Київський національний університет імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики
Кафедра моделювання складних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
на здобуття ступеня бакалавра
за спеціальністю 113 «Прикладна математика»

на тему:

Цифрова обробка сигналів: методи і їх порівняння

студента 4 курсу
Владислава ЖИТНИКА

Науковий керівник:
доцент, доктор фізико-математичних наук
Андрій ШАТИРКО

Робота заслухана на засіданні кафедри моделювання складних систем
та рекомендована до захисту в ЕК, протокол № ..10.... від ..05.06.23
2023 р.

Завідувач кафедри МСС доктор. техн. наук, доц.
Дмитро ЧЕРНІЙ

Київ – 2023

Анотація

Випускна кваліфікаційна бакалаврська робота: 43 сторінки, 29 рисунків, 1 таблиця, 3 додатки, 12 інформаційних джерел.
Тема: Цифрова обробка сигналів: методи і їх порівняння.
Об'єкт дослідження: набір цифрових сигналів взятих з відкритих джерел і способів їх обробки.
Мета роботи: дослідження цифрових способів і методів обробки сигналів.
Предмет дослідження: реалізація програмного забезпечення для аналізу FM-сигналу і перетворення його на звук.
Результати дослідження:

- проведено дослідження в галузі цифрової обробки сигналів.
- розглянуто види сигналів, їх застосування.
- описано повний ланцюжок генерації сигналу $->$ модуляції $->$ передачі $->$ прийому $->$ демодуляції $->$ декодування.
- реалізовано програму для цифрової обробки аналогових і цифрових сигналів.
- розглянуто протокол RDS що використовується для кодування FM-радіо.

ЦИФРОВА ОБРОБКА СИГНАЛІВ, ЦИФРОВІ ФІЛЬТРИ, МОДУЛЯЦІЯ, ПЕРЕТВОРЕННЯ ФУР'Є, ОБРОБКА РАДІО СИГНАЛІВ.

Abstract

Graduation qualifying master's thesis: 43 pages, 29 images, 1 table, 3 applications, 12 sources.
Topic: Digital signal processing: methods and their comparison.
Object of study: a set of digital signals and methods of their processing.
The goal of the work: conduct research of digital methods and methods of signal processing.
Subject of study: implementation of software for analyzing the FM signal and converting it to sound.
Results of the research:

- conducted research in the field of digital signal processing.
- types of signals and their application are considered.
- the complete chain of signal generation $->$ modulation $->$ transmission $->$ reception $->$ demodulation $->$ decoding is described.
- a program for digital processing of analog and digital signals has been implemented.
- the RDS protocol used for FM radio encoding is described.

DIGITAL SIGNALS PROCESSING, DIGITAL FILTERS, MODULATION, FOURIER TRANSFORM, RADIO SIGNALS PROCESSING.

Зміст

Розділ 1

Загальні питання цифрової обробки сигналів

1.1 Сигнали

На відміну від прийнятого уявлення сигнал - це залежність якоїсь величини від іншої. Наприклад, залежність напруги змінного струму від часу, або ж залежність густини гірської породи відносно відстані до поверхні Землі. Найчастіше в якості незалежності змінної береться час і ця змінна позначається латинською літерою t. Якщо не вказано інакше, залежна змінна називається амплітудою і позначається латинською літерою x.

Існує два типи сигналів: Аналоговий і Цифровий. Аналоговий сигнал неперервно змінюється відносно своєї незалежної змінної, а цифровий - дискретно. Аналогові сигнали зазвичай трапляються у природі і фізичних явищах, а цифрові - в кібернетиці.

Цифрова обробка сигналів (Digital Signal Processing, DSP *англ.*) - це метод обробки сигналів, який використовує математичні та логічні операції для отримання корисного навантаження з отриманого сигналу.

Ключовим розділом DSP є цифрова фільтрація сигналів, тобто покращення амплітудно-фазових характеристик бажаного сигналу або придушення небажаних.

Сигнали в побутових пристроях

Електромагнітні сигнали буквально оточують людей в повсякденному житті, проте залишаються невидимими. Стільниковий звязок в мобільних телефонах, Wi-fi, та навіть мікрохвильова піч генерують різки коливання електричної напруги і за допомогою антени випромінюють ці коливання в навколишній світ. В багатьох людей народженних після 2000 року в дитинстві була радіокерована машина. Низькі затрати на виробництво, і відповідно низька ціна, в поєднанні з високою інтерактивністю зробили цю іграшку доволі популярною в свій час. Отже, маємо стандартну схему двох керованих елементів, один з яких може лише передавати сигнал, а інший, відповідно, його приймає. Почнемо з пульта управління.

