Y_train = generate_training_data(num_samples, M, v_th)

dcae = build_dcae(X_train.shape[1:])
dcae.fit(X_train, X_train, epochs=50, batch_size=256, shuffle=True,
validation_split=0.2)

X_train_denoised = dcae.predict(X_train)

def build_cnn(input_shape, num_classes):
```

```
    model = tf.keras.models.Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(MaxPooling2D((2, 2)))
    model.add(BatchNormalization())
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(BatchNormalization())
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model

num_classes = len(v_th)
cnn = build_cnn(X_train_denoised.shape[1:], num_classes)

history = cnn.fit(X_train_denoised, Y_train, epochs=15, batch_size=32,
validation_split=0.2)

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
```

```
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

R_denoised = dcae.predict(R_combined_reshaped)
doa_prediction = cnn.predict(R_denoised)
predicted_angle_index = np.argmax(doa_prediction, axis=1)
predicted_angle = v_th[predicted_angle_index]
print(f"Predicted DOA: {predicted_angle} degrees")
```