НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____ Олександр КОВАЛЬ
“___”_____2021 р.

Дипломна робота

на здобуття ступеня бакалавра

спеціальності 121 “Інженерія програмного забезпечення”

освітня програма “Інженерія програмного забезпечення розподілених систем”

на тему: “Автоматизована система адаптивної фільтрації гідроакустичних сигналів на основі машинного навчання”

Виконав:
студент IV курсу, групи ТВ-з71
Козак Олександр Андрійович _____

Керівник:
доцент, кандидат технічних наук, доцент
Кублій Лариса Іванівна _____

Консультант:
доцент, кандидат технічних наук
Варава Іван Андрійович _____

Рецензент:
старший науковий співробітник, кандидат технічних наук,
професор кафедри інформаційних технологій та програмування
Інституту комп’ютерних технологій Університету “Україна”
Самарай Валерій Петрович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ — 2021 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

спеціальності 121 “Інженерія програмного забезпечення”

освітня програма “Інженерія програмного забезпечення розподілених систем”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр КОВАЛЬ
(підпис)
“___”_____2021р.

ЗАВДАННЯ
на дипломну роботу студенту
Козаку Олександру Андрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Автоматизована система адаптивної фільтрації гідроакустичних сигналів на основі машинного навчання
керівник роботи Кублій Лариса Іванівна, к.н.т., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від “_26_” травня 2021 р. № _1340-с_

2. Строк подання студентом роботи 09 червня 2021 р.

3. Вихідні дані до роботи Результатом роботи є автоматизована програмна система фільтрації гідроакустичних сигналів на основі машинного навчання, візуалізації спектрів та осцилограм кожного з 4 каналів сигналу, навчання автоенкодера визначати сигнал певної частоти з поданого набору.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Провести огляд сучасних методів фільтрації гідроакустичних сигналів. Реалізувати алгоритми обробки параметрів вхідного сигналу. Розробити користувацький інтерфейс. Запропонувати розв’язання задачі адаптивної фільтрації гідроакустичних сигналів на основі машинного навчання, з побудовою файлу навчальної вибірки і подальшою його обробкою. Розробити програмний модуль для візуалізації подання отриманих даних.

5. Перелік ілюстративного матеріалу Аналіз існуючих рішень, структура програмного продукту, діаграма прецедентів, архітектура системи, схема роботи

автоенкодера, засоби розробки, результати, приклади роботи, висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1. Задача розробки автоматизованої системи адаптивної фільтрації гідроакустичних сигналів на основі машинного навчання	Варава Іван Андрійович, доцент, кандидат технічних наук		

7. Дата видачі завдання “10” жовтня 2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	03.12.2020	
2.	Вивчення та аналіз задачі	01.03.2021	
3.	Розробка архітектури та загальної структури системи	12-19.04.2021	
4.	Розробка структур окремих підсистем	12-24.04.2021	
5.	Програмна реалізація системи	19.04.2021 — 07.05.2021	
6.	Оформлення пояснювальної записки	14.05.2021-24.05.2021	
7.	Захист програмного продукту	11.05.2021	
8.	Передзахист	25.05.2021	
9.	Захист	16.06.2021	

Студент _____ _____Олександр КОЗАК_____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Лариса КУБЛІЙ_____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Мета роботи — створення програмної системи для адаптивної фільтрації гідроакустичного сигналу. Для написання системи обрано мову програмування Python; графічний користувацький інтерфейс реалізованого за допомогою пакету Tkinter; для обробки даних використано бібліотеки статистичного напряму Seaborn, Numpy, Scipy; для візуалізації даних — бібліотека MatPlotLib.

Дипломна записка складається зі вступу, 5 розділів і висновків. Записка містить 89 сторінок, 21 рисунок, 21 посилання і 4 додатки.

Результати досліджень представлені на XIX Міжнародній науково-практичній конференції молодих вчених і студентів “Сучасні проблеми наукового забезпечення енергетики”, 20-23 квітня 2021 року.

Ключові слова: Python, автоенкодер, машинне навчання, аналіз даних, інтерпретована мова програмування, візуалізація, гідроакустика.

ABSTRACT

The purpose of the work is to create a software system for adaptive filtering of the sonar signal. Python programming language was chosen to write the system; graphical user interface implemented using the Tkinter package; statistical data libraries Seaborn, Numpy, Scipy were used for data processing; for data visualization — MatPlotLib library.

The diploma note consists of an introduction, 5 sections and conclusions. The note contains 89 pages, 21 figures, 21 references and 4 appendices.

The research results were presented at the XIX International Scientific and Practical Conference of Young Scientists and Students “Modern Problems of Scientific Support of Energy”, April 20-23, 2021.

Keywords: Python, autoencoder, machine learning, data analysis, visualization, hydroacoustics.

ЗМІСТ

ВСТУП

Гідроакустика має широке практичне застосування — її використовують для підводної локації, зв'язку, океанологічних досліджень, розв'язання військових задач. Звук дуже якісно поширюється у воді, тому його можна чути і виявляти на великих відстанях.

На сьогодні гідроакустика зіштовхується з проблемою розробки теорій, які описують поширення звуку в воді, та створення відповідних математичних моделей для пояснення експериментальних сигналів. Можливим розв'язанням цієї проблеми є створення автоматизованої системи адаптивної фільтрації гідроакустичних сигналів на основі машинного навчання. У даний час одним з основних інструментів, який використовується для обробки сигналів, є перетворення Фур'є. Проте він надає лише інформацію про частоту, присутню в сигналі, і не дає ніякої інформації про часовий інтервал, в якому існує частота. Застосування вейвлет-аналізу може частково розв'язати цю проблему, використовуючи різні конфігурації вейвлетів, які реалізують різні варіанти коефіцієнта невизначеності. Проблема фільтрації отриманих даних також дуже важлива.

Не дивлячись на потенціал даного напрямку, сьогодні у вільному і комерційному доступі не вистачає програмного забезпечення для виконання фільтрації та обробки гідроакустичних сигналів.

Метою дипломної роботи є створення програмної системи для відображення роботи адаптивної фільтрації гідроакустичних сигналів з використанням машинного навчання для автоматичного підбору автоенкодера для усунення шумів у сигналах.

Для розв'язання цих проблем створено зручний та інтуїтивно зрозумілий додаток, який можна використовувати для відображення осцилограм, спектрів і для адаптивного усунення шумів з усіх каналів гідроакустичного сигналу.

Результати роботи можуть бути використані гідроакустичними станціями виявлення сигналів, у навчальних закладах для поглиблення знань студентів про гідроакустику.

1. ЗАДАЧА РОЗРОБКИ АВТОМАТИЗОВАНОЇ СИСТЕМИ АДАПТИВНОЇ ФІЛЬТРАЦІЇ ГІДРОАКУСТИЧНИХ СИГНАЛІВ НА ОСНОВІ МАШИННОГО НАВЧАННЯ

В останні роки технологія машинного навчання швидко розвивалася і застосовувалася в галузі обробки зображень та аудіо-файлів. Основна сфера використання штучного інтелекту для створення комп'ютерів пов'язана з вивченням візуальних і звукових даних. Важливою є обробка великих обсягів даних у конкретному контексті.

Існує багато класичних методів розв'язання проблем аудіо-досліджень, але ці методи не завжди ефективні при збільшенні обсягу інформації. У той самий час сучасні методи, засновані на використанні нейронних мереж [1] на великих обсягах даних, покращують свою продуктивність, оскільки етапами побудови моделі є підбір навчальної вибірки і процес навчання.

Основне завдання дипломної роботи — створити програмну систему для адаптивного фільтрування гідроакустичних сигналів. Фільтрація водних звуків широко застосовується у пошуку і розпізнаванні об'єктів, підводних комунікаціях, навігації, дослідженні океану тощо. Швидкість поширення звуку у водному середовищі не є постійною величиною, вона залежить від різних факторів.

За допомогою математичних моделей важко розглянути вплив багатьох факторів. Тому використання машинного навчання для фільтрації акустичних сигналів у водних середовищах є важливим напрямом досліджень.

Завданнями дипломної роботи є:

— створення модулів для зчитування і запису даних;

— розділення вхідних даних на 4 канали;

— створення компонентів для обробки й аналізу гідроакустичних сигналів;

— створення моделі машинного навчання для адаптивної фільтрації каналів гідроакустичних сигналів;

— створення інтерфейсу для взаємодії з програмним продуктом.

Завдяки використанню сучасних технологій і методів розробки програмного забезпечення програмна система має простий при використанні інтерфейс, може швидко працювати, незалежно від операційної системи.

1.1. Сфери застосування

Результати роботи можуть бути використані гідроакустичними станціями виявлення сигналів, що входять до складу гідроакустичних комплексів вимірювання і аналізу параметрів сигналів, метою яких є виявлення джерела випромінювання і його класифікація.

Розироблена система може бути використана як для гідроакустичних досліджень, так і з навчальною метою. Компоненти системи ввійдуть в спеціалізований програмний комплекс.

Використання штучного інтелекту при обробці гідроакустичних даних сприятиме інформаційному забезпеченню операторів-гідроакустиків і командирів підводних човнів не тільки координатами підводних об’єктів, але і наданням повної картини бойового простору для оптимальних дій у відповідь.

1.2. Навчання автоенкодера

Автоенкодери [2] — це нейронні мережі прямого поширення, які відновлюють вхідний сигнал на виході. Всередині у них є прихований шар, який представляє собою код, що описує модель. Автоенкодери конструюються таким чином, щоб не мати можливості точно скопіювати вхід на виході. Вхідний сигнал відновлюється з помилками через втрати при кодуванні, але, щоб їх мінімізувати, мережа змушена вчитися відбирати найбільш важливі ознаки.

Автоенкодери складаються з двох частин (рисунок 1.1): енкодера і декодера. Енкодер перетворює вхідний сигнал в його уявлення (код), а декодер відновлює сигнал за його кодом.

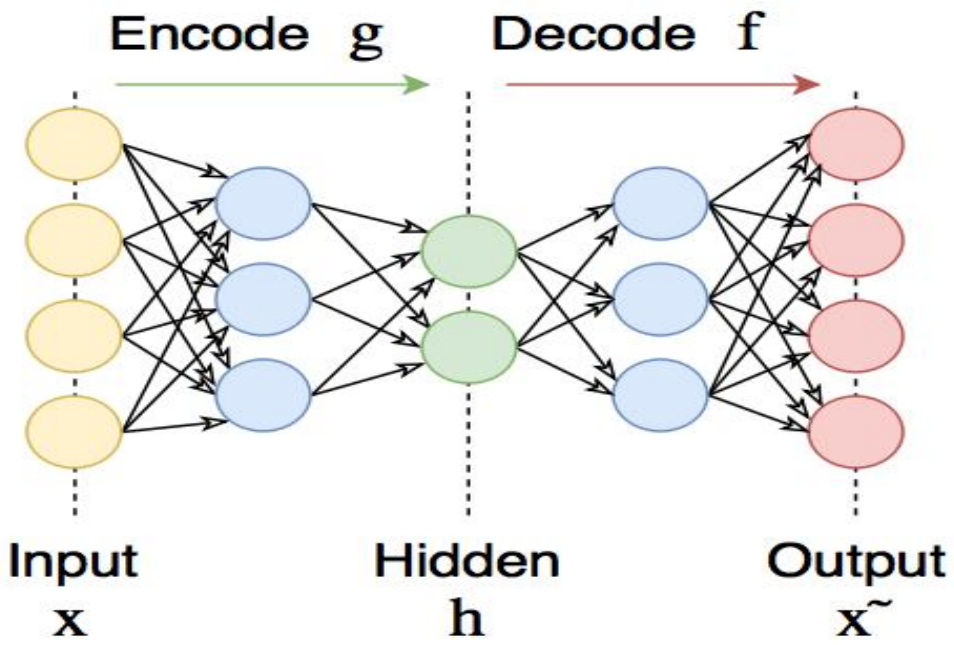

Рисунок 1.1 — Схема роботи нейронного автоенкодера [2]

При цьому сімейства функцій енкодера і декодера обмежені, щоб автоенкодер був змушений відбирати найбільш важливі властивості сигналу.

Сама по собі здатність автоенкодерів стискати дані використовується рідко, оскільки, як правило, вони працюють гірше, ніж вручну написані алгоритми для конкретних типів даних на зразок звуків або зображень. Також для них критично важливо, щоб дані належали тій генеральної сукупності, на якій мережа навчалася. Навчивши автоенкодер на цифрах, його не можна застосовувати для кодування чогось іншого (наприклад, зображень людей).

Автоенкодери можна навчити прибирати шум з даних — для цього треба на вхід подавати зашумлені дані і на виході порівнювати з даними без шуму. Різниця сигнал/шум до і після обробки сигналу і буде результатом ефективності автоенкодера. У програмній системі адаптивної фільтрації гідроакустичних сигналів є вивід значень сигнал/шум до фільтрації та після і, відповідно, видно у скільки разів сигнал було знешумлено; середнє значення для програмної системи, написаної в рамках дипломної роботи, є ≈ 2, що вважається хорошим показником для знешумлення гідроакустичного сигналу.

1.3. Відображення осцилограм каналів сигналу

Осцилограма [3] — крива електричного (або перетвореного на електричний) процесу, записана осцилографом. Під інтенсивністю, або силою, звуку звичайно розуміють сумарну інтенсивність усіх складових даного сигналу; її вимірюють за допомогою так званої кривої інтенсивності на осцилограмі.

При роботі зі складною радіоелектронною апаратурою часто виникає завдання відтворення миттєвого значення напруги залежно від часу. Її розв'язання дає можливість відразу оцінити багато параметрів коливань, наприклад, спотворення їхньої форми, наявність перешкод і багато іншого. Відтворення форми сигналів відіграє важливу роль при перевірці та налаштуванні аудіо- й відеотракту апаратури.

Для візуалізації сигналів використовують прилади, які називають осцилографами.

У розробленій програмній системі візуалізація осцилограм 4-ох каналів вхідного і вихідного сигналів відображається за допомогою використання функцій бібліотеки Seaborn мови Python.

Бібліотека створення статистичних графіків Seaborn — одна з найбільш використовуваних бібліотек візуалізації даних в Python. Бібліотека пропонує простий, інтуїтивно зрозумілий прикладний програмний інтерфейс (API) для візуалізації даних, який можна налаштовувати. Приклади використання бібліотеки в програмній системі подано на рисунках 1.2 і 1.3.

Відображення осцилограм 4-ох каналів (три канали проекції коливальної швидкості v_x, v_y, v_z і 1 канал тиску) вхідного сигналу зображено на рисунку 1.2.

Візуалізація спектрів відбувається до і після обробки dat-файлів і створюється для всіх 4-ох каналів сигналу. Візуалізація має вигляд 2D-графіка, який відображає залежність амплітуди від часу.

Відображення осцилограм 4-ох каналів фільтрованого сигналу подано на рисунку 1.3.