Рис. 1.1: Фото плати пульта управління

На фото можна помітити необхідний мінімум радіодеталей. Чотири кнопки для отримання вхідних даних від користувача, котушка індуктивності, резистор, для генерації несучого сигналу з заданою частотою, LED-індикатор передачі сигналу і антена. З іншої сторони друкованої плати присутня мікросхема з позначенням TX-2. Конкретно для даної моделі я не зміг знайти публічно-доступний даташит з описом функціоналу, проте враховуючи маркування можу припустити що функціонал даної інтегральної мікросхеми нічим не відрізняється від передавачів що використовують протол TX-2/RX-2 для кодування сигналу.[5][6]

Схема приймача виглядає більш складно, оскільки процес де-модуляції є складнішим. Проте, якщо прибрати функціонал відділення несучого сигналу від модулюючого, отримаємо просту схему, в якій на вхід мікросхеми подається інформаційний сигнал, де за допомогою тригерів

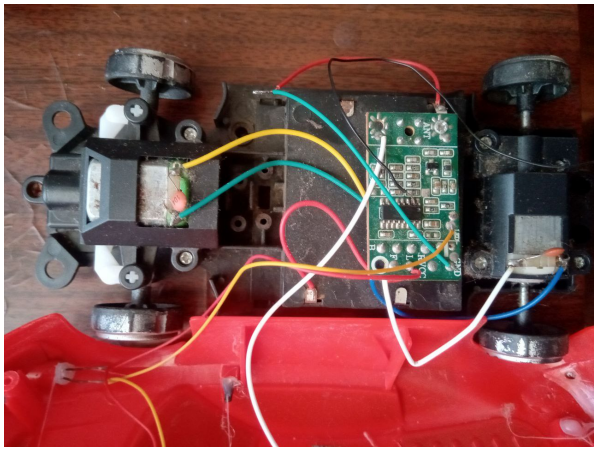

Рис. 1.2: Фото плати керованої машини

і програмованих логічних послідовностей визначається одна із чотирьох комманд: рух вперед, рух назад, поворот вправо, поворот вліво. Варто зазначити, що у приймачі, як і передавачі, частота несучої хвилі контролюється за допомогою резистору підєднаного до вбудованого осцилятора.

Окрему увагу заслуговує протокол передачі даних Tx2-Rx2. Хоча якогось офіційно затвердженого стандарту не існує, проте всі мікросхеми, в документації яких зазначено такий протокол керуються схожими принципами і можуть мати схожу будову друкованої плати. Наприклад, для передачі інформації використовуються дві функції сигналів: W1 і W2. W2 є дискретним сигналом з частотою в 500Hz і 75% робочого циклу при широтно-імпульсній модуляції і серія із чотирьох послідовних таких сигналів використовується для інформування процесора про зміну команди виконання. Далі йде сигнал W1 з частотою 1kHz і 50% робочого циклу, і їх кількість позначає наступний режим роботи. Наприклад, якщо мікропроцесор отримав 10 сигналів W1 після чотирьох W2, то керуючий пристрій повинен їхати вперед. Якщо ж сигналів W1 було 58 - повернули на ліво.

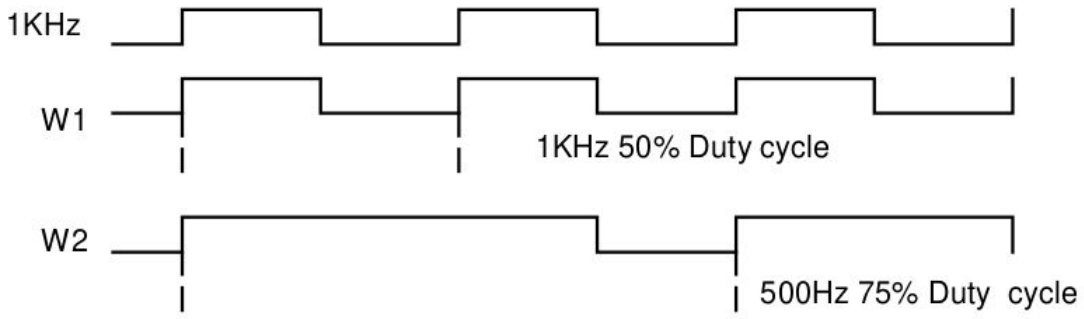

Рис. 1.3: Схема кодування сигналу у протоколі Tx2-Rx2

1.2 Дискретизація та квантування

Загальні відомості

Розглянемо загальний вигляд циклу роботи системи цифрової обробки сигналів

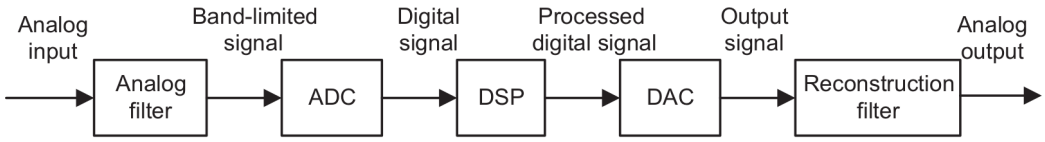

Рис. 1.4: Схема систем для цифрової обробки

Як бачимо, аналоговий сигнал подається на вхід до аналового фільтру, звідки отримуємо сигнал з бажаними амплітудно-частотними характеристиками. Далі, сигнал направляється в аналогово-цифровий перетворювач(АЦП). Далі, АЦП розбиває сигнал на проміжки, для кожного проміжку проводить квантизацію і перетворює квантовий рівень в цифровий.

Як було сказано раніше, дані, що передаються за допомогою аналогового сигналу - неперервні. Отже, мають нескінченну кількість точок. Для їх обробки потрібно була б нескінченна пам'ять та нескінченний час, що в реальному світі неможливо зробити. Проте, можна використати неперервність і обчислити параметри сигналу з мінімальною похибкою. Нехай існує неперервний сигнал $x(t)$ і квантовий сигнал $y(t)$, що має вигляд:

$$y(t) = x(t - (t \, mod \, T))$$

де T - період квантування.

Для спрощеного представлення, $y(n)$ можна переписати наступним чином:

$$y(t) = x(nT), \quad t \in [nT, (n+1)T)$$

При всіх $n \in (0, 1, ...)$. Графічно зображується:

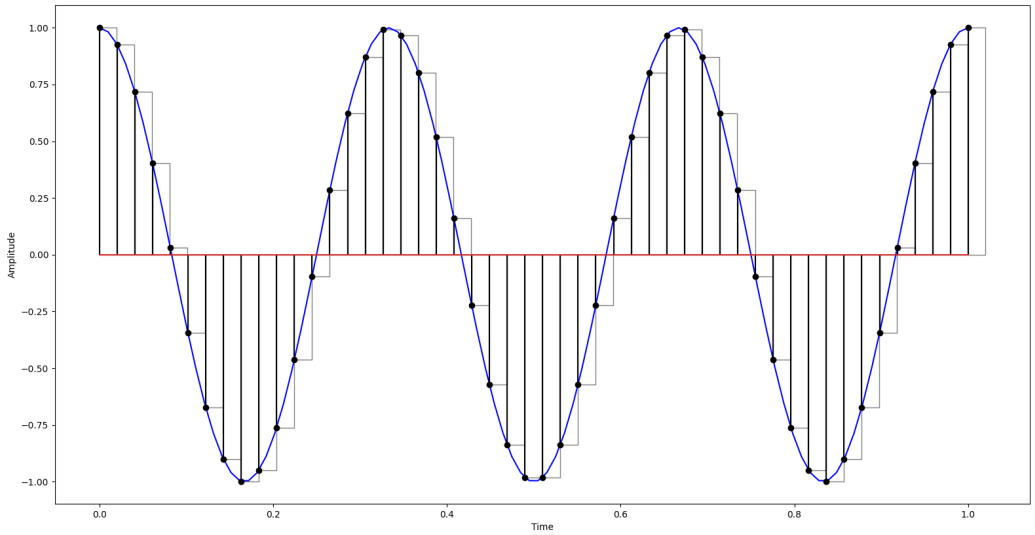

Рис. 1.5: Аналоговий сигнал x(t) і квантований y(t)

Тоді, виконується наступна рівність:

$$\lim_{T \to 0} y(t) = x(t)$$

Іншими словами, при достатньо малих проміжках періодах квантування функція $y(t)$ наближає $x(t)$. Проте, в багатьох випадках надмірна точність є надлишковою і частоту розбиття на проміжки можна зменшити без втрати інформації, якщо наперед знати деякі характеристики цільового сигналу. В наступних розділах буде розглянуто як знайти мінімально-допустиму швидкість розбиття для нормальної роботи АПЦ.

Квадратурна квантизація

Термін “квадратура” має багато значень в різних констекстних областях, проте у випадку DSP і особливо роботи з SRD це означає дві синусоїдні хвилі відхилені одна від одної на 90 градусів. В загальному випадку цей кут називається фазою. Чому саме 90 градусів? В такому випадку дві синусоїдні функції будуть формувати пару ортогональних функцій, що в свою чергу приносить багато можливостей, наприклад

бути базисом над полем функцій. Для простоти написання і подальшого розуміння, зазвичай одну з функцій записують як синус, а іншу - косинус. Як і для звичайних гармонічних сигналів, амплітуда коливань є одним з найбільш важливих параметрів. Історично, параметр при косинусі позначають літерою I (оскільки в англомовних джерелах дана компонента називається "in-phase", тобто є свого роду відліком фази), тоді як при синусі - Q (оскільки називається "quadrature"). Тобто пара буде виглядати як:

$$Icos(2\pi ft)$$

$$Qsin(2\pi ft)$$

IQ квантування набагато більш зрозуміле якщо його сприймати з погляду передавача сигналу. Перед нами постає завдання: найбільш ефективний, з погляду затрачених ресурсів, часу, обладнання способом передати сигнал через повітря до цілі. Для цього необхідно не лише змінювати амплітуду, а з цим у сучасних системах проблем нема, оскільки рівень амплітуду це те ж саме що й рівень електричної напруги, під'єднаної до даної ділянки кола, а також і фазу, що вже не є примітивною задачею. Проте, для виконання поставленої задачі, достатньо знати шкільний курс математики, а саме одну з так званих тригонометричних тотожностей:

$$acos(x) + bsin(x) = Asin(x + \varphi)$$

де, параметри A і φ можна отримати з наступних рівностей:

$$A^2 = a^2 + b^2$$

$$sin(\varphi) = b/A$$

$$tg(\varphi) = b/a$$

В результаті маємо: використовуючи лише два вхідних сигнали зі сталою фазою на виході можливо отримати сигнал зі змінною фазою і амплітудою, що можна використати для одного чи іншого методу модуляції. Для цього необхідно згенерувати два синусоїдальних сигнали з різницею фаз у 90 градусів, збільшити або ж зменшити амплітуду кожного з них до відповідного значення і додати два сигнали. Дані операції легко реалізуються на електричних схемах, використовуючи лише базові елеметни такі як резистори, транзистори та конденсатори