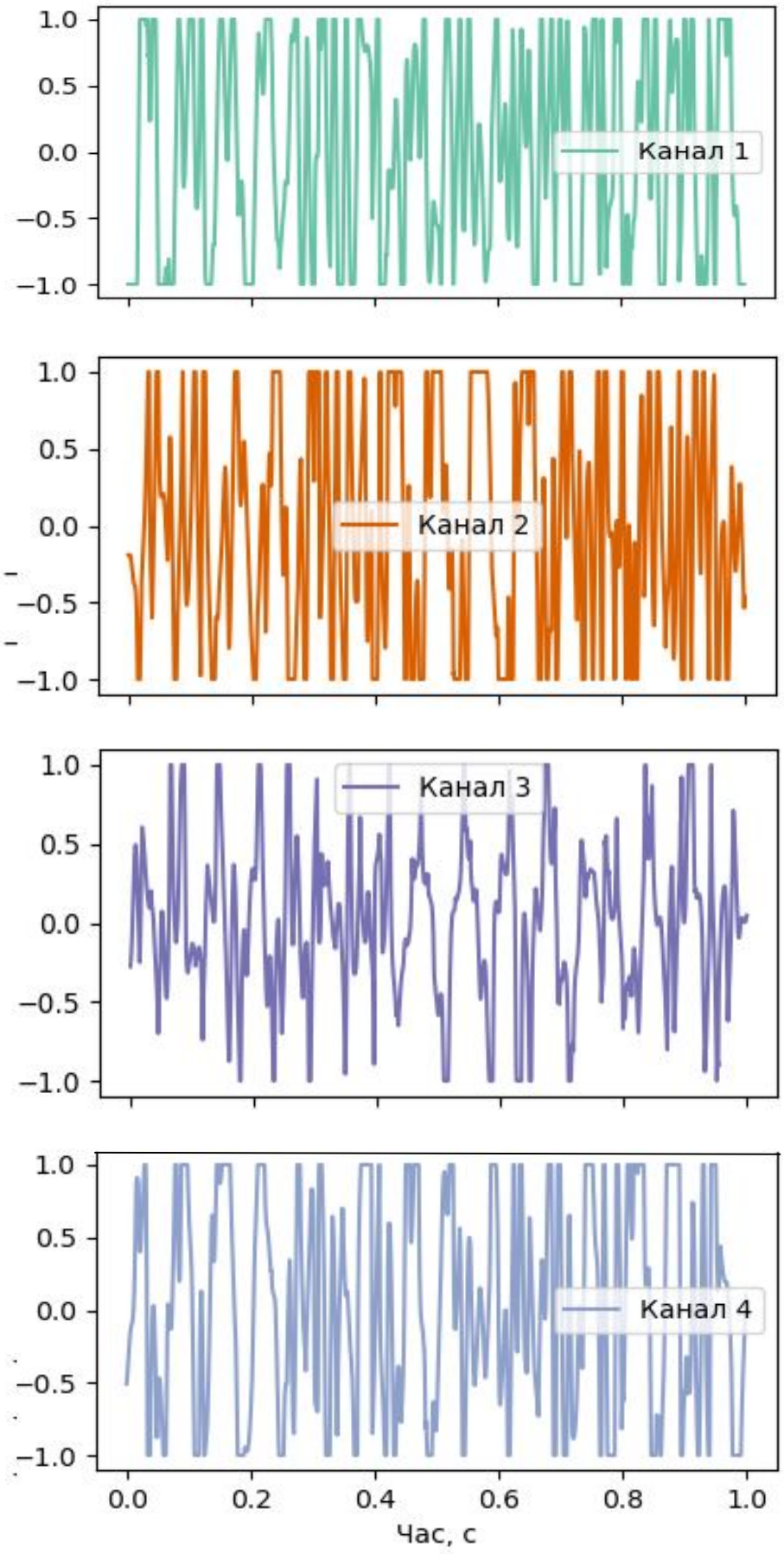

Рисунок 1.2 — Відображення осцилограм вхідного файлу

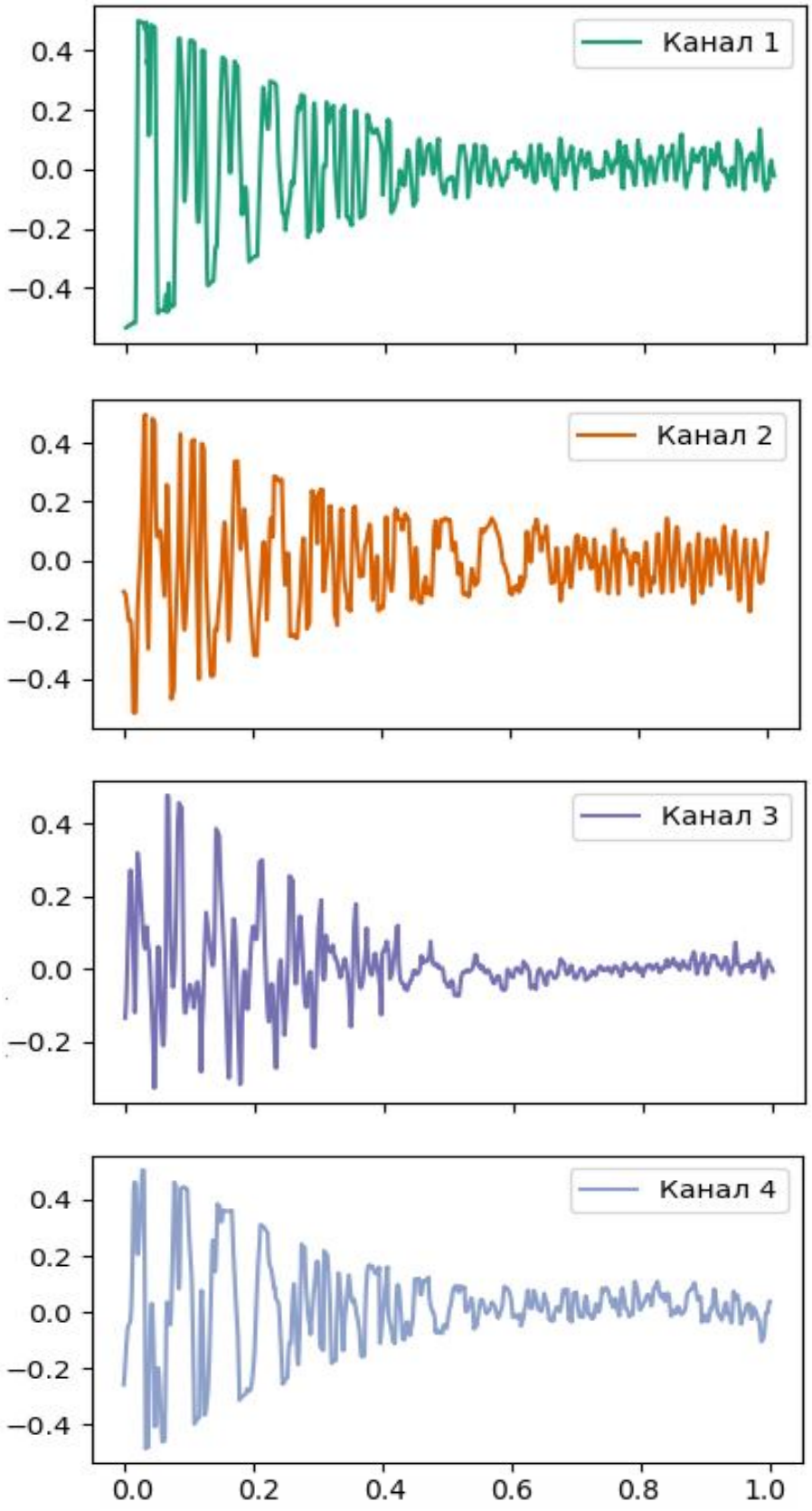

Рисунок 1.3 — Відображення осцилограм вихідного файлу

Програмну систему легко розширити, тому на її основі можна розробляти різні додатки.

2. ОГЛЯД ПРОБЛЕМ АДАПТИВНОЇ ФІЛЬТРАЦІЇ СИГНАЛІВ

Для створення високоякісних програмних продуктів необхідно врахувати існуючі типи сигналів, механізми фільтрації, обробки та зберігання гідроакустичних сигналів. Дослідити, які проблеми можуть виникнути в різних ситуаціях, розглянути іймовірні варіанти розв’язання перешкод. Також слід розглянути і проаналізувати наявні аналоги.

2.1. Види сигналів

Сигнал визначається як напруга або струм, який може бути переданий як повідомлення або як інформація. За своєю природою всі сигнали є аналоговими, незалежно, чи це сигнал постійного або змінного струму, цифровий або імпульсний.

Сигнал [4] — зміна фізичної величини (наприклад, температури, тиску повітря, світлового потоку, сили струму тощо), що використовується для пересилання даних. Саме завдяки цій зміні сигнал може нести в собі якусь інформацію.

Розрізняють електричні сигнали (зміна напруги або сили струму), акустичні (зміна частоти звуку), оптичні (зміна інтенсивності або кольору світла), а також аналогові та цифрові сигнали, які використовуються в аналоговій та цифровій техніці.

Класифікація сигналів за фізичною природою носія інформації:

— механічний сигнал — сигнал у вигляді механічної дії твердого тіла, у якого дієвою величиною є сила, момент сили або переміщення;

— електричний сигнал — сигнал у вигляді електричної дії, дієвою величиною якого є сила струму або напруга;

— радіосигнал — сигнал у вигляді дії електромагнітного випромінювання,

дієвою величиною якого є напруженість електричного поля або магнітного поля;

— оптичний сигнал — сигнал у вигляді дії оптичного випромінювання, дієвою величиною якого є потік випромінювання;

— акустичний сигнал — сигнал у вигляді дії звуку, дієвою величиною якого є звуковий тиск;

— гідравлічний (пневматичний) сигнал — сигнал у вигляді механічної дії рідини (газу), дієвою величиною якого є тиск.

2.2. Методи фільтрації сигналів

Фільтрація є однією з найпоширеніших операцій обробки сигналів. Метою фільтрації є усунення перешкод, які містяться в сигналі, або виявлення різних компонентів сигналу, які відповідають певним характеристикам досліджуваного процесу.

У технології цифрової обробки сигналів використовується все більше цифрових фільтрів (ЦФ). За допомогою ЦФ певні частоти в сигналі виділяються, перериваються або послаблюються для поліпшення відношення сигнал/шум. Крім того, ЦФ широко використовують як структурний елемент обладнання та систем обробки сигналів. Тому ЦФ мають важливе практичне значення.

Сучасні цифрові фільтри — це обладнання, яке використовують у різних технічних галузях. Останнім часом методи фільтрації, присвячені вимірюванню та розрахунку, всебічно розвиваються. Ці методи засновані на реалізації спеціальних вагових функцій.

Фільтрацію сигналу, тобто зміну його спектра, як правило, здійснюють з метою збільшення відношення сигнал/шум, зменшення впливу завад або виділення (підсилення) якої-небудь корисної якості сигналу. Наприклад, для вимірювання сигналів, отриманих від термопар, найчастіше доводиться застосовувати фільтри, що послаблюють завади мережі. Вихідний корисний сигнал термопар становить, як правило, декілька мілівольт і завада від силової мережі може бути порівнянна з корисним сигналом або навіть перевищувати його. Другим прикладом фільтрації є

фільтрування сигналу, одержаного від давача моменту, який розвиває двигун деякого транспортного засобу. Виділяючи за допомогою фільтра постійну складову цього сигналу, отримують інформацію про середню потужність двигуна. Якщо виділити і проаналізувати високочастотні складові сигналу, то можна зробити висновок про якість системи регулювання, а також про вібрацію, зумовлену роботою двигуна.

2.3. Перетворення Фур'є

Перетворення Фур'є [5] — інтегральне перетворення однієї комплексно значної функції дійсної змінної на іншу. Тісно пов'язане з перетворенням Лапласа та аналогічне розкладу в ряд Фур'є для неперіодичних функцій. Це перетворення розкладає дану функцію на осциляторні функції. Використовується для того, щоб розрахувати спектр частот для сигналів, змінних у часі. Перетворення названо на честь французького математика Жана Батиста Жозефа Фур'є, який ввів поняття в 1822 році.

Дискретне перетворення Фур'є (ДПФ) є базовим алгоритмом цифрової обробки сигналів у частотній області. Завдяки наявності ефективних алгоритмів його обчислення — алгоритмів швидкого перетворення Фур'є (ШПФ) — ДПФ широко використовують для цифрової фільтрації та спектрально-кореляційного аналізу сигналів.

Обчислення перетворень Фур'є вимагає дуже великої кількості множень (складність алгоритму $O(N^2)$) і обчислень синусів. Існує спосіб виконати ці перетворення значно швидше з обчислювальною складністю $O(N\log_2 N)$. Цей спосіб називається швидким перетворенням Фур'є. Алгоритм ШПФ — це спосіб швидкого обчислення ДПФ:

$$X(jk) = \sum_{n=0}^{n-1} x(n) e^{-j\left(\frac{2\pi}{N}\right)kn},$$

що дає можливість усунути притаманну ДПФ надмірність. Вони ґрунтуються на властивостях комплексної експоненти $e^{-j\left(\frac{2\pi}{N}\right)kn}$, яку для зручності позначають

W_N^{kn} ($W_N^{kn} = e^{-j\left(\frac{2\pi}{N}\right)kn}$), її симетрії $W_N^{(N-k)n} = W_N^{(N-n)k} = \left(W_N^{kn}\right)^*$ і періодичності $W_N^{(N+k)(N+n)} = W_N^{kn}$ з періодом, рівним довжині оброблюваної реалізації сигналу N (кількості точок ШПФ). Відповідно до останньої властивості експоненті $W_N^{pkn} = W_{N/p}^{kn}$ відповідає період N/p, де p — цілі числа, на які ділиться N. Використання даних властивостей в алгоритмах ШПФ виключає велику кількість повторюваних при обчисленні ДПФ операцій.

У результаті швидкодія ШПФ може залежно від N в сотні разів перевершувати швидкодію стандартного алгоритму.

При цьому слід підкреслити, що алгоритм ШПФ є точним. Він навіть точніше стандартного, тому що, скорочуючи кількість операцій, він призводить до менших помилок заокруглення.

2.4. Існуючі програмні засоби розв'язання задач адаптивної фільтрації гідроакустичних сигналів

На сьогоднішній день існує багато програм для фільтрації сигналів. Серед них найбільш відомі: Фільтрація Сигналів від ZETLAB, програма цифрової обробки сигналів DSP, програма обробки сигналів SPTool.

Програму Фільтрація Сигналів від ZETLAB [6] використовують для фільтрації сигналів, які надходять на вхідні канали аналізаторів спектра, тензометричних станцій, сейсмостанцій для подальшої обробки програмами ZETLAB. Програма також може застосовуватися для обробки віртуальних каналів, створених такими програмами, як Віброметр, Тензодатчик та ін. Програма Фільтрація Сигналів працює з сигналами як в режимі реального часу, так і при відтворенні записаних сигналів.

При проведенні різних видів випробувань, вимірювань, діагностики і розпізнавання мови в складних умовах навколишніх перешкод з'являється проблема достовірного оцінювання будь-якого параметра сигналу, наприклад, рівня, частоти, коефіцієнта кореляції з іншим сигналом. Якщо корисний сигнал і перешкода

поділяються в частотній області, то найпоширенішим методом є метод фільтрації сигналів.

Цифрова обробка сигналу використовується майже у всіх фірмових апаратах. Цифрова фільтрація істотно покращує умови прийому сигналу у зашумленому середовищі.

Принцип DSP [7] (Digital signal processing) — перетворити аналоговий сигнал, який підлягає фільтрації, за допомогою аналого-цифровий перетворювач в потік даних і за спеціальними алгоритмами цифрової фільтрації виконати його обробку. Потім, подавши оброблений сигнал на цифро-аналоговий перетворювач, виконати зворотне перетворення сигналу в аналоговий, який вже не містить непотрібних частотних складових. Принцип DSP дає можливість створити фільтр, швидко змінити його тип і параметри, забезпечити стабільність характеристик, кореляційну обробку, недоступні аналоговим фільтрам.

Можна вказати два основні недоліки програми DSP.

1. Загальний для всіх DSP-програм недолік — затримка сигналу при обробці приблизно на 0,5 с (залежить від продуктивності комп'ютера, смуги і добротності фільтру). Тому при налаштуванні, особливо з вузькосмуговим фільтром, можливі ефекти запізнювання. Або налаштовуватися при вимкненому DSP, а потім, визначившись з прийнятою станцією, його ввімкнути. При передачі безперервного сигналу слухати самого себе при ввімкнутому DSP не можливо, оскільки відбувається затримка сигналу в півсекунди. Тому для нормальної роботи з DSP треба зробити найпростіший релейний комутатор, який за сигналом з трансивера перемикає навушники з виходу звукової карти безпосередньо на вихід трансивера. Цим самим комутатором зручно відключати DSP.

2. У DSP не можна одночасно ввімкнути кілька фільтрів. Може працювати або смуговий, або кореляційний, або режекторний.

Програма обробки сигналів SPTool [8] має власні засоби графічного інтерфейсу користувача (GUI), які забезпечують не тільки графічне подання сигналів і характеристик фільтрів, але і візуально-орієнтоване, і інтерактивне керування створенням сигналів і пристроїв фільтрування. До засобів GUI

відноситься програма обробки сигналів SPTool (Signal Processing Tool). Наявність цієї програми дає можливість у деяких випадках відмовитися від прямого програмування задач обробки сигналів і проектування фільтрів і навіть від використання функцій командного режиму роботи.

Програма обробки сигналів SPTool дає можливість виконувати такі операції:

— імпортувати сигнали з .mat-файлів або робочої області MATLAB;

— переглядати графіки сигналів (у тому числі кількох одночасно);

— застосовувати до сигналів різні методи спектрального аналізу і переглядати отримані графіки;

— розраховувати дискретні фільтри з використанням функцій пакета (в тому числі шляхом прямого редагування розташування нулів і полюсів);

— пропускати сигнали через фільтри і аналізувати отримувані вихідні сигнали.

Щоб обробляти будь-які сигнали за допомогою SPTool, передусім необхідно сформувати ці сигнали в MATLAB, а потім імпортувати отримані вектори значень цих сигналів в програму SPTool.

Слід зазначити, що в частині аналізу та синтезу фільтрів можливості програми SPTool є досить незначними.

3. АНАЛІЗ ТА ОБҐРУНТУВАННЯ ВИКОРИСТАННЯ ЗАСОБІВ РЕАЛІЗАЦІЇ

Для написання програмної системи фільтрації сигналів використано мову програмування Python. Розробка програми відбувалася в командній оболонці для інтерактивних обчислень Jupyter Notebook [9]. Графічний інтерфейс створено в середовищі розробки Visual Studio Code.

3.1. Середовище розробки Visual Studio Code

Середовище Visual Studio Code [10] — безкоштовний популярний редактор коду від Microsoft.

Редактор розраховано на розробників різних рівнів. З одного боку він підходить новачкам, оскільки його інтерфейс інтуїтивно простий і зрозумілий. З іншого боку в VS Code вбудовано багато можливостей, цікавих досвідченим розробникам.

Редактор коду Visual Studio Code відрізняє від конкурентів наявність вбудованого компілятора (налагоджувача). Після невеликого налаштування можна шукати помилки в коді прямо з редактора, наприклад, поставити точку зупинки і спостерігати за виконанням конкретної ділянки коду. Крім цього, у редактора є вбудована консоль, в яку може виводитися результат роботи або повідомлення про помилку, якщо щось пішло не так. Компілятор (налагоджувач) можна налаштувати під різні мови і різні завдання.

Особливостями Visual Studio Code є:

— редактор VS Code дає можливість розробляти як консольні додатки, так і додатки з графічним інтерфейсом, у тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-додатки, веб-служби як в рідному, так і в керованому кодах для всіх платформ;

— у редакторі присутні вбудований налагоджувач, інструменти для роботи з Git і засоби рефакторингу, навігації по коду, автодоповнення типових конструкцій і контекстної підказки;

— продукт підтримує розробку на платформах ASP.NET і Node.js і вважається легким рішенням, яке дає можливість обійтися без інтегрованого середовища розробки;

— великим плюсом редактора є підтримка великої кількості мов, таких як C++, C#, Python, PHP, JavaScript та інших.

Редактор коду Visual Studio Code надає користувачеві ряд можливостей:

— вбудовані інструменти інтеграції з GitHub, GIT, а також Visual Studio Team Services для швидкого тестування, складання, упаковки і розгортання різних типів додатків;

— зручність роботи з Unity-проектами;

— робота з Mono і Node.js за допомогою вбудованого налагоджувача;

— підтримка TypeScript і JavaScript;

— публікація створених додатків в Microsoft Azure через сервіс Visual Studio Team Services;

— підтримка практично всіх мов програмування;

— написання коду для конкретного завдання з його подальшою інтеграцією в проект (з надбудовою або безпосередньо);

— велика бібліотека шаблонів, готових фрагментів коду і сніпетів з можливістю додавання своїх елементів;

— одночасна робота з кількома проектами (у кількох вікнах);

— інтерфейс можна розділити на дві панелі для порівняння коду;

— функція налагодження.

3.2. Мова програмування Python

Мова програмування Python [11] проста для використання та, водночас, повноцінна мова програмування, яка надає багато засобів для структурування і

підтримки великих програм. Вона краще за мову С обробляє помилки і, будучи мовою дуже високого рівня, має вбудовані типи даних високого рівня, такі як гнучкі масиви і словники, ефективна реалізація яких мовою С потребує значних затрат часу.