Квадратури і комплексні числа

За допомогою IQ позначень можна задавати магнітуду і фазу, що логічно наштовхує на думку про комплексні числа. Комплексне число зазвичай відображається як одинична точка на декартовому графіку, де ціла частина відповідає відстані від початку координат до проекції на вісь абсцис, а уявна - від початку координат до проекції на вісь ординат. Проте в цьому випадку зручніше уявляти комплексне число як вектор від початку координат до даної точки. Нехай цей вектор має довжину l, відхиляється від додатнього напрямку осі абсцис на кут φ і позначає комплексне число $a + bj$. Тоді справедливі наступні рівності:

$$l = sqrt(a^2 + b^2)$$

$$\varphi = atrctan(b/a)$$

Як бачимо, ці рівності доволі схожі на ті, що задані в попередньому пункті. З цього можна провести паралелі між IQ позначенням і параметрами комплексного числа: I відповідає дійсній частині комплексного числа, а Q - уявній. Оскільки електромагнітний сигнал передається і приймається як раціональне число, то важливо розуміти, що використання комплексних чисел зводиться до відображення отриманих даних і, відповідно, подальшої обробки.

1.3 Попередня підготовка сигналу

Перед тим як користувач (у вигляді ядра операційної системи або ж звичайної десктопної програми) отримає масив з дискретизованими значеннями сигналів, над цим же сигналом відбувається багато дій зі сторони приймаючою мікросхеми. Якщо ж подавати на вхід сигнал без обробки то одразу виникнуть декілька труднощів. По-перше, потрібен дуже дорогий набір апаратного забезпечення: аналогово-цифровий перетворювач, який зможе проводити операцію квантування і дискретизації з вдвічі вищою частотою ніж частота цільвого сигналу, процесор який здатен передавати сигнали між пристроями в рази більшою частотою. До слова, максимальна частота сучасних процесорів нового покоління - 2-5GHz, тобто вони вже не могли б обробляти стільниковий звязок покоління 5G. По-друге, якщо виникне потреба зберегти сигнал для майбутнього аналізу, за лічені секунди вільного місця на диску не залишиться. Саме тому, для різних цілей і задач, існує три канонічних види архітектури приймача:

- **Пряме квантування:** Процес квантизації проводиться напряму, без додаткової обробки. Цей процес потребує дороговартісного устаткування або ж вхідного сигналу на низьких частотах (що в свою чергу збільшує необхідну довжину антени з необхідною резонуючою частотою, оскільки для прямих антен ідеальна довжина обраховується з рівняння: $l = \frac{c}{f}$, де c - швидкість світла, а f - необхідна резонуюча частота).

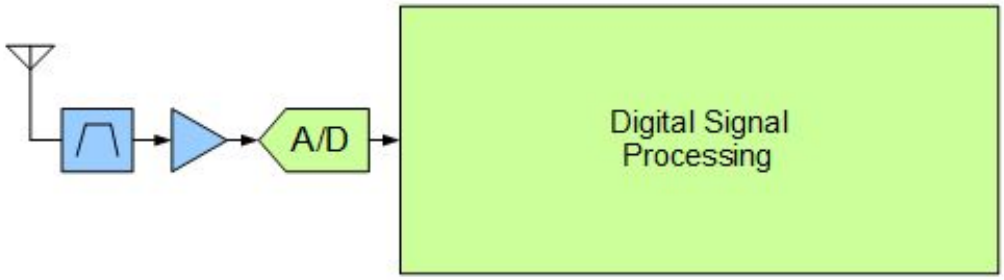

Рис. 1.6: Схема роботи приймача прямого квантування

- **Пряме перетворення:** Процес квантизації проводиться за допомогою локального осцилятора і двох міксерів, що поєднуть сигнал, згенерований осцилятором з отриманим сигналом. Це дозволяє зсунути вліво спектр отриманого сигналу, тобто якщо осцилятор працює на частоті f_o, то вхідний сигнал з частотою $f_o + \delta$ перетворюється в сигнал з частотою δ, а $f_o - \delta$ в $-\delta$. Це дозволяє проводити квантизацію з частотою яка дорівнює ширині сигналу. На виході отримують два сигнали в IQ нотації.

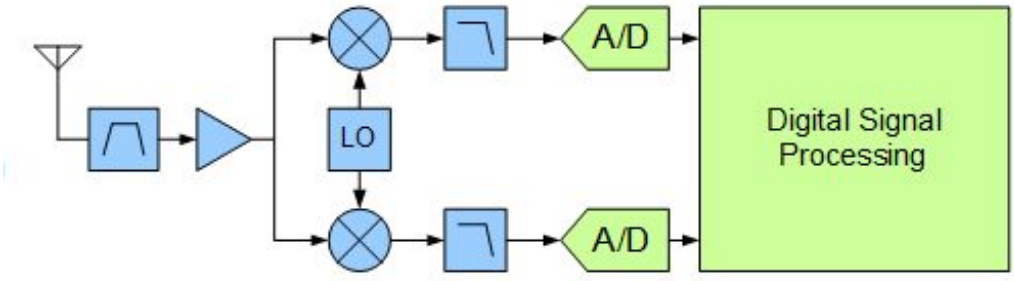

Рис. 1.7: Схема роботи приймача прямого перетворення

- **Супергетеродинний радіоприймач:** Принцип роботи схожий до прямого перетворення, проте вхідний сигнал зміщується не

до нуля, а до якоїсь заданої проміжної частоти. Оскільки весь спектр цільового сигналу знаходиться у додатній півосі, то відсутня потреба в уявних числах при квантизації. Використовується майже в усіх сучасних радіо-приймачах.

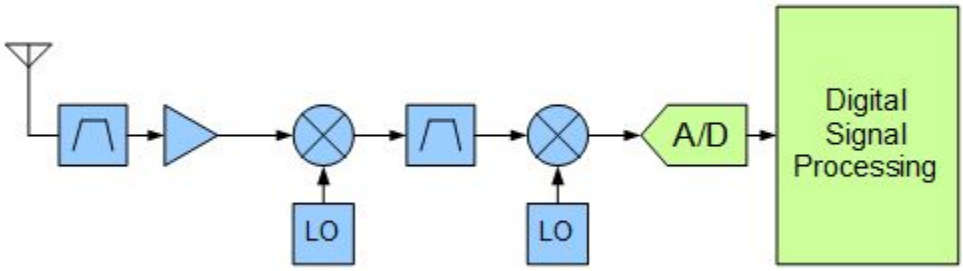

Рис. 1.8: Схема роботи супергетеродинного приймача

(Дані зображення взято з сайту *panoradio-sdr.de*)

1.4 Модуляція і демодуляція сигналу

Будь-який обмін інформацією у електричних системах (включаючи телекомунікаційні мережі, інтернет, радіостанції) відбувається за рахунок сигналів. В межах одного пристрою інформаціє закодовується і передається в рамках одного електричного ланцюга у вигляді зміни напруги. В сучасних системах, найбільше сигналів генерується і опрацьовується саме в інтегральних мікросхемах (англ. integrated circuit, в багатьох випадках використовується скорочення IC). В загальному випадку, форма і вигляд цих сигналів може бути довільною, оскільки не регламентується ні фізичними параметрами ні державними законами.

Трохи інша ситуація для випадку передачі інформації з одного пристрою на інший. В першу чергу це зумовлено фізичними властивостями електромагнітних хвиль, що виражається в спроможності передавача безперешкодно передати потрібну інформацію на необхідну відстань. Окрім цього, викристання радіочастот в багатьох країнах регулюються нормовимо-правовими актами, що регламентують розподіл смуг радіочастот відповідними радіослужбами і розподіляють визначені смуги радіочастот на частоти загального і спеціального призначення[1].

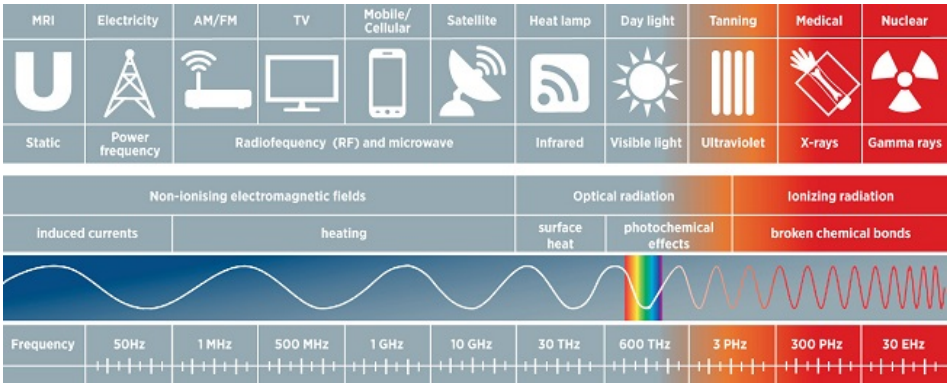

Рис. 1.9: Спектр електромагнітних коливань

Для того щоб приймач зміг правильно розкодувати сигнал, потрібно щоб між ним і передавачем існувала наперед встановлена система обробки сигналу. Перетворення інформації в сигнал називається модуляція, відповідно зворотний процес має назву демодуляція. Існує два види модуляції - Аналогова і Цифрова. Серед аналогових модуляцій найбільш поширеними є амплітудна модуляція, частотна модуляція і фазова модуляція. Серед дискретних - ASK (у вікіпедії перекладено як "амплітудна маніпуляція", мабуть варто якось краще назвати), OOK, FSK.