Завдяки більш загальним типам даних, Python застосовують до більш широкого кола задач, ніж Awk і навіть Perl.

Високорівнева мова програмування Python дає можливість розбивати програми на модулі, які потім можуть бути використані в інших програмах. Мова програмування Python поставляється з великою бібліотекою стандартних модулів, які можна використовувати як основу для нових програм або як приклади при вивченні мови. Стандартні модулі надають засоби для роботи з файлами, системними викликами, мережними з'єднаннями і навіть інтерфейсами до різних графічних бібліотек.

Інтерпретована мова Python дає можливість заощадити значну кількість часу, що, як правило, витрачається на компіляцію. Інтерпретатор можна використовувати інтерактивно, що дає можливість експериментувати з засобами мови, писати шаблони програм або тестувати функції при розробці "знизу-вгору". Мова Python дає можливість писати дуже компактні й зручні для читання програми. Програми, написані мовою Python, звичайно значно коротші від еквівалентів, написаних мовами С або С++ з кількох причин:

— типи даних високого рівня дають можливість подати складні операції однією інструкцією;

— групування інструкцій виконується за допомогою відступів замість фігурних дужок;

— немає необхідності оголошувати змінні.

Мова Python — розширювана мова: знання мови С дає можливість додавати нові функції, що вбудовуються, або модулі для виконання критичних операцій з максимальною швидкістю. Інтерпретатор мови Python може бути вбудований у програму, написану мовою С, і використовуватися як розширення або команда для цієї програми. Мова програмування Python використовується в даний час десятками

тисяч програмістів у всьому світі, і кількість людей, які використовують її, швидко зростає, подвоюється і потроюється щороку. Мова Python приваблює користувачів з багатьох причин. Вона використовується для розробки програм і дає можливість провести розробку набагато швидше, ніж традиційні мови типу C, C++ або Java. Ця мова працює однаково добре на Windows, UNIX, Macintosh і OS/2, може використовуватися для легкої розробки як малих додатків чи сценаріїв, так і для розгортання великих програм.

Мова Python пропонує доступ до потужного і легкого при використанні комплекту 29 інструментальних засобів графічного інтерфейсу користувача. Традиційні машинні мови типу C і Pascal мають ряд характеристик, наприклад, сувора типізація, базові типи, складні (і звичайно довгі) цикли, потреба у великих кількостях кодів для виконання відносно малих задач. Мова програмування Java досить нова, але має більшість характеристик, включених у цей перелік. Відсутність суворої типізації полегшує роботу з Python.

Відмінностей Python від інших мов досить багато. Основні з них:

— керування пам’яттю — цілком автоматичне — не потрібно хвилюватися щодо розподілу або звільнення пам’яті. Немає загрози “небезпечного посилання”. Мова програмування Java — єдина мова, яка пропонує таку концепцію;

— типи зв’язані з об’єктами, а не зі змінними. Це означає, що змінній може бути призначене значення будь-якого типу, і що, наприклад, масив може містити об’єкти різних типів. Традиційні мови не надають такої можливості;

— операції звичайно виконуються в більш високому рівні абстракції. Це частково результат того, як написана мова, і частково результат розширеної стандартної бібліотеки кодів, яка постачається разом з Python.

Ці та інші особливості Python роблять розгортання додатків надзвичайно швидким. Продуктивність створеного додатку залежить від його особливостей. Звичайно, для чисельного алгоритму, який виконує звичайну арифметику цілого числа в циклі for, неважливо, якою мовою він написаний. Але для “середнього” додатка збільшення продуктивності може бути досить значним.

Один недолік Python порівняно з найбільш традиційними мовами полягає в

тому, що це — не цілком компільована мова; замість цього, вона частково транслює програму до внутрішньої форми байт-коду і цей байт-код виконується інтерпретатором Python. Проте у перспективі сучасні комп'ютери мають так багато невикористовуваного обчислювального потенціалу, що для 90% додатків швидкодія пов'язана з вибором мови. Мова програмування Java теж компілюється в байт-код, але в даний час працює повільніше, ніж Python у більшості випадків. Крім того, дуже просто об'єднати Python з модулями, написаними мовами C або C++, які можна використовувати, щоб збільшити швидкість роботи програм у критичних ділянках.

3.3. Платформа Anaconda

Платформа Anaconda [12] — це дистрибутиви Python. Платформа надає все необхідне для розв'язання завдань щодо аналізу та обробки даних.

Дистрибутив Anaconda — це набір бінарних систем, який включає в себе інструменти Scipy, Numpy, Pandas і їхні залежності.

— бібліотека Scipy — це пакет статистичного аналізу;

— бібліотека Numpy — це пакет числових обчислень.

— бібліотека Pandas — рівень абстракції даних для об'єднання і перетворення даних.

Дистрибутив Anaconda корисна тим, що об'єднує все це в єдину систему. Двійкова система Anaconda — це установник, який збирає всі пакети з залежностями всередині системи.

Дистрибутив Anaconda включає в себе менеджера пакетів Conda. Крос-платформний менеджер пакетів Conda написано чистою мовою Python, що полегшує його використання в віртуальних середовищах Python. Крім того, Conda підходить для бібліотек C, пакетів R, Java тощо. Менеджер пакетів Conda встановлює виконавчі системи. Інструмент conda build створює пакети з вихідного коду, а conda install виконує установку з пакетів збірки Conda.

3.4. Система керування середовищем Conda

Система керування середовищем Conda [13] — це менеджер пакетів з відкритим кодом, який працює в операційних системах Windows, macOS і Linux. Система керування Conda проста щодо установки, виконання та оновлення пакетів і залежностей. Вона задумувалася для програм мовою Python, але може створювати пакети і дистрибутиви програмного забезпечення будь-якою мовою.

Система Conda як менеджер пакетів допомагає знаходитити і встановлювати пакети. Якщо потрібен пакет, для якого потрібна інша версія Python, то не потрібно перемикатися на інший менеджер середовища, оскільки Conda також є менеджером середовища. За допомогою лише кількох команд можна налаштувати абсолютно окреме середовище для запуску цієї іншою версією Python, продовжуючи при цьому запускати звичайну версію Python у звичайному середовищі.

У конфігурації за замовчуванням Conda може встановлювати і керувати тисячами пакетів на repo.anaconda.com, які створюються, перевіряються і підтримуються Anaconda.

Система Conda може бути об’єднана з системами безперервної інтеграції, такими як Travis CI і AppVeyor, щоб забезпечити часте автоматичне тестування коду.

Пакет Conda і менеджер середовища включені в усі версії Anaconda і Miniconda.

Система керування середовищем Conda також включена в Anaconda Enterprise, яка забезпечує керування корпоративними пакетами і середовищами для Python, R, Node.js, Java і інших стеків додатків.

3.5. Бібліотека NumPy

Бібліотека NumPy [14] — бібліотека мови Python, яка надає підтримку великих багатовимірних масивів і матриць разом з великою бібліотекою

високорівневих і дуже швидких математичних функцій для операцій з цими масивами.

Використовуючи NumPy, розробник може виконувати такі операції:

— математичні і логічні операції над масивами.

— перетворення Фур'є і процедури для маніпулювання з формою;

— операції, пов'язані з лінійною алгеброю. Бібліотека NumPy має вбудовані функції для лінійної алгебри та генерації випадкових чисел.

Бібліотекау NumPy часто використовують разом з такими пакетами, як SciPy (Scientific Python) і Mat-plotlib (бібліотека креслення). Ця комбінація широко використовується в якості заміни MatLab, популярної платформи для технічних обчислень.

Стандартний дистрибутив Python не поставляється в комплекті з модулем NumPy. Полегшеною альтернативою є установка NumPy за допомогою популярного установника пакетів Python — pip.

У програмній системі застосування функцій бібліотеки NumPy використовується, зокрема, для реалізації функції обчислення співвідношення сигнал/шум:

3.6. Бібліотека Seaborn

Бібліотека Seaborn [15] — це бібліотека візуалізації даних Python, заснована на бібліотеці для візуалізації даних двовимірною 2D-графікою matplotlib. Бібліотека візуалізації даних Seaborn надає високорівневий інтерфейс для малювання привабливих і інформативних статистичних графіків. Бібліотека Seaborn допомагає вивчити і зрозуміти оброблювані дані. Її функції побудови графіків працюють з фреймами даних і масивами, що містять цілі набори даних, і внутрішньо виконують необхідне семантичне відображення і статистичну агрегацію для створення інформативних графіків. Її декларативний API, орієнтований на набір даних, дає можливість зосередитися на тому, що означають різні елементи графіків, а не на деталях їхнього малювання.

Бібліотека Seaborn використовує бібліотеку matplotlib для малювання своїх графіків. Для інтерактивної роботи рекомендується використовувати інтерфейс Jupyter у режимі matplotlib.

Завдяки застосуванню модулів і функцій бібліотеки Seaborn відображаються осцилограми 4-ох каналів сигналу.

3.7. Бібліотека Matplotlib

Бібліотека Matplotlib[16] — це бібіліотека двовимірної графіки для мови програмування Python, за допомогою якої можна створювати високоякісні малюнки різних форматів. Бібліотека Matplotlib містить багато модулів. Модулі наповнені різними класами і функціями, ієрархічно пов’язаними між собою.

Створення малюнка в Matplotlib схоже з малюванням у реальному житті. Так, художнику потрібно взяти основу (полотно або папір), інструменти (пензлі або олівці), мати уявлення про майбутнє малюнка (що саме він буде малювати) і, нарешті, виконати все це і намалювати малюнок деталь за деталлю.

У Matplotlib всі ці етапи також існують, і в якості художника-виконавця тут виступає сама бібліотека. Від користувача потрібно керувати діями художника-matplotlib, визначаючи, що саме він має намалювати і якими інструментами. Як правило, створення основи і процес безпосереднього відображення малюнка виконує Matplotlib. Таким чином, користувач бібліотеки Matplotlib виступає в ролі керівника. І чим простіше йому керувати кінцевим результатом роботи matplotlib, тим краще.

Оскільки Matplotlib організована ієрархічно, а найбільш простими для розуміння людиною є саме високорівневі функції, то знайомство з Matplotlib починають з самого високорівневого інтерфейсу matplotlib.pyplot. Так, щоб намалювати гістограму за допомогою цього модуля, потрібно виконати всього одну команду: matplotlib.pyplot.hist (arr).

Інтерфейс matplotlib.pyplot є набором команд і функцій, які роблять синтаксис графічних matploltib-команд схожим на команди, які використовуються в середовищі MATLAB. Спочатку matplotlib планувався як вільна альтернатива

MATLAB, де в одному середовищі були б засоби як для малювання, так і для чисельного аналізу. Так в Matplotlib з'явився pylab, який об'єднує модулі pyplot і numpy в один простір імен.

3.8. Модуль Pathlib

Модуль Pathlib в Python спрощує роботу з файлами і папками. Він доступний в Python 3.4 і пізніших версіях. Модуль Pathlib поєднує в собі найкраще з модулів файлової системи Python — os, os.path, glob тощо.

У мові програмування Python більшість скриптів передбачає роботу з файловими системами, а отже, взаємодію з назвами файлів і шляхами. Саме для цього в Python є модуль Pathlib, який містить корисні функції для виконання завдань, пов'язаних з файлами. Модуль Pathlib надає зручний для читання і простий спосіб створення шляхів, подаючи шлях файлової системи у вигляді належних об'єктів. Модуль дає можливість створювати код, який можна переносити між платформами.

3.9. Бібліотека SciPy

Бібліотека SciPy [17] — це бібліотека Python з відкритим вихідним кодом, призначена для розв'язання наукових і математичних проблем. Вона побудована на базі NumPy і дає можливість керувати даними, а також візуалізувати їх за допомогою різних високорівневих команд. Якщо імпортується SciPy, то NumPy окремо імпортувати не потрібно.

Бібліотеки NumPy і SciPy є бібліотеками Python, які використовуються для математичного і числового аналізів. Бібліотека NumPy містить дані масивів і операції, такі як сортування, індексація, а SciPy складається з числового коду. І хоча в NumPy є функції для роботи з лінійної алгеброю, перетвореннями Фур'є тощо, в SciPy вони подані в повному вигляді разом з багатьма іншими функціями. Для повноцінного наукового аналізу в Python потрібно встановлювати NumPy і SciPy, оскільки остання бібліотека побудована на базі NumPy.

Результат використання у програмі бібліотек SciPy, matplotlib і Pathlib для побудови та налаштування відтворення спектрограм подано на рисунку 3.1.

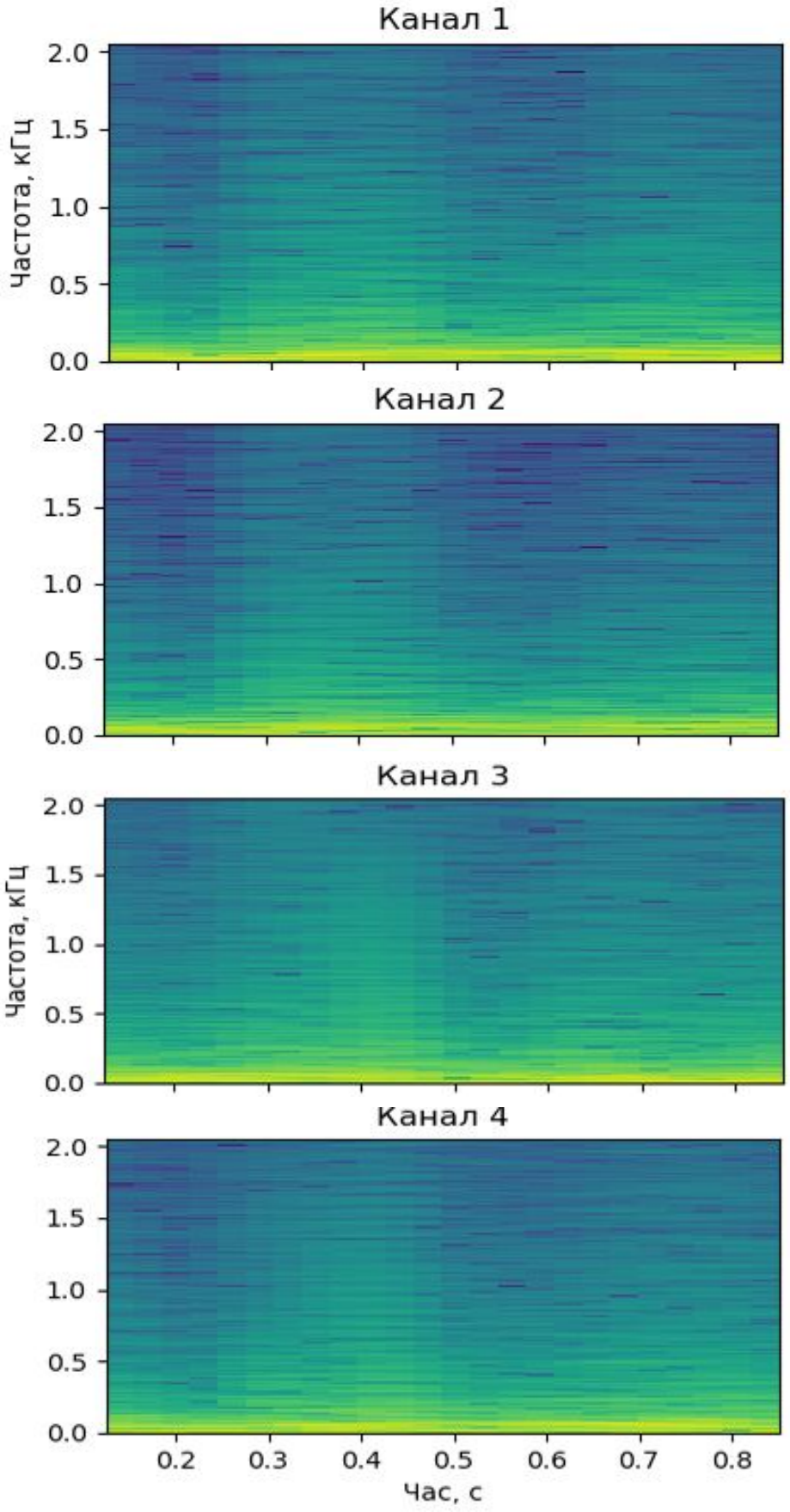

Рисунок 3.1 — Спектрограми 4-ох каналів вхідного файлу

Застосування цих бібліотек та їхніх функцій дало можливість виводити детальну інформацію про кожен канал гідроакустичного сигналу до та після фільтрації сигналу.

3.10. Модуль Noisereduce

Для знешумлення каналів вхідного dat-файлу використовується модуль noisereduce [18]. Код цього модулю відкритий і доступний на Github [19]. Даний модуль було оптимізовано під розроблену програмну систему. Зокрема було розв’язано проблему виведення кроків роботи фільтру, тобто вивід окремо шуму та сигналу до фільтрації, а також адаптовано під роботу з .dat-файлами та іншими компонентами програмної системи.

У програмній системі даний модуль застосовуєся для кожного каналу сигналу окремо, саме в цьому і полягає адаптивність програмної системи. Наприклад, результат використання даного модуля для фільтрації 1-го каналу .dat-файлу подано на рисунках 4.2-4.4.

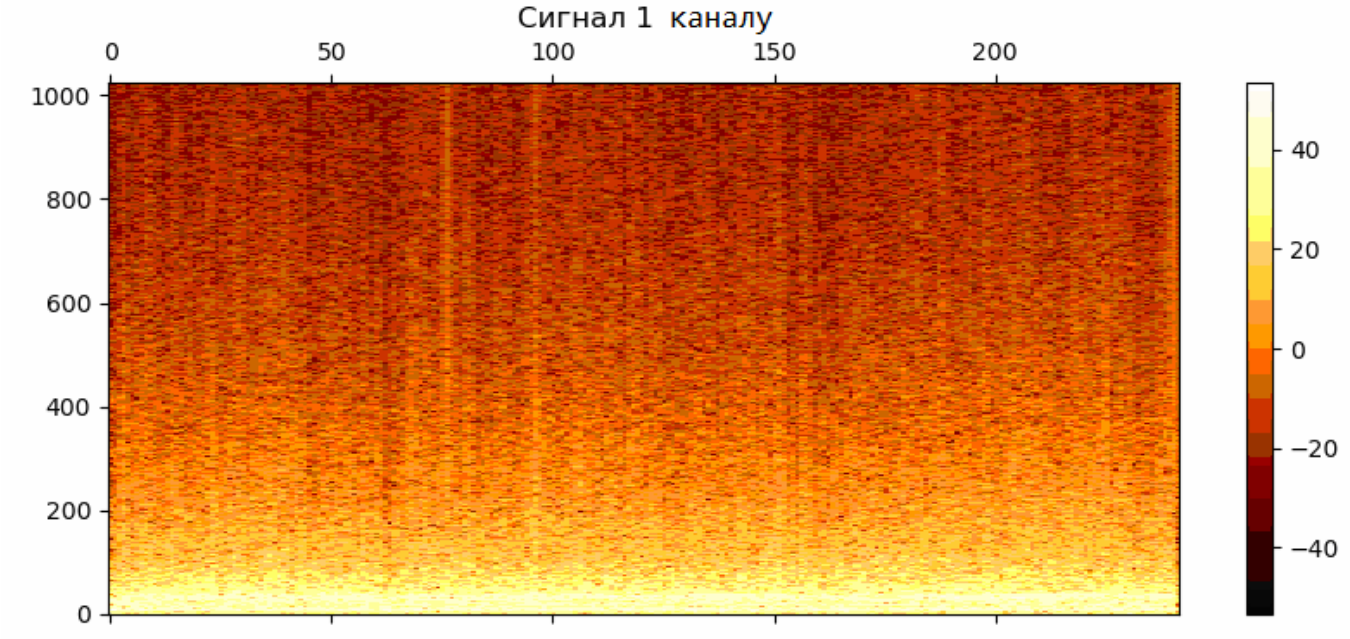

Рисунок 3.2 — Сигнал 1-го каналу до знешумлення

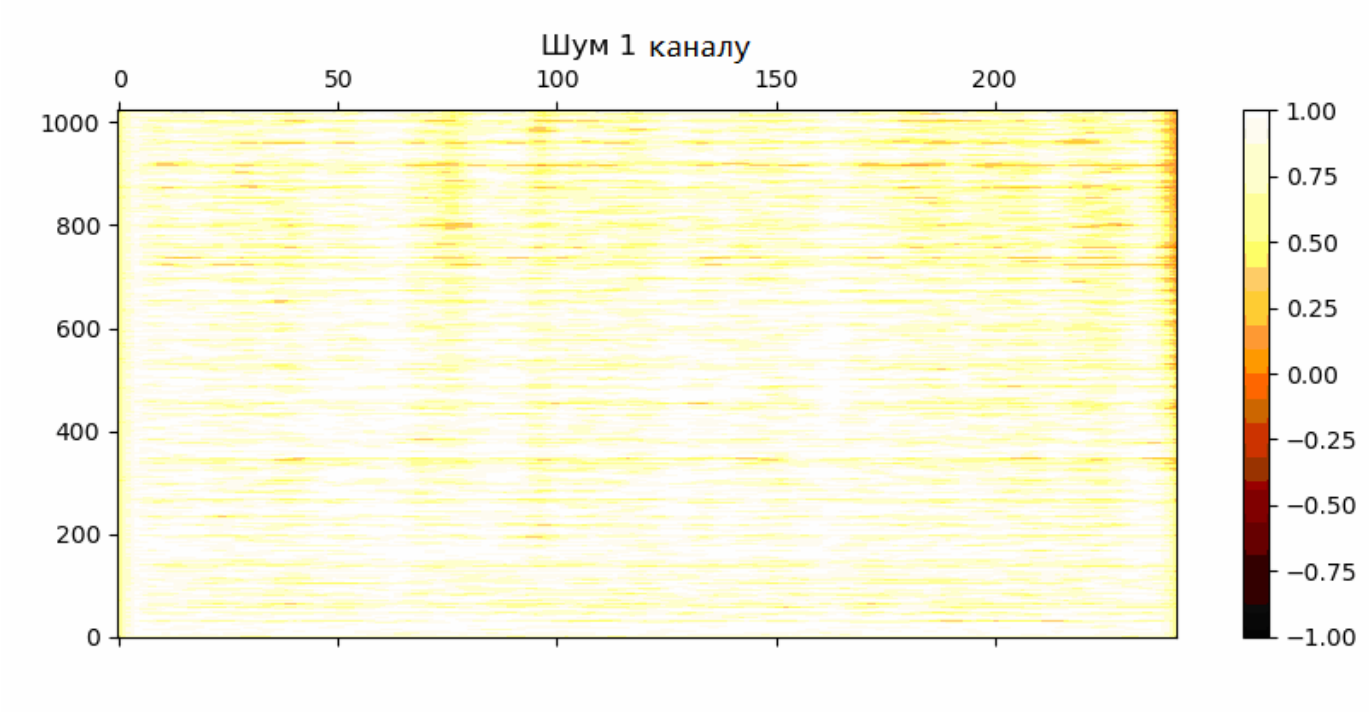

Рисунок 3.3 — Шум, вилучений з 1-го каналу

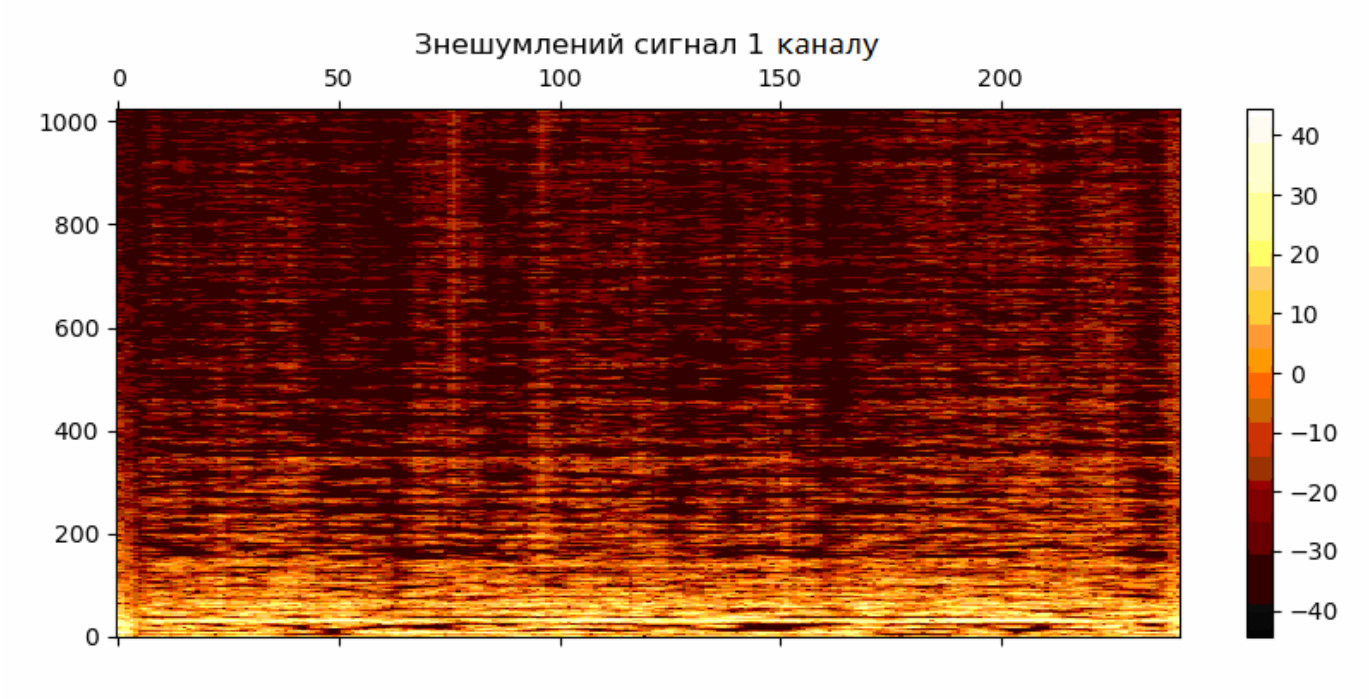

Рисунок 3.4 — Сигнал 1-го каналу після знешумлення

Відношення сигнал / шум для каналу 1 після фільтрації змінилося в 2,34 разів, що є хорошим показником для фільтрації гідроакустичного сигналу.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Для розв’язання задачі фільтрації гідроакустичних сигналів шляхом побудови програмної системи машинного навчання використовуються вибрані раніше інструменти розробки.

Програмна реалізація системи вимагає ретельно продуманої архітектури, яка чітко розділена на окремі компоненти і між якими встановлюється зв’язок.

Поставлена задача фільтрації сигналів складається з таких підзадач:

— обробка .dat-файлів для подальшої передачі на нейронну мережу (автоенкодер);

— візуалізація осцилограм, спектрів і спектрограм кожного з каналів;

— ілюстрація адаптивності шляхом визначення рівня “сигнал-шум” для кожного каналу окремо;

— застосування автоенкодера для фільтрації сигналів;

— розробка графічного інтерфейсу користувача.

Розв’язання цих підзадач пов’язане з такими питання, як аналіз даних, обробка і фільтрація аудіо-файлів, візуалізація даних. Для реалізації цих питань відповідно мають бути розроблені і створені окремі програмні компоненти. Зв’язок між ними буде встановлено за допомогою графічного інтерфейсу.

Графічний інтерфейс запускається в головному Python-процесі. Усі графіки будуються за допомогою модифікованого модуля Plotting і виводяться в графічний інтерфейс користувача, що є дуже зручним для аналізу сигналу після фільтрації.

Робота з файловою системою здійснюється за допомогою спеціального вбудованого пакету мови програмування Path, який дає можливість взаємодіяти з файлами в будь-якій операційній системі, що підтримує інтерпретатор CPython.

Модуль Noisereduce, який використовується для знешумлення зображень і сигналів за допомогою шумового автоенкодера, було адаптовано і використано для фільтрації гідроакустичного сигналу. Зокрема в даному модулі було переписано функцію виклику автоенкодера. За замовчуванням модуль обробляє сигнал повнісю,

не розбиваючи його на канали. Цю проблему було вирішеню, переписавши функцію модуля для кожного каналу.

Програмовані компоненти основної програми побудовано з використанням інтерфейсів між модулями і класами так, що систему можна легко розширити і змінити.

4.1. Архітектура програмної системи

Для реалізації поставленої задачі було створено програмне забезпечення, структура якого зображена на рисунку 4.1. Розроблений програмний продукт складається з трьох файлів з розширенням типу .py.

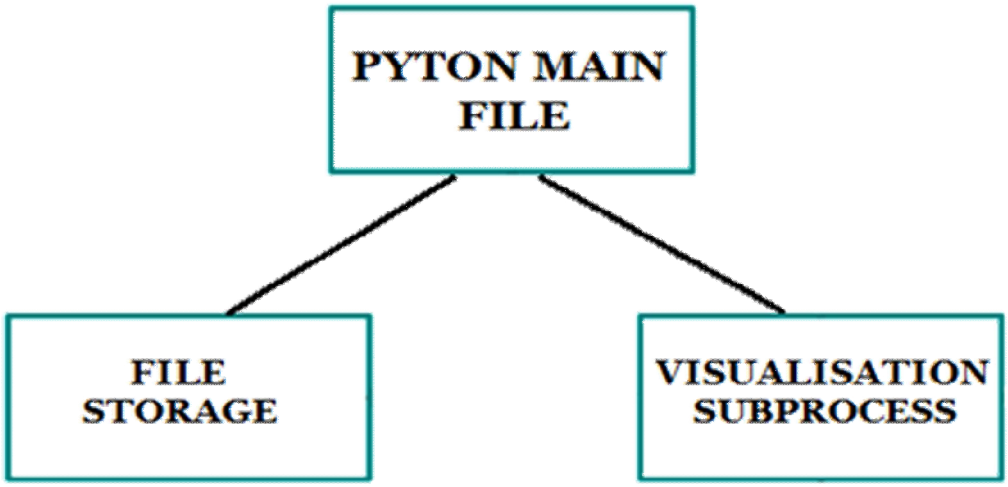

Рисунок 4.1 — Структура програмної системи

Перший файл — це Main-файл. Він є основним виконуваним файлом.

Другий файл File Storage — це файл, у якому на основі нейронної мережі запрограмовано роботу автоенкодера. Файл Main звертається до функцій файлу File Storage після обробки гідроакучтичного сигналу і розбиття .dat-файлу на 4 окремих канали.

Третій файл — відповідає за візуалізацю роботи автоенкодера. До цього файлу Main-файл звертається після отримання результатів фільтрації з File Storage для

їхнього коректного виведення у графічний інтерфейс користувача. Кожен з модулів має свої глобальні змінні, а також свої функції.

Взаємодія всіх вказаних вище компонентів між собою надає можливість коректного виконання програми.

Під час розробки програмної системи створено діаграму прецедентів, яка описує функціонал системи. Діаграму прецедентів програмної системи подано на рисунку 4.2.

Рисунок 4.2 — Діаграма прецедентів

Відповідно до діаграми можна зробити висновок, що система має широкий функціонал, а саме:

— завантаження .dat-файлу гідроакустичного сигналу;

— запуск обробки та фільтрації сигналу;

— побудову графіків до обробки сигналу;

— побудову графіків після обробки сигналу;

— збереження результатів фільтрації.

Архітектура системи нараховує три точки входу в програму — це Makefile-інтерфейс, Jupyter-інтерфейс і графічний користувацький інтерфейс.

Усі точки входу взаємодіють з центральним компонентом, який надає інтерфейси для роботи з іншими модулями. Серед них — модуль конфігурації, який надає можливість сконфігурувати проект, модуль для роботи автоенкодера, модуль зчитування і запису даних. Він включає в себе адаптери для роботи з файловою системою.

У ході розробки виникла проблема виведення всіх результатів для 4-ох каналів в графічний інтерфейс користувача (в одному вікні). Для усунення проблеми було вирішено модифікувати під програмну систему модуль з відкритим кодом noisereduce. Також даний модуль було оптимізовано для більш вузького напрямку, а саме для фільтрації гідроакустичного сигналу.

У результаті ці програмні рішення оптимізували програмну систему і дали можливість працювати з гідроакустичним сигналом і даними через той самий інтерфейс, не відкриваючи кожен графік в окремому вікні. Архітектуру системи зображено на рисунку 4.3.

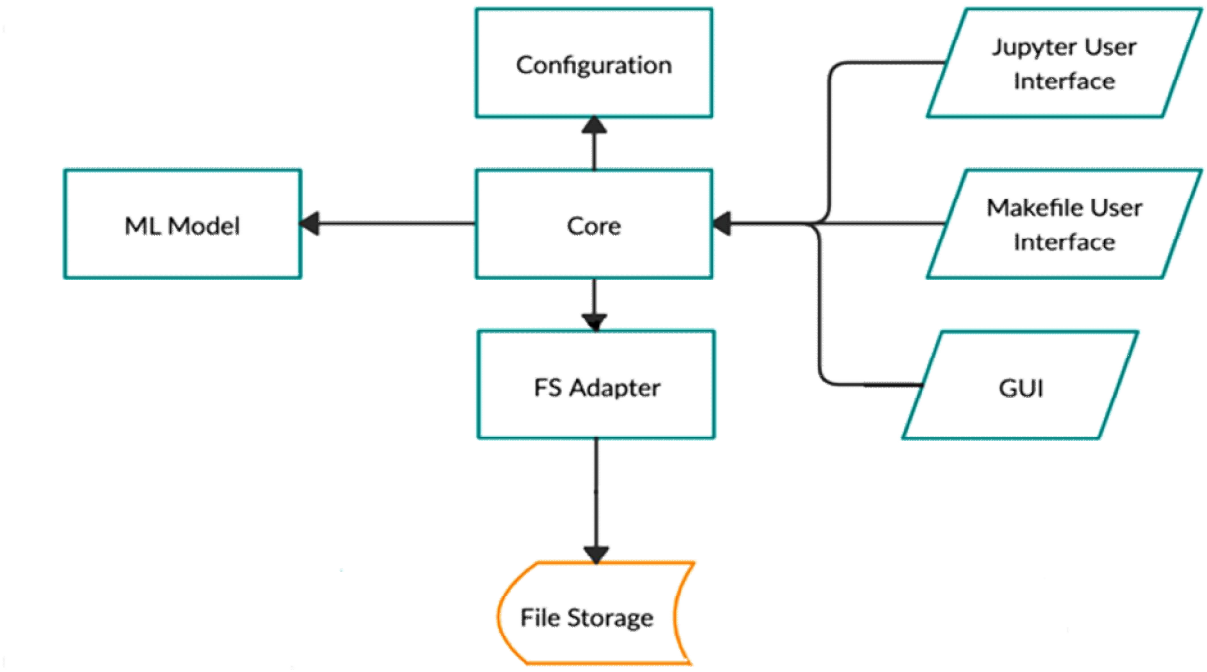

Рисунок 4.3 — Архітектура програмної системи

Інтерфейс Jupyter дає можливість виконувати різні Python-операції з використанням веб-інтерфейсу і зберігати історію в спеціальних файлах для надання можливості їхнього подальшого відновлення і використання.

Архітектура програмного забезпечення цієї системи написана за принципом “SOLID”, що є загальноприйнятою основою для написання коду в об’єктно-орієнтованому стилі. Дотримання цих принципів надає можливості щодо відносно простого розширення системи в майбутньому.

Як видно з вищевикладеного, програмний продукт написано відповідно до сучасних стандартів проектування і є легко розширюваним, щоб дозволити подальший розвиток продукту.

4.2 Алгоритм виконання роботи

На початковому етапі отримується аудіо-сигнал, записаний в форматі .dat, за допомогою написаної функції select_file(). Сам гідроакустичний сигнал має частоту 4096 Гц. Сигнал складається з чотирьох каналів. Оскільки .dat-файл отримано в шістнадцятковому поданні, для роботи програми потрібно зробити перетворення даного файлу і кожен елемент файлу перетворити в тип integer.

Після отримання файлу програмна система обчислює частоту, перевіряє кількість каналів і вираховує довжину сигналу. Всі вище сказані обробки відбуваються в основному файлі Main в функції main_fun().