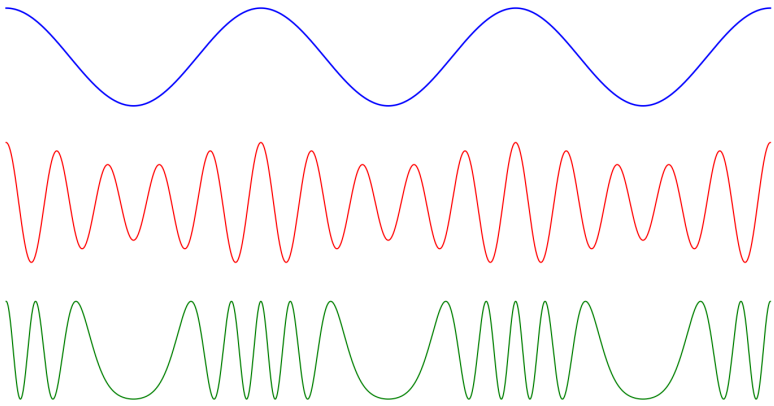

Рис. 1.10: Типи аналогової модуляції. Зверху вниз:
- Оригінальний сигнал
- Використання AM-модуляції
- Використання FM-модуляції

1.5 Перетворення Фур'є

У часовій області, чигнал задається своїми значеннями амплітуди у відношенні до часу або порядкового номеру. Проте, в деяких випад-

ках важливіше мати інформацію про частотну складову сигналу, ніж часову. Тобто необхідне відображення чигналу в частотній області, що називається спектром.

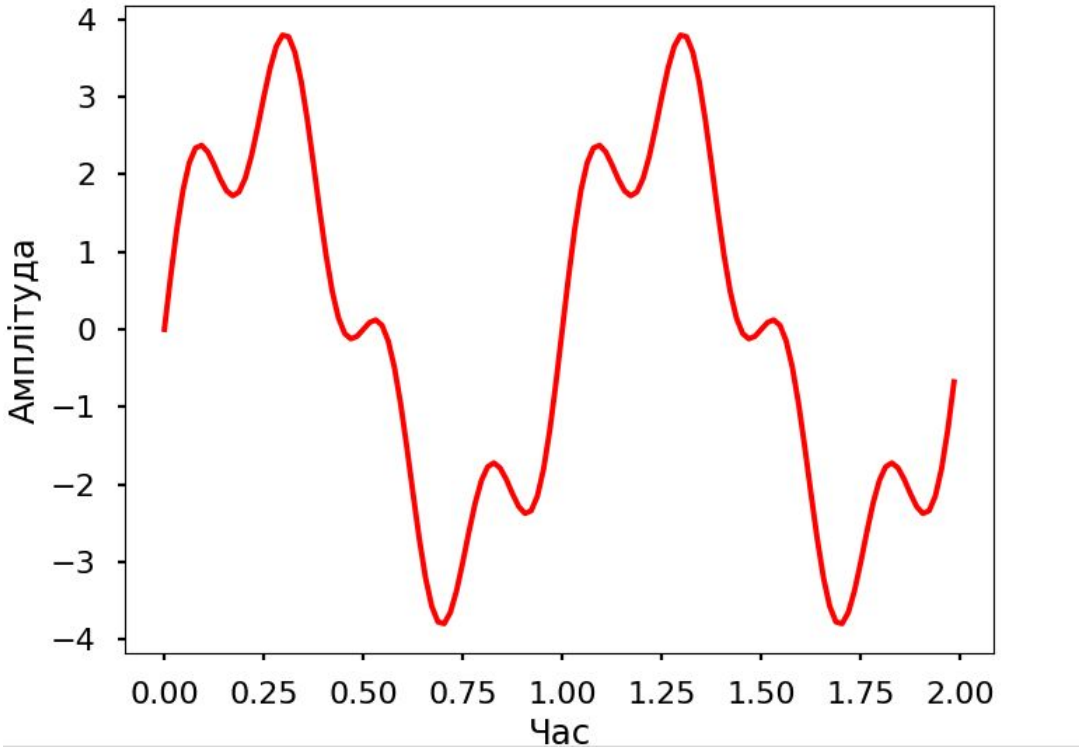

Рис. 1.11: Сигнал, поданий в звичайному вигляді

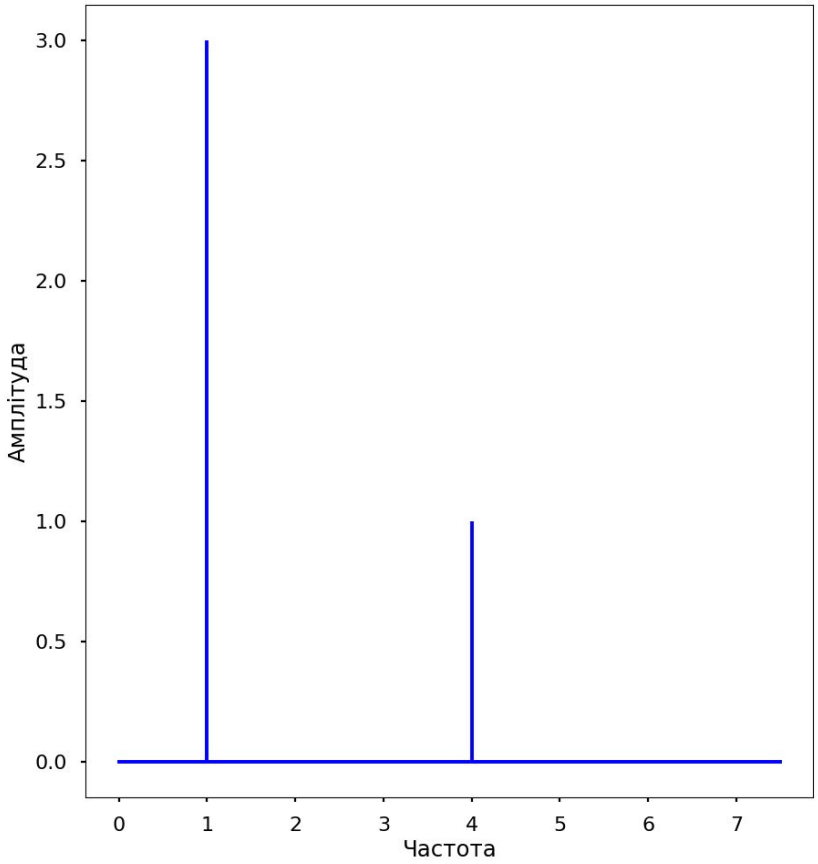

Рис. 1.12: Сигнал, поданий в частотному представленні

Алгоритм, що перетворює значення дискретного сигналу в частотну область називають дискретним перетворенням Фурьє або ж DFT (від англ. Discrete Fourier transform). DFT широко застосовується у спектральному аналізі, аудіо/фото/відео-обробці і системах комунікації.

DFT в своїй основі, як не дивно, використовує розклад періодичної функції в ряд Фур'є в комплексній формі. Отже, маємо формулу для ряду Фур'є:

$$f(x) = \sum_{n=-\inf}^{\inf} c_n * exp(i2\pi x \frac{n}{N}) \tag{1.5.1}$$

, де T - відома імовірна частота функції f(x), $c_k = \frac{1}{T} \int_T f(x) exp(-i2\pi t \frac{k}{T})\, dt$ - коефіцієнти Фур'є

Оскільки маємо справу з дискретно заданою функцією, замінимо інтегрування сумою Дарбу. Для цього необхідно знати частоту дискретизації f_0. Також, підставимо значення часового проміжку дискретизації T_0 ($T_0 = 1/f_0 = T/N$) замість періоду T в формулу 1.5.1.

Отримаємо:

$$c_k = \frac{1}{N}\sum_{n=0}^{N-1} x(n)exp(-i2\pi k\frac{n}{N}), \quad -\infty < k < \infty$$

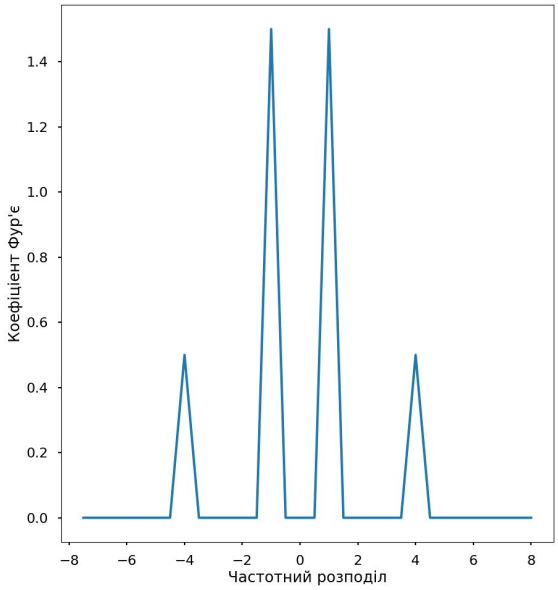

Рис. 1.13: Амплітудний спектр гармонічного сигналу

Легко побачити що даний вираз є періодичним, з періодом N, тобто $c_{k+N} = c_k$. Важливо розуміти, що межі частотного спектру лінійно залежать від вибраної частоти дискретизації. Більше того, Найквіст у своїй роботі "Certain Topics in Telegraph Transmission Theory" [2] доводить, що для дискретизації гармонічного сигналу, максимальна вагома частота якого є F_{max} необхідно обрати частоту дискретизації $f_0 > f_{max} = 2F_{max}$ у 1924 році. Пізніше Клауд Шенон у своїй роботі "Communication in the Presence of Noise" чітко сформував умови теореми, яка зараз відома під назвою "Теорема квантування":

If a function f(t) contains no frequencies higher than W cps, it is completely determined by giving its ordinates at a series of points spaced 1/2 W seconds apart

де W - це кутова частота, а f(t) - вхідний сигнал. Зазвичай F_{max} називають частотою Найквіста. Також частота $f_0/2$ називається частотою згортання (folding frequency) оскільки функція $|c_k|$ є симетричною відносно даної точки. Якщо на вхід DFT подається дискретизований сигнал що складається лише з дійсних чисел, то ліва частина (тобто

спектральний проміжок $[-f_0/2, 0)$ буде повністю симетричним відносно правого. Даний факт дає нам змогу в деяких випадках розглядати амплітудний спектр сигналу лише на проміжку $[0, f_0/2]$

Проте, на практиці коефіцієнти Фур'є c_k при обробці сигналів не використовуються, натомість використовуються так звані коефіцієнти DFT що являють собою:

$$X(k) \ = \ Nc_k \ = \ \sum_{n=0}^{N-1} x(n) exp(-i2\pi k \frac{n}{N}) \tag{1.5.2}$$