Також було запрограмовано функцію signaltonoise (рисунок 4.4).

```
29
30  def signaltonoise(a, axis=0, ddof=0):
31      a = np.asanyarray(a)
32      m = a.mean(axis)
33      sd = a.std(axis=axis, ddof=ddof)
34      return np.where(sd == 0, 0, m/sd)
35
```

Рисунок 4.4 — Функція Main-файлу signaltonoise

Ця функція визначає рівень сигнал/шум у кожному каналі сигналу. Результати функції виводяться перед початком фільтрації та в кінці, що надає змогу зручно проаналізувати результати фільтрації.

Наступним етапом в програмній системі за допомогою фукнцій бібліотек Seaborn та Matplotlib реалізовано побудову осцилограм, магнітудних спектрів і спектрограм за кожним каналом сигналу.

Вивід спектру (рисунок 4.5) реалізовано з використанням швидкого перетворення Фур'є. Швидке перетворення Фур'є (ШПФ)— алгоритм для швидкого обчислення дискретного перетворення Фур'є. Якщо для прямого обчислення дискретного перетворення Фур'є з N точок даних потрібно $O(N^2)$ арифметичних дій, то ШПФ дає можливість вирахувати такий самий результат, використовуючи $O(N \log N)$ операцій. Алгоритм ШПФ часто застосовують для цифрової обробки сигналів при трансформації дискретних даних з часового у частотний діапазон.

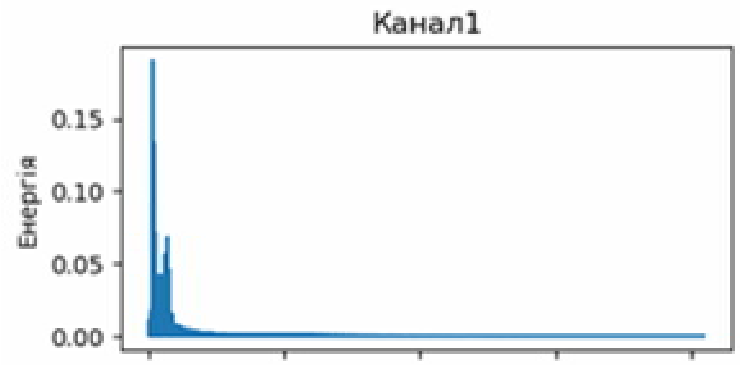

Рисунок 4.5 — Відображення спектра першого каналу

Звуковий спектр — це подання звуку — як правило, короткого зразка звуку — з точки зору кількості вібрацій на кожній окремій частоті. Як правило, він представлений у вигляді графіка потужності або тиску залежно від частоти. Потужність або тиск, як правило, вимірюється в децибелах, а частота вимірюється у вібраціях за секунду (або герцах, Гц). Звуковий спектр відображає різні частоти, присутні в звукові. Більшість звуків складається із складної суміші вібрацій. Шуми, затухаючі звуки мають суцільний спектр. Комбіновані спектри характерні для шумів деяких механізмів.

Після цього програма звертається до розробленого автоенкодера. Кожен канал окремо подається на автоенкодер і вертає результати до Main-файлу, відфільтровані 4 канали сигналу збираються в один файл, який зберігається на локальному компютері користувача.

Автоенкодер розроблено як нейронна мережа прямого поширення або багатошаровий персептрон. Мета мережі прямого поширення — апроксимувати деяку функцію f^*. Наприклад, у випадку класифікатора $y = f^*(x)$ відображає вхід x в категорію y. Мережа прямого поширення визначає відображення $y = f(x;\theta)$ і шляхом навчання знаходить значення параметрів θ, які дають найкращу апроксимацію.

Мережі прямого поширення виключно важливі для практичного застосування машинного навчання. Вони лежать в основі багатьох важливих комерційних додатків.

Шумовий автоенкодер (DAE) — це автоенкодер, який отримує на вході спотворені дані і навчається передбачати справжні, неспотворені дані. Процедура навчання автоенкодера в розробленій програмній системі показана на рисунку 4.6.

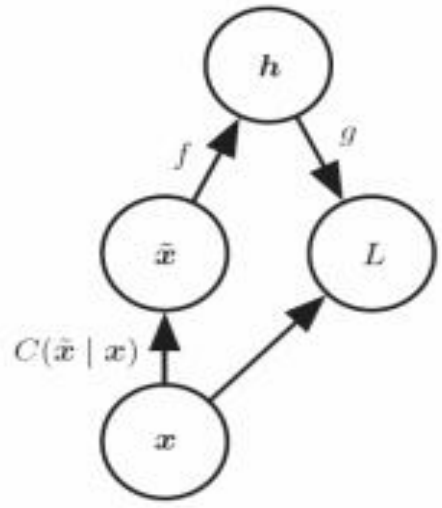

Рисунок 4.6 — Процедура навчання автоенкодера [20]

Спочатку відбувається процес $C(\tilde{x}|x)$, який представляє умовний розподіл спотворених прикладів $\tilde{x}$ за умови справжніх прикладів x. Потім автоенкодер

навчається розподілу реконструкції $\mathrm{p}_{\mathrm{reconstruct}}(x|\tilde{x})$, яке оцінюється за навчальними параметрами $(x,\tilde{x})$ так [20]:

1) вибирається навчальний приклад x з навчальних даних;

2) вибирається спотворена версія $\tilde{x}$ з $C(\tilde{x}|x=x)$;

3) використовується $(x,\tilde{x})$ як навчальний приклад для оцінки розподілу реконструкції автоенкодера $\mathrm{p}_{\mathrm{reconstruct}}(x|\tilde{x}) = \mathrm{p}_{\mathrm{decoder}}(x|h)$, де h — вихід кодувальника; у написаній програмній системі це відфільтрований канал сигналу, $f(\tilde{x})$, а $\mathrm{p}_{\mathrm{decoder}}$ визначається декодером $g(h)$.

Результати адаптивоної фільтрації автоенкодером після збереження виводяться окремо за каналами у графічний інтерфейс користувача для зручного аналізу всіх даних. Сам вплив автоенкодера на канали сигналу можна аналізувати завдяки виведенню наприкіці результатів значень і відношення рівня сигнал/шум, як це зображено на рисунку 4.7.

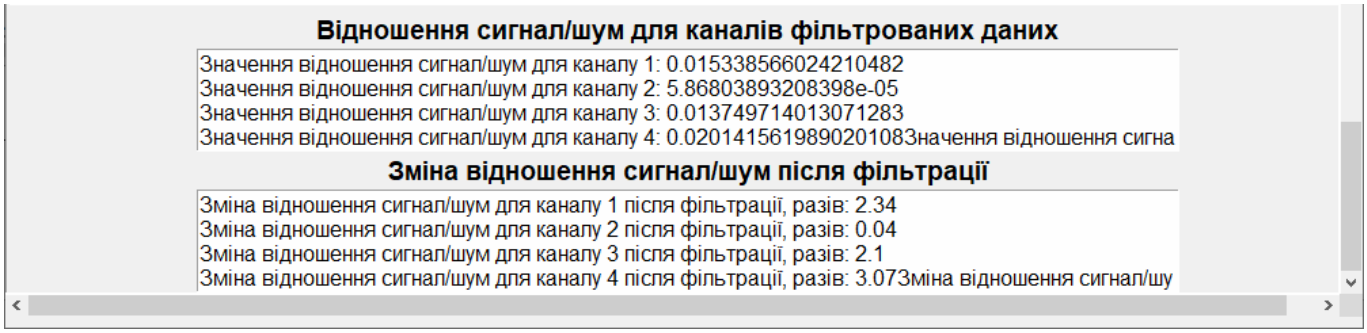

Рисунок 4.7 — Відображення відношення і зміну відношення сигнал/шум після фільтрації

Вище розглянуто програмну реалізацію програмної системи, подано покроковий алгоритм виконання роботи програми, структуру, діаграму прецендентів та архітектуру системи.

5. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Для використання створеної системи адаптивної фільтрації гідроакустичних сигналів на основі машинного навчання розроблено зручний інтуїтивно зрозумілий користувацький інтерфейс. Графічний інтерфейс є стандартним і використовується для базової взаємодії користувача з програмним продуктом.

Для забезпечення стабільної роботи створеної програмної системи треба дотримуватися рекомендацій щодо використання.

Для коректної роботи програмної системи, необхідно враховувати потужність комп'ютера, на якому вона буде працювати, дотримуватися вимог щодо версії операційної системи.

5.1. Системні вимоги

Для роботи з програмною системою фільтрації гідроакустичних сигналів необхідні такі мінімальні системні й технічні ресурси:

— операційна система — Windows 7,8,10, Ubuntu 20;

— процесор — 2 ядра, від 2.0 Ггц;

— оперативна пам'ять — 4 Гб;

— жорсткий диск — 50 Гб;

— аудіокарта — DirectX-сумісна.

5.2. Робота користувача з програмною системою

Для запуску проекту необхідно в командному рядку операційної системи ввести команду, як це зображено на рисунку 5.1.

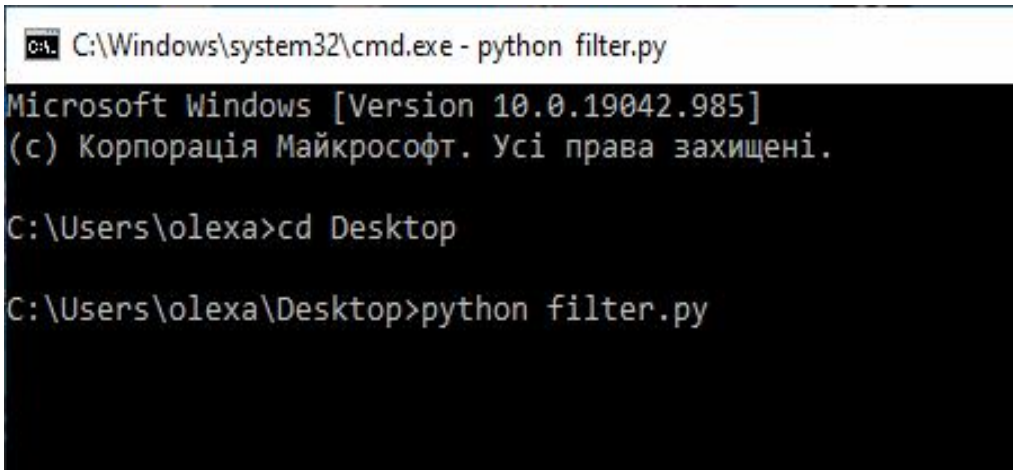

Рисунок 5.1 — Запуск програмної системи

Після запуску програми з'являється початкова форма, вигляд якої подано на рисунку 5.2.

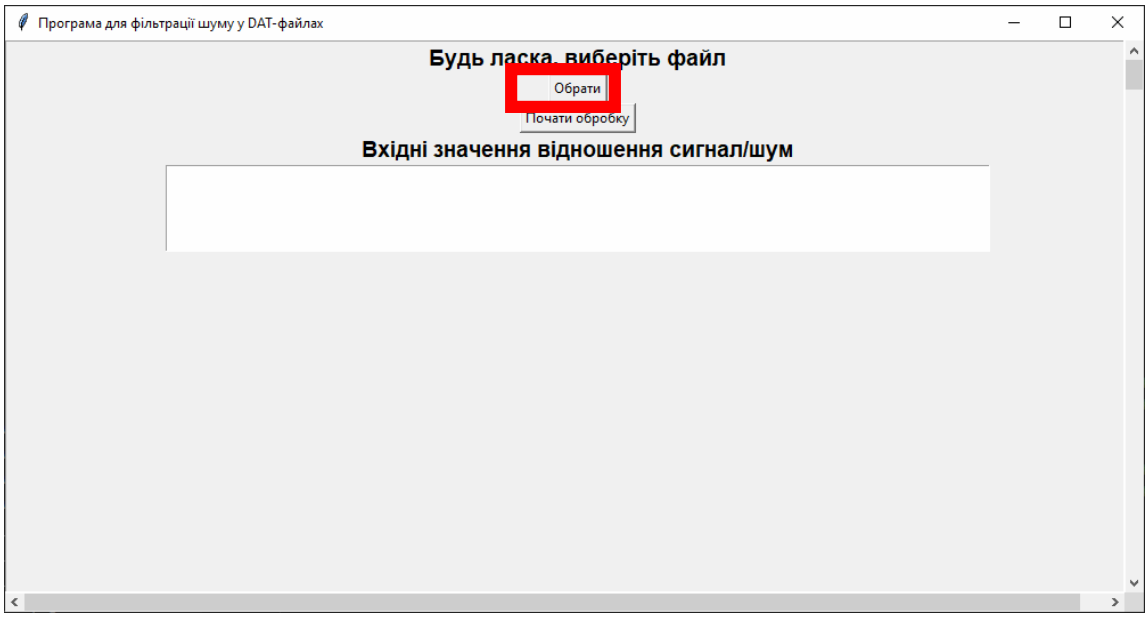

Рисунок 5.2. — Відображення початкового вікна програми

Після натискання на кнопку "Обрати" користувачеві відкривається "Провідник" і пропонується вибрати .dat-файл гідроакустичного сигналу для фільтрації (рисунок 5.3).

Після вибору файлу у вікні програми необхідно натиснути кнопку "Почати обробку" (рисунок 5.4).

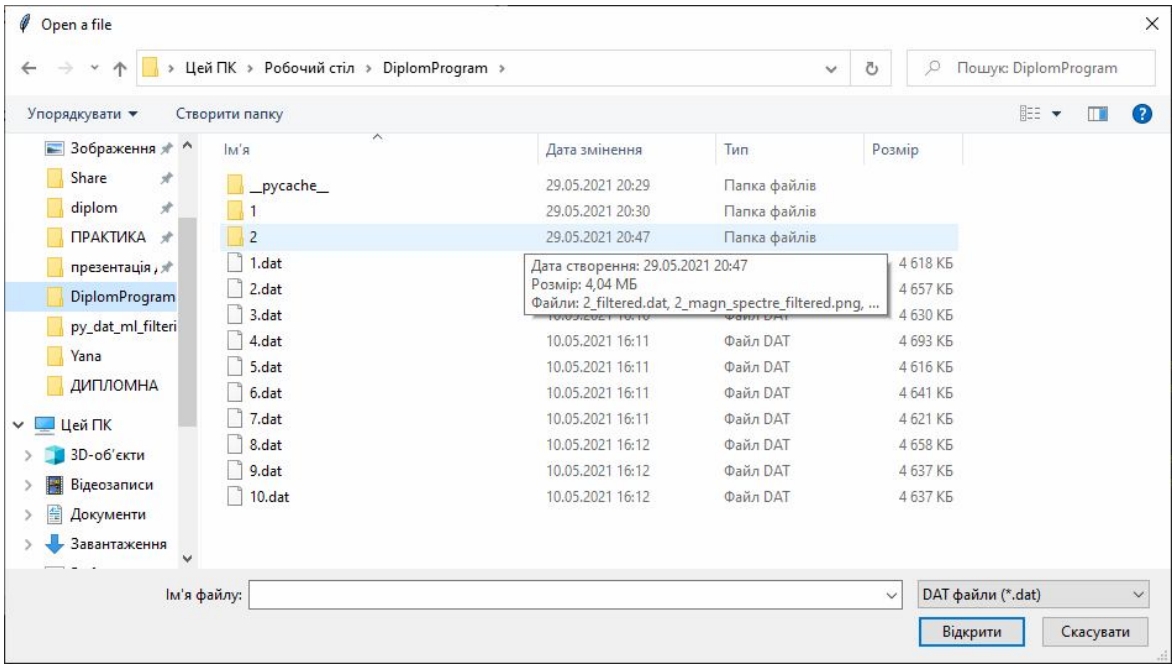

Рисунок 5.3 — Вибір файлу для обробки

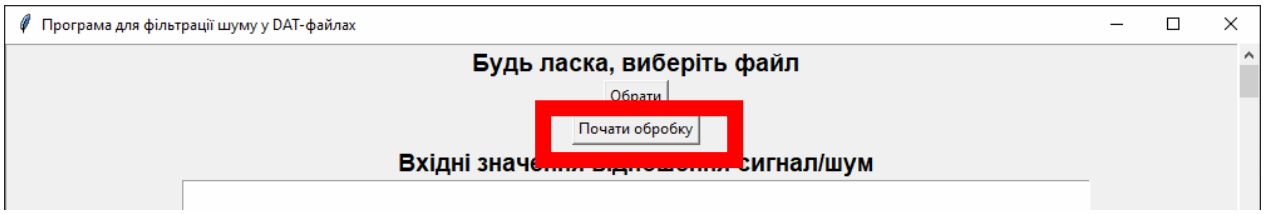

Рисунок 5.4. — Ініціалізація обробки файлу гідроакустичного сигналу

Після обробки файлу починають відображатися дані фільтрації гідроакустичного сигналу.

5.3. Результати роботи програмного продукту

Інтуїтивно зрозумілий користувацький інтерфейс системи адаптивної фільтрації гідроакустичних сигналів дає можливість повною мірою використовувати розроблений функціонал.

У папці з .dat-файлами гідроакустичних сигналів користувач має можливість вибрати файл сигналу для фільтрації і запустити процес фільтрації. Результати обробки відображаються у вигляді відношень і графіків характеристик сигналу, шуму, знешумленого сигналу.

Як перші результати обробки сигналу виводяться відношення сигнал-шум не фільтрованого гідроакустичного сигналу. Після цього виводяться осцилограми по кожному з 4-ох каналів сигналу. Наступними даними виводяться магнітудні спектри вхідного сигналу (рисунок 5.5).

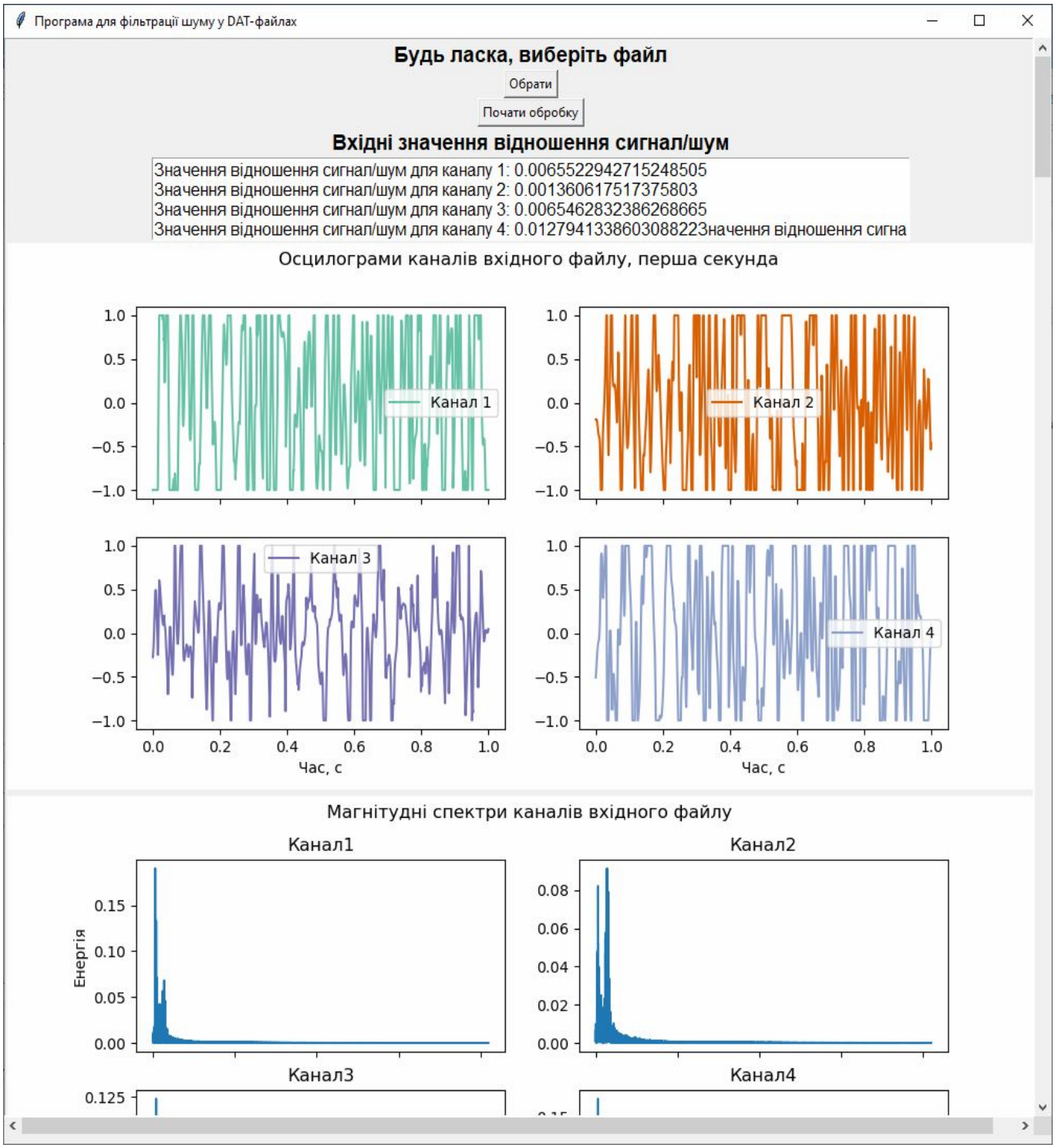

Рисунок 5.5 — Відношення сигнал/шум, осцилограми, магнітудні спектри вхідного сигналу

Такі ж дані виводяться і після фільтрації сигналу. Також реалізовано вивід сигналу після обробки автоенкодером. На графіках (рисунку 5.6) відображається шум, сигнал, та знешумлений сигнал.

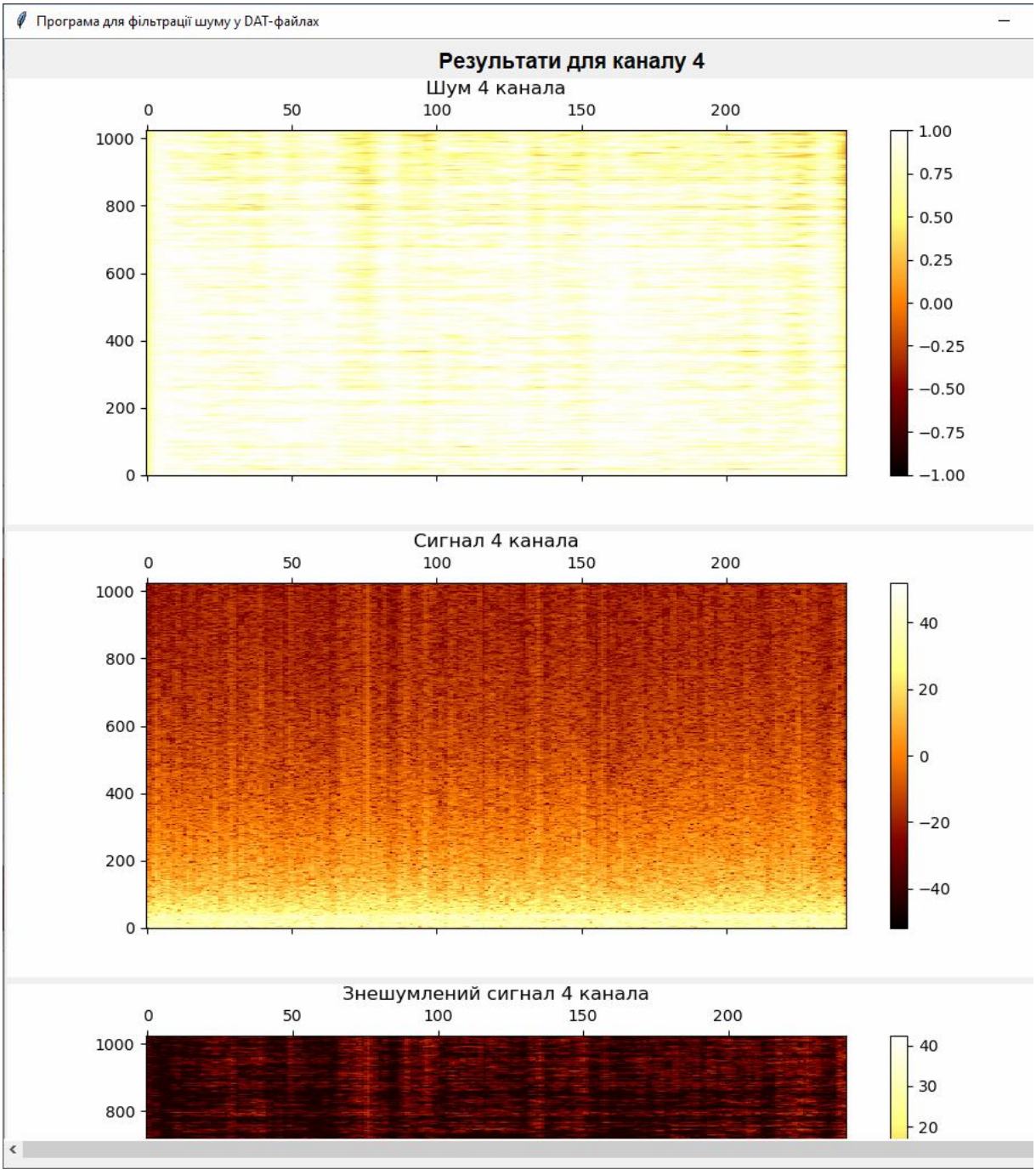

Рисунок 5.6 — Відображення шуму, сигналу та знешумленого сигналу.

На рисунку 5.7 показано спектрограми вже відфільтрованого сигналу. Під ними відображаються значення сигнал/шум для відфільтрованого сигналу та відношення зміни даної змінної до та після фільтрації.

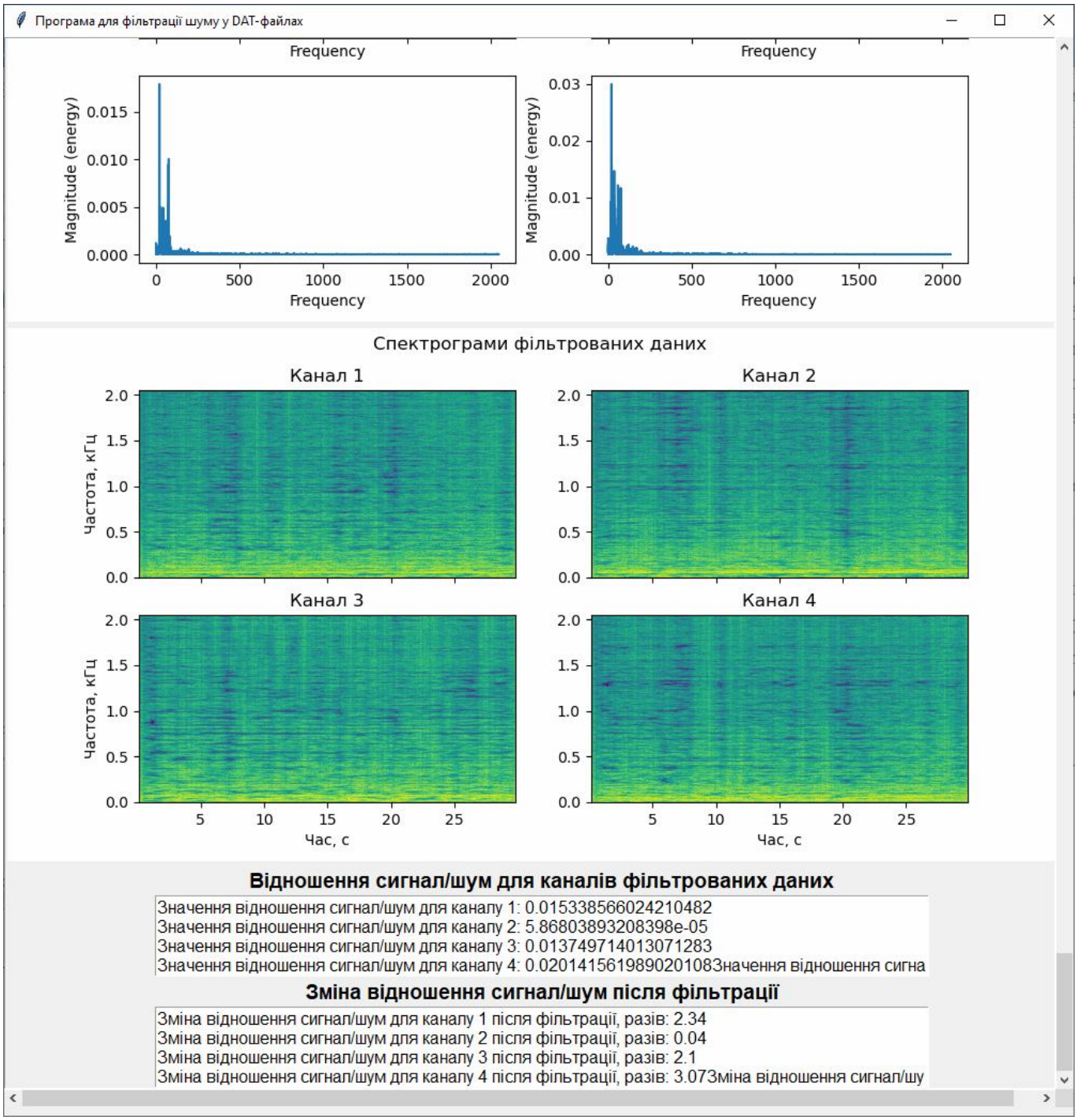

Рисунок 5.7 — Відображення результатів для відфільтрованого сигналу

Дані фільтруються окремо за кожним із 4-ох каналів гідроакустичного сигналу, саме в цьому і полягає адаптивність програмної системи.

Усі результати фільтрації автоматично зберігаються у папку з розташуванням програмної системи.

ВИСНОВКИ

У результаті виконання дипломної роботи проведено аналіз існуючих програмних систем обробки звукових сигналів. Розроблено програмну систему, призначену для адаптивної фільтрації гідроакустичних сигналів на основі машинного навчання. Автоенкодер (нейронну мережу прямого поширення) написано мовою програмування Python. Створено інтерфейс взаємодії з програмним продуктом. Реалізовано відображення необхідних графіків, а саме осцилограм 4-ох каналів, спектрів, спектрограм.

Створена програмна система є гнучкою для розширення і може бути використана як частина чи основа для більш широко направлених програмних рішень у сфері обробки гідроакустичних сигналів.

Користувачами системи можуть бути студенти навчальних закладів для поглиблення знань про гідроакустику та нейроні мережі прямого поширення, працівники сфери гідроакустики для проведення аналізу, обробки й фільтрації сигналів.

Результати роботи доповідалися на XIX Міжнародній науково-практичній конференції молодих вчених і студентів “Сучасні проблеми наукового забезпечення енергетики” 2021 року [21].

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Борисов Е.С. Методы обучения нейронных сетей с большим количеством скрытых слоёв [Електронний ресурс] — 2016. — Режим доступу: http://mechanoid.su/neural-net-mlp-deep.html

2. Боршигов К. Автоэнкодеры: типы архитектур и применение [Електронний ресурс] — 2018. — Режим доступу: https://neurohive.io/ru/osnovy-data-science/avtojenkoder-tipy-arhitektur-i-primenenie/

3. Словник української мови: в 11 томах [Електронний ресурс] — 1974. — Режим доступу: http://sum.in.ua/s/oscyloghrama

4. Сигнал і його типи [Електронний ресурс] — 2021. — Режим доступу: http://mcx.lab-101.org.ua/Tema1.htm

5. Дасгупта С., Пападимитриу Х., Вазирани У. Алгоритмы. — М. : МЦНМО, 2014. — 320 с.

6. Фильтрация сигналов [Електронний ресурс] — Режим доступу: https://zetlab.com/shop/programmnoe-obespechenie/funktsii-zetlab/avtomatizatsiya/filtratsiya-signalov/ .

7. Кулибин. Программа цифровой обработки сигнала “DSP” [Електронний ресурс] — 2013. — Режим доступу: http://radio-stv.ru/ot-chitateley/tsifrovaiy-obrabotka-signala

8. Фандеев В.П. Программа обработки сигналов SPTool [Електронний ресурс] — 2020. — Режим доступу: https://poznayka.org/s26791t2.html

9. The repositories for the Jupyter project [Електронний ресурс] — 2021. — Режим доступу: https://jupyter.org/documentation

10. Документація Visual Studio Code [Електронний ресурс] — 2021. — Режим доступу: https://code.visualstudio.com/docs

11. Python Software Foundation, Python 3.9.5 documentation [Електронний ресурс] — 2021. — Режим доступу: https://docs.python.org/3/

12. Документація Anaconda [Електронний ресурс] — 2021. — Режим доступу: https://www.anaconda.com/products/individual

13. Документація Conda [Електронний ресурс] — 2017. — Режим доступу: https://docs.conda.io/en/latest/

14. NumPy v1.20 Manual [Електронний ресурс] — 2021. — Режим доступу: https://numpy.org/doc/stable/

15. Waskom M. Seaborn Tutorial Python [Електронний ресурс] — 2020. — Режим доступу: https://seaborn.pydata.org/generated/seaborn.lineplot.html .

16. The Matplotlib development team. Matplotlib: Visualization with Python [Електронний ресурс] — 2021. — Режим доступу: https://matplotlib.org/.

17. Getting started with SciPy [Електронний ресурс] — 2021. — Режим доступу: https://www.scipy.org/getting-started.html

18. Tim Sainburg. Noisereduce Tutorial [Електронний ресурс] — 2019. — Режим доступу: https://github.com/timsainb/noisereduce#steps-of-algorithm.

19. GitHub Packages [Електронний ресурс] — 2021. — Режим доступу: https://docs.github.com/en/packages

20. Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение. — 2-е изд., испр. — М.: ДМК Пресс, 2018. — 652 с.

21. Козак О.А. Автоматизована система адаптивної фільтрації гідроакустичних сигналів на основі машинного навчання / О.А. Козак, Л.І. Кублій // Сучасні проблеми наукового забезпечення енергетики: Матеріали XIX Міжнародної науково-практичної конференції молодих вчених і студентів, м. Київ, 20-23 квітня 2021 року. — К.: КПІ ім. Ігоря Сікорського, Вид-во “Політехніка”, 2021. — Т. 2. — С. 206-207.

ДОДАТОК А

Автоматизована система адаптивної фільтрації гідроакустичних сигналів на основі машинного навчання

Специфікація

УКР.НТУУ “КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТВз7111_21Б

Аркушів 2

Київ — 2021

Позначення	**Найменування**	Примітки
Документація		
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ ТВз7111_21Б	Записка.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ ТВз7111_21Б 12-1	src/main.py	Модуль конфігурації програми
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ ТВз7111_21Б 12-2	src/noisereduce.py	Модулі центрального компонента
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ ТВз7111__21Б 12-3	src/plotting.py	Модуль головного вікна графічного інтерфейсу
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ ТВз7111__21Б 13-1	Опис.docx	Опис програми

ДОДАТОК Б

Автоматизована система адаптивної фільтрації гідроакустичних сигналів на основі машинного навчання

Текст програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ ТВз7111_21Б 12

Аркушів 24

Київ — 2021

```
from tkinter import *
import tkinter as tk
from tkinter.tix import *
from tkinter import filedialog as fd
from tkinter import ttk
from tkinter.messagebox import showinfo
from scipy.signal import spectrogram
from scipy.fft import fftshift
import seaborn as sns
import noisereduce as nr
import matplotlib.pyplot as plt
import numpy as np
import librosa
import pathlib
import time as t
import threading
import warnings
from tqdm.autonotebook import tqdm
import scipy.signal

#оскільки функція обрахування співвідношення сигнал/шум більше не підтримується, пишемо власну
def signaltonoise(a, axis=0, ddof=0):
    a = np.asanyarray(a)
    m = a.mean(axis)
    sd = a.std(axis=axis, ddof=ddof)
    return np.where(sd == 0, 0, m/sd)

def select_file():
```

```
    filetypes = (
        ('DAT файли', '*.dat'),
        ('Всі файли', '*.*')
    )
    global filename
    filename = fd.askopenfilename(parent=window,
        title='Open a file',
        initialdir='/',
        filetypes=filetypes)



def main_fun():

    # читаємо файл
    data = np.loadtxt(filename)
    global fname_pure
    fname_pure =  filename.split('/')[-1].split('.')[0]
    channels = 4
    rate=4096
    length = data.shape[0] / rate
    time = np.linspace(0., length, data.shape[0])
    data = data.astype(float)
    # нормалізуємо
    data = librosa.util.normalize(data, axis=1)

    # створюємо папку для рисунків та фільтрованого файлу
    p = pathlib.Path(fname_pure)
    p.mkdir(exist_ok=True)
```

```
    # рахуємо відношення сигнал/шум для кожного каналу
    snr01 = np.abs(signaltonoise(data[:,0]))
    snr02 = np.abs(signaltonoise(data[:,1]))
    snr03 = np.abs(signaltonoise(data[:,2]))
    snr04 = np.abs(signaltonoise(data[:,3]))
    s = '''Значення відношення сигнал/шум для каналу 1: {}\nЗначення відношення сигнал/шум для каналу 2: {}\nЗначення відношення сигнал/шум для каналу 3: {}\nЗначення відношення сигнал/шум для каналу 4: {}'''.format(snr01, snr02, snr03, snr04)

    tex1.insert(END, s)

    #будуємо осцилограми
    fig, axes = plt.subplots(2, 2, sharex=True, figsize=(10,5))
    fig.suptitle('Осцилограми каналів вхідного файлу, перша секунда')
    sns.lineplot(ax=axes[0, 0], x=time[:4096], y=data[:4096, 0], label="Канал 1", color='#66c2a5')
    sns.lineplot(ax=axes[0, 1], x=time[:4096], y=data[:4096, 1], label="Канал 2", color='#d95f02')
    sns.lineplot(ax=axes[1, 0], x=time[:4096], y=data[:4096, 2], label="Канал 3", color='#7570b3')
    sns.lineplot(ax=axes[1, 1], x=time[:4096], y=data[:4096, 3], label="Канал 4", color='#8da0cb')
    axes[1, 0].set_xlabel('Час, с')
    axes[1, 1].set_xlabel('Час, с')
    fig.savefig(fname_pure + '/' + fname_pure + '_oscillograms_noisy.png', format='png')

    window.img1 = PhotoImage(file=fname_pure + '/' + fname_pure + '_oscillograms_noisy.png')
    canvas1.create_image((0,0), image=window.img1, anchor='nw')
```

```
# Будуємо магнітудні спектри для першої секудни вхідного файлу
fig, axes = plt.subplots(2, 2, sharex=True, figsize=(10,5))
fig.suptitle('Магнітудні спектри каналів вхідного файлу')
ax0=axes[0, 0]
ax0.magnitude_spectrum(data[:, 0] , Fs=rate)
ax1=axes[0, 1]
ax1.magnitude_spectrum(data[:, 1], Fs=rate)
ax2=axes[1, 0]
ax2.magnitude_spectrum(data[:, 2], Fs=rate)
ax3=axes[1, 1]
ax3.magnitude_spectrum(data[:, 3], Fs=rate)

axes[0, 0].set_title('Канал1')
axes[0, 1].set_title('Канал2')
axes[1, 0].set_title('Канал3')
axes[1, 1].set_title('Канал4')



axes[1, 0].set_xlabel('Частота')
axes[0, 0].set_xlabel('')
axes[1, 1].set_xlabel('Частота')
axes[0, 1].set_xlabel('')
axes[0, 0].set_ylabel('Енергія')
axes[0, 1].set_ylabel('')
axes[1, 0].set_ylabel('Енергія')
axes[1, 1].set_ylabel('')


fig.savefig(fname_pure + '/' + fname_pure + '_magn_spectre_noisy.png',
```

```
format='png')

   window.img2 = PhotoImage(file=fname_pure + '/' + fname_pure + '_magn_spectre_noisy.png')
   canvas2.create_image((0,0), image=window.img2, anchor='nw')


   # Будуємо спектрограми вихідного файлу
   fig, axes = plt.subplots(2, 2, sharex=True, figsize=(10,5))
   fig.suptitle('Спектрограми каналів вхідного файлу, перша секунда')

   freqs0, times0, Sx0 = spectrogram(data[:4096, 0], fs=rate, window='hanning',
                                     nperseg=1024, noverlap=900,
                                     detrend=False, scaling='spectrum')
   freqs1, times1, Sx1 = spectrogram(data[:4096, 1], fs=rate, window='hanning',
                                     nperseg=1024, noverlap=900,
                                     detrend=False, scaling='spectrum')
   freqs2, times2, Sx2 = spectrogram(data[:4096, 2], fs=rate, window='hanning',
                                     nperseg=1024, noverlap=900,
                                     detrend=False, scaling='spectrum')
   freqs3, times3, Sx3 = spectrogram(data[:4096, 3], fs=rate, window='hanning',
                                     nperseg=1024, noverlap=900,
                                     detrend=False, scaling='spectrum')

   ax0=axes[0, 0]
   ax0.pcolormesh(times0, freqs0 / 1000, 10 * np.log10(Sx0), cmap='viridis', label="channel01")
   ax0.set_ylabel('Частота, кГц')
   ax0.set_title('Канал 1')
```

```
    ax1=axes[0, 1]
    ax1.pcolormesh(times1, freqs1 / 1000, 10 * np.log10(Sx1), cmap='viridis', label="channel02")
    ax1.set_title('Канал 2')

    ax2=axes[1, 0]
    ax2.pcolormesh(times2, freqs2 / 1000, 10 * np.log10(Sx2), cmap='viridis', label="channel03")
    ax2.set_ylabel('Частота, кГц')
    ax2.set_title('Канал 3')
    ax2.set_xlabel('Час, с')

    ax3=axes[1, 1]
    ax3.pcolormesh(times3, freqs3 / 1000, 10 * np.log10(Sx3), cmap='viridis', label="channel04")
    ax3.set_title('Канал 4')
    ax3.set_xlabel('Час, с')

    fig.savefig(fname_pure + '/' + fname_pure + '_spectrograms_noisy.png', format='png')

    window.img3 = PhotoImage(file=fname_pure + '/' + fname_pure + '_spectrograms_noisy.png')
    canvas3.create_image((0,0), image=window.img3, anchor='nw')

    # фільтруємо шум за допомогою маски, побудованої по першій секунді каналу

    reduced_noise01 = nr.reduce_noise(audio_clip=data[:, 0], noise_clip=data[0:4096, 0], verbose=True, ch=1)
    pb['value'] += 200
```

```
    # window.update_idletasks()
    reduced_noise02 = nr.reduce_noise(audio_clip=data[:, 1], noise_clip=data[0:4096, 1],
verbose=True, ch=2)
    pb['value'] += 200

    # window.update_idletasks()
    reduced_noise03 = nr.reduce_noise(audio_clip=data[:, 2], noise_clip=data[0:4096, 2],
verbose=True, ch=3)
    pb['value'] += 200

    # window.update_idletasks()
    reduced_noise04 = nr.reduce_noise(audio_clip=data[:, 3], noise_clip=data[0:4096, 3],
verbose=True, ch=4)
    pb['value'] += 200

    # window.update_idletasks()
    window.img_01_01 = PhotoImage(file="Шум 1 канала__1_.png")
    canvas_01_01.create_image((0,0), image=window.img_01_01, anchor='nw')
    window.img_01_02 = PhotoImage(file="Сигнал 1 канала__1_.png")
    canvas_01_02.create_image((0,0), image=window.img_01_02, anchor='nw')
    window.img_01_03 = PhotoImage(file="Знешумлений сигнал 1 канала__1_.png")
    canvas_01_03.create_image((0,0), image=window.img_01_03, anchor='nw')


    window.img_02_01 = PhotoImage(file="Шум 2 канала__2_.png")
    canvas_02_01.create_image((0,0), image=window.img_02_01, anchor='nw')
    window.img_02_02 = PhotoImage(file="Сигнал 2 канала__2_.png")
    canvas_02_02.create_image((0,0), image=window.img_02_02, anchor='nw')
    window.img_02_03 = PhotoImage(file="Знешумлений сигнал 2 канала__2_.png")
    canvas_02_03.create_image((0,0), image=window.img_02_03, anchor='nw')
```

```
    window.img_03_01 = PhotoImage(file="Шум 3 канала__3_.png")
    canvas_03_01.create_image((0,0), image=window.img_03_01, anchor='nw')
    window.img_03_02 = PhotoImage(file="Сигнал 3 канала__3_.png")
    canvas_03_02.create_image((0,0), image=window.img_03_02, anchor='nw')
    window.img_03_03 = PhotoImage(file="Знешумлений сигнал 3 канала__3_.png")
    canvas_03_03.create_image((0,0), image=window.img_03_03, anchor='nw')


    window.img_04_01 = PhotoImage(file="Шум 4 канала__4_.png")
    canvas_04_01.create_image((0,0), image=window.img_04_01, anchor='nw')
    window.img_04_02 = PhotoImage(file="Сигнал 4 канала__4_.png")
    canvas_04_02.create_image((0,0), image=window.img_04_02, anchor='nw')
    window.img_04_03 = PhotoImage(file="Знешумлений сигнал 4 канала__4_.png")
    canvas_04_03.create_image((0,0), image=window.img_04_03, anchor='nw')

    # Об'єднуємо фільтровані дані у двовимірний масив
    data_n_red=np.stack((reduced_noise01, reduced_noise02,
reduced_noise03,reduced_noise04), axis=1)
    # Будуємо осцилограми фільтрованих даних
    fig, axes = plt.subplots(2, 2, sharex=True, figsize=(10,5))
    fig.suptitle('Осцилограми фільтрованих даних, перша секунда')

    sns.lineplot(ax=axes[0, 0], x=time[:4096], y=data_n_red[:4096, 0], label="Канал 1",
color='#1b9e77')
    sns.lineplot(ax=axes[0, 1], x=time[:4096], y=data_n_red[:4096, 1], label="Канал 2",
color='#d95f02')
    sns.lineplot(ax=axes[1, 0], x=time[:4096], y=data_n_red[:4096, 2], label="Канал 3",
color='#7570b3')
```

```
    sns.lineplot(ax=axes[1, 1], x=time[:4096], y=data_n_red[:4096, 3], label="Канал 4", color='#8da0cb')
    axes[1, 0].set_xlabel('Час, с')
    axes[1, 1].set_xlabel('Час, с')
    fig.savefig(fname_pure + '/' + fname_pure + '_oscillograms_filtered.png', format='png')

    window.img4 = PhotoImage(file=fname_pure + '/' + fname_pure + '_oscillograms_filtered.png')
    canvas4.create_image((0,0), image=window.img4, anchor='nw')

    # Будуємо магнітудні спектри для першої секудни фільтрованих даних
    fig, axes = plt.subplots(2, 2, sharex=True, figsize=(10,5))
    fig.suptitle('Магнітудні спектри каналів фільтрованих даних')
    ax0=axes[0, 0]
    ax0.magnitude_spectrum(data_n_red[:, 0], Fs=rate)
    ax1=axes[0, 1]
    ax1.magnitude_spectrum(data_n_red[:, 1], Fs=rate)
    ax2=axes[1, 0]
    ax2.magnitude_spectrum(data_n_red[:, 2], Fs=rate)
    ax3=axes[1, 1]
    ax3.magnitude_spectrum(data_n_red[:, 3], Fs=rate)
    fig.savefig(fname_pure + '/' + fname_pure + '_magn_spectre_filtered.png', format='png')

    ax0.set_title('Канал 1')
    ax1.set_title('Канал 2')
    ax2.set_title('Канал 3')
    ax3.set_title('Канал 4')
```

```
ax1.set_xlabel('Частота')
ax0.set_xlabel('')
ax3.set_xlabel('Частота')
ax2.set_xlabel('')
ax0.set_ylabel('Енергія')
ax1.set_ylabel('')
ax2.set_ylabel('Енергія')
ax3.set_ylabel('')

window.img5 = PhotoImage(file=fname_pure + '/' + fname_pure + '_magn_spectre_filtered.png')
canvas5.create_image((0,0), image=window.img5, anchor='nw')

# Будуємо спектрограми фільтрованих даних
fig, axes = plt.subplots(2, 2, sharex=True, figsize=(10,5))
fig.suptitle('Спектрограми фільтрованих даних')

freqs0, times0, Sx0 = spectrogram(data_n_red[:, 0], fs=rate, window='hanning',
                                  nperseg=1024, noverlap=900,
                                  detrend=False, scaling='spectrum')

freqs1, times1, Sx1 = spectrogram(data_n_red[:, 1], fs=rate, window='hanning',
                                  nperseg=1024, noverlap=900,
                                  detrend=False, scaling='spectrum')
freqs2, times2, Sx2 = spectrogram(data_n_red[:, 2], fs=rate, window='hanning',
                                  nperseg=1024, noverlap=900,
                                  detrend=False, scaling='spectrum')
freqs3, times3, Sx3 = spectrogram(data_n_red[:, 3], fs=rate, window='hanning',
                                  nperseg=1024, noverlap=900,
                                  detrend=False, scaling='spectrum')
```

```
    ax0=axes[0, 0]
    ax0.pcolormesh(times0, freqs0 / 1000, 10 * np.log10(Sx0), cmap='viridis', label="channel01")
    ax0.set_ylabel('Частота, кГц')
    ax0.set_title('Канал 1')

    ax1=axes[0, 1]
    ax1.pcolormesh(times1, freqs1 / 1000, 10 * np.log10(Sx1), cmap='viridis', label="channel02")
    ax1.set_title('Канал 2')


    ax2=axes[1, 0]
    ax2.pcolormesh(times2, freqs2 / 1000, 10 * np.log10(Sx2), cmap='viridis', label="channel03")
    ax2.set_ylabel('Частота, кГц')
    ax2.set_title('Канал 3')
    ax2.set_xlabel('Час, с')

    ax3=axes[1, 1]
    ax3.pcolormesh(times3, freqs3 / 1000, 10 * np.log10(Sx3), cmap='viridis', label="channel04")
    ax3.set_title('Канал 4')
    ax3.set_xlabel('Час, с')
    fig.savefig(fname_pure + '/' + fname_pure + '_spectrograms_filtered.png', format='png')

    window.img6 = PhotoImage(file=fname_pure + '/' + fname_pure + '_spectrograms_filtered.png')
```

```
canvas6.create_image((0,0), image=window.img6, anchor='nw')

#Визначаємо рівень сигнал/шум для фільтрованих даних
snr01_nr = np.abs(signaltonoise(data_n_red[:,0]))
snr02_nr = np.abs(signaltonoise(data_n_red[:,1]))
snr03_nr = np.abs(signaltonoise(data_n_red[:,2]))
snr04_nr = np.abs(signaltonoise(data_n_red[:,3]))

s = '''Значення відношення сигнал/шум для каналу 1: {}\nЗначення відношення сигнал/шум для каналу 2: {}\nЗначення відношення сигнал/шум для каналу 3: {}\nЗначення відношення сигнал/шум для каналу 4: {}'''.format(snr01_nr, snr02_nr, snr03_nr, snr04_nr)
tex2.insert(END, s)

    # Визначаємо зміну відношення сигнал / шум після фільтрації
s='''Зміна відношення сигнал/шум для каналу 1 після фільтрації, разів: {}\nЗміна відношення сигнал/шум для каналу 2 після фільтрації, разів: {}\nЗміна відношення сигнал/шум для каналу 3 після фільтрації, разів: {}\nЗміна відношення сигнал/шум для каналу 4 після фільтрації, разів: {}'''.format(round(snr01_nr/snr01, 2), round(snr02_nr/snr02, 2), round(snr03_nr/snr03, 2), round(snr04_nr/snr01, 2))
tex3.insert(END, s)

#Зберігаємо отримані результати
np.savetxt(fname_pure + '/' + fname_pure + '_filtered.dat', data_n_red, fmt=['%.3f','%.3f', '%.3f','%.3f'])
#-----------------------------------------------------------------------------------------------
```



```
import scipy.signal
import numpy as np
```

```
import librosa
from plotting import plot_reduction_steps1, plot_reduction_steps2,
plot_reduction_steps3,plot_reduction_steps4
from tqdm.autonotebook import tqdm
import warnings


def _stft(y, n_fft, hop_length, win_length, use_tensorflow=False):
    if use_tensorflow:
        # return librosa.stft(y=y, n_fft=n_fft, hop_length=hop_length,
win_length=win_length, center=True)
        return _stft_tensorflow(y, n_fft, hop_length, win_length)
    else:
        return librosa.stft(
            y=y, n_fft=n_fft, hop_length=hop_length, win_length=win_length, center=True
        )


def _istft(y, n_fft, hop_length, win_length, use_tensorflow=False):
    if use_tensorflow:
        # return librosa.istft(y, hop_length, win_length)
        return _istft_tensorflow(y.T, n_fft, hop_length, win_length)
    else:
        return librosa.istft(y, hop_length, win_length)


def _stft_librosa(y, n_fft, hop_length, win_length):
    return librosa.stft(
        y=y, n_fft=n_fft, hop_length=hop_length, win_length=win_length, center=True
    )
```

```
def _istft_librosa(y, hop_length, win_length):
    return librosa.istft(y, hop_length, win_length)


def _stft_tensorflow(y, n_fft, hop_length, win_length):
    return (
        tf.signal.stft(
            y,
            win_length,
            hop_length,
            n_fft,
            pad_end=True,
            window_fn=tf.signal.hann_window,
        )
        .numpy()
        .T
    )


def _istft_tensorflow(y, n_fft, hop_length, win_length):
    return tf.signal.inverse_stft(
        y.astype(np.complex64), win_length, hop_length, n_fft
    ).numpy()


def _amp_to_db(x):
    return librosa.core.amplitude_to_db(x, ref=1.0, amin=1e-20, top_db=80.0)
```

```
def _db_to_amp(x,):
    return librosa.core.db_to_amplitude(x, ref=1.0)


def update_pbar(pbar, message):
    """ writes to progress bar
    """
    if pbar is not None:
        pbar.set_description(message)
        pbar.update(1)


def _smoothing_filter(n_grad_freq, n_grad_time):

    smoothing_filter = np.outer(
        np.concatenate(
            [
                np.linspace(0, 1, n_grad_freq + 1, endpoint=False),
                np.linspace(1, 0, n_grad_freq + 2),
            ]
        )[1:-1],
        np.concatenate(
            [
                np.linspace(0, 1, n_grad_time + 1, endpoint=False),
                np.linspace(1, 0, n_grad_time + 2),
            ]
        )[1:-1],
    )
    smoothing_filter = smoothing_filter / np.sum(smoothing_filter)
    return smoothing_filter
```

```
def mask_signal(sig_stft, sig_mask):

    sig_stft_amp = sig_stft * (1 — sig_mask)
    return sig_stft_amp


def convolve_gaussian(sig_mask, smoothing_filter, use_tensorflow=False):

    if use_tensorflow:
        smoothing_filter = smoothing_filter * (
            (np.shape(smoothing_filter)[1] — 1) / 2 + 1
        )
        smoothing_filter = smoothing_filter[:, :, tf.newaxis, tf.newaxis].astype(
            "float32"
        )
        img = sig_mask[:, :, tf.newaxis, tf.newaxis].astype("float32")
        return (
            tf.nn.conv2d(img, smoothing_filter, strides=[1, 1, 1, 1], padding="SAME")
            .numpy()
            .squeeze()
        )
    else:
        return scipy.signal.fftconvolve(sig_mask, smoothing_filter, mode="same")


def load_tensorflow(verbose=False):

    try:
```

```
        # import tensorflow as tf
        globals()["tf"] = __import__("tensorflow")

        if verbose:
            available_gpus = tf.config.experimental.list_physical_devices("GPU")
            print("GPUs available: {}".format(available_gpus))
        if int(tf.__version__[0]) < 2:
            warnings.warn(
                "Tensorflow version is below 2.0, reverting to non-tensorflow backend"
            )
            return False
    except:
        warnings.warn(
            "Tensorflow is not installed, reverting to non-tensorflow backend"
        )
        return False
    return True


def reduce_noise(
    audio_clip,
    noise_clip,
    n_grad_freq=2,
    n_grad_time=4,
    n_fft=2048,
    win_length=2048,
    hop_length=512,
    n_std_thresh=1.5,
    prop_decrease=1.0,
    pad_clipping=True,
```

```
    use_tensorflow=False,
    verbose=False,
    ch=1,
):
    # load tensorflow if you are using it as a backend
    if use_tensorflow:
        use_tensorflow = load_tensorflow(verbose)

    if verbose:
        pbar = tqdm(total=7)
    else:
        pbar = None

    update_pbar(pbar, "STFT on noise")
    # STFT over noise
    noise_stft = _stft(
        noise_clip, n_fft, hop_length, win_length, use_tensorflow=use_tensorflow
    )
    noise_stft_db = _amp_to_db(np.abs(noise_stft))  # convert to dB
    # Calculate statistics over noise
    update_pbar(pbar, "STFT on signal")
    mean_freq_noise = np.mean(noise_stft_db, axis=1)
    std_freq_noise = np.std(noise_stft_db, axis=1)
    noise_thresh = mean_freq_noise + std_freq_noise * n_std_thresh
    # STFT over signal
    update_pbar(pbar, "STFT on signal")

    # pad signal with zeros to avoid extra frames being clipped if desired
    if pad_clipping:
        nsamp = len(audio_clip)
```

```
      audio_clip = np.pad(audio_clip, [0, hop_length], mode="constant")

  sig_stft = _stft(
    audio_clip, n_fft, hop_length, win_length, use_tensorflow=use_tensorflow
)
# spectrogram of signal in dB
sig_stft_db = _amp_to_db(np.abs(sig_stft))
update_pbar(pbar, "Generate mask")

# calculate the threshold for each frequency/time bin
db_thresh = np.repeat(
    np.reshape(noise_thresh, [1, len(mean_freq_noise)]),
    np.shape(sig_stft_db)[1],
    axis=0,
).T
# mask if the signal is above the threshold
sig_mask = sig_stft_db < db_thresh
update_pbar(pbar, "Smooth mask")
# Create a smoothing filter for the mask in time and frequency
smoothing_filter = _smoothing_filter(n_grad_freq, n_grad_time)

# convolve the mask with a smoothing filter
sig_mask = convolve_gaussian(sig_mask, smoothing_filter, use_tensorflow)

sig_mask = sig_mask * prop_decrease
update_pbar(pbar, "Apply mask")
# mask the signal

sig_stft_amp = mask_signal(sig_stft, sig_mask)
```

```
update_pbar(pbar, "Recover signal")
# recover the signal
recovered_signal = _istft(
    sig_stft_amp, n_fft, hop_length, win_length, use_tensorflow=use_tensorflow
)
# fix the recovered signal length if padding signal
if pad_clipping:
    recovered_signal = librosa.util.fix_length(recovered_signal, nsamp)

recovered_spec = _amp_to_db(
    np.abs(
        _stft(
            recovered_signal,
            n_fft,
            hop_length,
            win_length,
            use_tensorflow=use_tensorflow,
        )
    )
)
if verbose:

    if ch==1:
        plot_reduction_steps1(
            noise_stft_db,
            mean_freq_noise,
            std_freq_noise,
            noise_thresh,
            smoothing_filter,
            sig_stft_db,
```

```
        sig_mask,
        recovered_spec,
    )
if ch==2:
    plot_reduction_steps2(
        noise_stft_db,
        mean_freq_noise,
        std_freq_noise,
        noise_thresh,
        smoothing_filter,
        sig_stft_db,
        sig_mask,
        recovered_spec,
    )
if ch==3:
    plot_reduction_steps3(
        noise_stft_db,
        mean_freq_noise,
        std_freq_noise,
        noise_thresh,
        smoothing_filter,
        sig_stft_db,
        sig_mask,
        recovered_spec,
    )
if ch==4:
    plot_reduction_steps4(
        noise_stft_db,
        mean_freq_noise,
        std_freq_noise,
```

```
            noise_thresh,
            smoothing_filter,
            sig_stft_db,
              sig_mask,
              recovered_spec,
           )
    return recovered_signal
```

```
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ ТВз7111_21Б 12-3
import matplotlib.pyplot as plt
import numpy as np
# from test1 import main_fun

def plot_spectrogram1(signal, title):
    fig, ax = plt.subplots(figsize=(10, 4))
    cax = ax.matshow(
        signal,
        origin="lower",
        aspect="auto",
        cmap=plt.cm.afmhot,
        vmin=-1 * np.max(np.abs(signal)),
        vmax=np.max(np.abs(signal)),
    )
    fig.colorbar(cax)
    s = 1
    ax.set_title(title)
    # plt.tight_layout()
    # plt.show()
    fig.savefig(title + '__1_.png', format='png')
```

```
def plot_spectrogram2(signal, title):
    fig, ax = plt.subplots(figsize=(10, 4))
    cax = ax.matshow(
        signal,
        origin="lower",
        aspect="auto",
        cmap=plt.cm.afmhot,
        vmin=-1 * np.max(np.abs(signal)),
        vmax=np.max(np.abs(signal)),
    )
    fig.colorbar(cax)
    s = 1
    ax.set_title(title)
    # plt.tight_layout()
    # plt.show()
    fig.savefig(title + '__2_.png', format='png')

def plot_spectrogram3(signal, title):
    fig, ax = plt.subplots(figsize=(10, 4))
    cax = ax.matshow(
        signal,
        origin="lower",
        aspect="auto",
        cmap=plt.cm.afmhot,
        vmin=-1 * np.max(np.abs(signal)),
        vmax=np.max(np.abs(signal)),
    )
    fig.colorbar(cax)
```

```
    s = 1
    ax.set_title(title)
    # plt.tight_layout()
    # plt.show()
    fig.savefig(title + '__3_.png', format='png')

def plot_spectrogram4(signal, title):
    fig, ax = plt.subplots(figsize=(10, 4))
    cax = ax.matshow(
        signal,
        origin="lower",
        aspect="auto",
        cmap=plt.cm.afmhot,
        vmin=-1 * np.max(np.abs(signal)),
        vmax=np.max(np.abs(signal)),
    )
    fig.colorbar(cax)
    s = 1
    ax.set_title(title)
    # plt.tight_layout()
    # plt.show()
    fig.savefig(title + '__4_.png', format='png')
```

ДОДАТОК В

Автоматизована система адаптивної фільтрації гідроакустичних сигналів на основі машинного навчання

Опис програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ ТВз7111_21Б 13-1

Аркушів 8

Київ — 2021

АНОТАЦІЯ

Програмна система надає можливість виконувати фільтрацію гідроакустичних сигналів. У ході написання програмної системи було реалізовано графічний користувацький інтерфейс для легкої взаємодії з нею. Використовуючи програму, користувач може фільтрувати .dat-файл гідроакустичного сигналу і зберегти дані фільтрації на локальному комп’ютері.

Після завершення фільтрації автоматично виконується візуалізація сигналу та виводиться детальна інформація про сигнал, а саме, спектрограми, осцилограми та інші дані до та після фільтрації гідроакустичного сигналу. Ці дані дають можливість оцінити, наскільки успішно було вилучено шум з сигналу, та ознайомитись з новими параметрами фільтрованого сигналу. Також у кінці фільтрації виводиться результат відношення сиглал/шум до та після фільтрації, що надає змогу побачити, який канал сигналу і в скільки разів було знешумлено.

Програмну систему створено в середовищі розробки Visual Studio Code і Jupyter Notebook з використанням мови програмування Python та її бібліотек.

ЗМІСТ

-1-

ЗАГАЛЬНІ ВІДОМОСТІ

Розроблений програмний продукт складається з основного файлу, написаного мовою програмування Python, а також двох модулів Noisereduce і Plotting, які в свою чергу забезпечують фільтрацію сигналу та вивід інформації.

Графічний інтерфейс запускається в головному Python-процесі. Усі вікна візуалізації виконуються за допомогою модифікованого модуля Plotting і виводяться в графічний інтерфейс користувача, що є дуже зручним для аналізу сигналу після фільтрації.

Програмна система надає можливість адаптивно фільтрувати гідроакустичні сигнали, та відображати їх у вигляді осцилограми, спектру і спектрограм.

-2-

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Створена програмна система розроблена для адаптивної фільтрації гідроакустичних сигналів. Вона може бути використана для дослідження гідроакустичних сигналів, дослідження Світового океану та для військових цілей з метою розпізнавання надводних і підводних об’єктів.

-3-

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Розроблена програмна система складається з кількох компонентів:

— Основний додаток, що виконується в головному процесі Python;

— Вікна візуалізації даних, що виконуються в модулі Python, процес візуалізації починається після відкриття файлу сигналу для фільтрації через графічний інтерфей та запуску фільтрації.

— Компонент для фільтрації гідроакустичного сигналу.

Взаємодія всіх вказаних вище компонентів між собою надає можливість коректного виконання програми.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для використання програмного продукту необхідний комп’ютер з підтримкою віртуалізації, достатні ресурси для розгортання та запуску одночасно декількох процесів мови Python та наявність інтерпретатора мови Python.

-5-

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними програмної системи є .dat-файли гідроакустичних сигналів.

Вихідні дані представляють собою текстову, графічну інформацію про гідроакустичний сигнал і фільтраваний сигнал в формані .dat-файлу.

ДОДАТОК Г

Автоматизована система адаптивної фільтрації гідроакустичних сигналів на основі машинного навчання

АПРОБАЦІЯ

Матеріали XIX Міжнародної науково-практичної конференції молодих вчених і студентів 2021 року,

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ ТВз7111_21Б

Аркушів 6

2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

СУЧАСНІ ПРОБЛЕМИ НАУКОВОГО ЗАБЕЗПЕЧЕННЯ ЕНЕРГЕТИКИ

Матеріали XIX Міжнародної
науково-практичної конференції
молодих вчених і студентів
м. Київ, 20–23 квітня 2021 року

ТОМ 2

Київ- 2021

ПОКАЖЧИК АВТОРІВ ДОКЛАДІВ

УДК 004.855.5:519.25

Студент 4 курсу, гр. ТВ-з71 Козак О.А.
Доц., к.т.н. Кублій Л.І.

АВТОМАТИЗОВАНА СИСТЕМА АДАПТИВНОЇ ФІЛЬТРАЦІЇ ГІДРОАКУСТИЧНИХ СИГНАЛІВ НА ОСНОВІ МАШИННОГО НАВЧАННЯ

Гідроакустика має широке практичне застосування — її використовують для підводної локації, зв'язку, океанологічних досліджень, розв'язання військових задач. Звук дуже якісно поширюється у воді, тому його можна чути і виявляти на великих відстанях. На сьогоднішній день гідроакустика зіштовхується з проблемою розробки теорій, які описують поширення звуку в морській воді, а також створення адекватних математичних моделей, що пояснюють експериментальні сигнали. Можливим розв'язанням цієї проблеми є створення автоматизованої системи адаптивної фільтрації гідроакустичних сигналів на основі машинного навчання. Також одним із напрямків розв'язання поставлених перед гідроакустикою задач є використання вейвлетів — математичних функцій, які дають можливість аналізувати різні частотні компоненти даних, зокрема, дають можливість врахувати прив'язку звуку до часу.

Вейвлет-перетворення є потужним інструментом для обробки сигналів, стискання даних і зменшення шуму. Більше того, вейвлет-перетворення починають конкурувати з основною математичною технікою в аналізі сигналів — перетворенням Фур'є.

Перетворення Фур'є [1] на даний час є одним з основних механізмів обробки гідроакустичних сигналів. Це інтегральне перетворення однієї комплекснозначної функції дійсної змінної на іншу. Воно розкладає дану функцію на осциляторні функції. Його використовують для того, щоб розрахувати спектр частот для сигналів, змінних у часі.

У дипломному проекті застосовується швидке перетворення Фур'є (FFT) [2], алгоритм якого є модифікацією звичайного перетворення Фур'є з застосуванням методу "розділяй і володарюй". Цей метод полягає в рекурсивному розбитті розв'язуваної задачі на дві підзадачі того ж типу, але меншого розміру і розбиття виконуються доти, поки всі підзадачі не стануть елементарними. Застосування швидкого перетворення Фур'є зробило можливою цифрову обробку сигналів. При цьому швидкість аналізу даних оцінюється як $O(n\log_2 n)$, у той час, як звичайне перетворення Фур'є потребує $O(n^2)$ часу. За допомогою швидкого аналізу Фур'є ціла сигнальна форма радарного відлуння (ехо) може зберігатися як невеликий набір даних. Ці дані можуть використовуватися для ідентифікації радіолокаційних цілей.

Проте перетворення Фур'є дає інформацію тільки про присутню в сигналі частоту, і не дає ніякої інформації про те, в який проміжок часу ця частота присутня. Таким чином, перетворення Фур'є не може відрізнити стаціонарний сигнал від нестаціонарного, що є значною проблемою при його застосуванні.

Відмінною особливістю вейвлет-аналізу [3] є те, що в ньому можна використовувати сімейства функцій, які реалізують різні варіанти співвідношення невизначеності. Відповідно, користувач має можливість гнучкого вибору між ними і можливість застосування тих вейвлетних функцій, які найбільш ефективно розв'язують поставлені завдання. Таким чином, враховуючи багатоканальність гідроакустичного сигналу і результати, які можна отримати після їхнього вейвлет-аналізу, можна визначити напрямок руху і відстань до об'єкта.

Вейвлет-перетворення відкривають новий спосіб розуміння і дослідження сигналів. Саме тому розроблена програмна система базується на вейвлет-перетворенні. Вона пропонує користувачеві завантажити аудіо-файл гідроакустичного сигналу в форматі .rad і виконує його фільтрацію за допомогою дискретного вейвлет-перетворення.

У розробленій програмній системі застосовується машинне навчання [4]. При цьому одним із основних завдань є навчання машинної моделі. Машинне навчання є спрощеною версією навчання людини. Як правило, в машинному навчанні наявний певний набір прикладів, спостережень, реакцій до цих спостережень. Задача полягає в тому, щоб сконструювати такі моделі, які будуть максимально ефективно описувати наявні дані і робити достовірні прогнози.

У галузі аналізу даних машинне навчання є методом, який застосовують для побудови складних моделей та алгоритмів для прогнозування. Такі аналітичні моделі дають можливість дослідникам, науковцям з обробки даних, інженерам і аналітикам "виробляти надійні, повторювані рішення та результати" і розкривати "приховані розуміння" шляхом навчання з часових співвідношень і тенденцій в даних [5].

Для створення автоматизованої системи адаптивної фільтрації на основі машинного навчання сформульовано завдання, які визначили логіку і структуру дослідження:

— навчитися фільтрувати сигнали. Параметри фільтра мають вибиратися адаптивно відносно деяких критеріїв з використанням машинного навчання;

— розробити структуру класів програмного забезпечення;

— реалізувати застосунок для фільтрації гідроакустичних сигналів на основі машинного навчання.

При реалізації автоматизованої системи адаптивної фільтрації гідроакустичних сигналів на основі машинного навчання застосовано:

— швидке перетворення Фур'є, за допомогою якого визначається амплітуда і частота отриманого сингалу;

— вейвлет-перетворення для фільтрації сигналу;

— машинне навчання, щоб програма могла адаптивно вибирати параметри фільтра відносно деяких критеріїв. Наприклад, одним з таких критеріїв може бути стабільність фази певної частоти.

Програмне забезпечення реалізовано з використанням інтерпретованої об'єктно-орієнтованої мови програмування Python. Однією з використовуваних бібліотек є SciPy — бібліотека Python з відкритим вихідним кодом, призначена для розв'язування наукових і математичних проблем. Вона побудована на базі оптимізованої бібліотеки роботи з багатовимірними масивами NumPy і дає можливість керувати даними, а також візуалізувати. У свою чергу в цій бібліотеці розміщується бібліотека Matplotlib, однією з функцій якої є візуалізація результатів двовимірною графікою.

Розроблена автоматизована програмна система забезпечує фільтрацію гідроакустичних сигналів. Параметри фільтра вибираються адаптивно відносно заданих критеріїв з використанням машинного навчання. Система може використовуватися гідроакустичними станціями виявлення сигналів, що входять до складу гідроакустичних комплексів вимірювання й аналізу параметрів сигналів, метою яких є виявлення джерел випромінювання і їхня класифікація.

Перелік посилань:

1. An Interactive Guide To The Fourier Transform. — https://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/

2. Дасгупта С., Пападимитриу Х., Вазирани У. Алгоритмы. — М.: МЦНМО, 2014. — 320 с. (С. 61-74).

3. Приложения вейвлет-анализа. — https://basegroup.ru/community/articles/wavelet-applications

4. Плас Дж. Вандер. Python для сложных задач: наука о данных и машинное обучение. — СПб: Питер, 2018. — 576 с.

5. Dickson Ben. Exploiting machine learning in cybersecurity. — https://techcrunch.com/2016/07/01/exploiting-machine-learning-in-cybersecurity/