У випадку ряду Фур'є представлена не лише формула розкладу функції на коефіцієнти Фур'є, а й відповідна, тобто обернена до неї, формула знаходження функції за заданими коефіцієнтами 1.5.1. Як не дивно, обернена формула існує і для алгоритму DFT (і має назву inverse DFT):

$$x(n) \ = \ \frac{1}{N} \sum_{k=0}^{N-1} X(k) exp(i2\pi k \frac{n}{N})$$

Амплітудний і фазовий спектр

Як можна помітити з виразу 1.5.2, кожен член $X(k)$ є числом комплексним, а отже його можна представити у вигляді суми дійсної і уявної частини, наприклад:

$$X(k) \ = \ R(k) + iI(k)$$

де R(k) і I(k) - дійсна і уявна частина.

Тоді магнітуду і зсув (їх ще називають амплітудний і фазовий спектр відповідно) для кожного коефіцієнту DFT можна знайти за допомогою наступних формул:

$$A(k) = |X(k)| = \sqrt{R(k)^2 + I(k)^2}$$

$$\theta(k) = tan^{-1} \frac{R(k)}{I(k)}$$

Враховуючи симетричність частотного розподілу відносно точки початку координат, у багатьох випадках є сенс записати амплітудний спектр лише для проміжку $[0, f_0/2]$. Таким чином, зміниться і формула для обрахунку:

$$\bar{A}(k) = \begin{cases} |X(k)|, & k == 0 \\ 2|X(k)|, & k = 1, 2, ..., N/2 \end{cases}$$

1.6 Швидке перетворення Фур'є

Проаналізуємо формулу 1.5.2 на асимптотику. Для кожного значення k (із проміжку $\overline{0, N}$) потрібно виконати N операцій додавання, отже загалом маємо $O(N^2)$ операцій в нотації Ландау, що доволі багато. Проте, за рахунок ділення сум на дві частини, можливо оптимізувати алгоритм до $O(N * log(N))$. Даний алгоритм отримав назву Швидке Перетворення Фур'є (Fast Fourier Transform, FFT).

Хоча алгоритмів котрі можуть називатись FFT, а отже оптимізують DFT до $O(N * log(N))$, існує декілька, в даній роботі викладено лише найбільш поширений, а саме метод Кулі-Тьюка [4].

В оригінальній роботі автори розглядають можливість оптимізації DFT за умови, коли N можна представити як добуток цілих чисел, тобто $N = r_1 \cdot r_2 \cdot ...r_m$. В такому випадку асимптотична кількість операцій буде $O(N \cdot (r_1 + r_2 + ... + r_m))$. Якщо ж припустити що $r_1 = r_2 = ... = r_m = r$ то $N = r^m$ і, відповідно, асимптотика буде мати значення $O(r \cdot N \cdot m) = O(r \cdot N \cdot log_r N)$. За таких умов значення $r = 3$ є формально найбільш ефективним вибором (оскільки дає найбільш вигідне співвідношення складності обчислень до кількості вхідних даних) проте використання $r = 2$ дає змогу зекономити кількість використаної оперативної памяті або ж використовувати операції бінарного множення. На практиці, багато реалізованих алгоритмів потребують на вхід точки дискретизації, кількість яких буде степенем двійки.

1.7 Z-перетворення

z-перетворення відіграє дуже важливу роль у описі архітектури і аналізі цифрових систем. Воно також використовується для побудови цифрових фільтрів ічастотному аналізі цифрових сигналів. Но для початку потрібне чітке визначення. Z-перетворення дискретної послідовності x(n) позначається сивмолом Z(x(n)) і визначається наступною формулою:

$$Z(x(n)) = \sum_{n=0}^{\inf} x(n) * z^{-n} = x(0) * z^{-0} + x(1) * z^{-1} + \cdots$$

де z - компелксне число. Сумування проводиться від $n = 0$ до $n = \inf$ оскільки логічно припустиит, що дискретний сигнал $x(n) = 0$ для $n < 0$. Оскільки маємо справу з нескінченною сумою, то очевидно що в багатьох випадках дана сума буде прямувати до нескінченності. Для

подальшого аналізу найбільшу цінність мають ті значення z, при яких дана сума збігається до якогось числа. Така множина значення z називається *областю збіжності*. Очевидно, що ця область залежить від розглядуваної послідовності x(n). Нижче наведено декілька прикладів z-перетворення і області збіжності для елементарних сигналів:

x(n), n>0	z-перетворення, Z(x(n))	Область збіжності
$\delta(n)$	1	$\|z\| > 0$
a*u(n)	$\frac{a * z}{z - 1}$	$\|z\| > 1$
n*u(n)	$\frac{z}{(z-1)^2}$	$\|z\| > 1$
$n^2 * u(n)$	$\frac{z * (z+1)}{(z-1)^3}$	$\|z\| > 1$
$a^n * u(n)$	$\frac{z}{z-a}$	$\|z\| > a$

де символами $\delta(n)$, u(n), a позначені дельта функція, функція Хевісайда та довільна константа відиінна від нуля, відповідно.

Властивості z-перетворення

Властивості z-перетворення широко використовуються при отриманні z-перетворюваних функцій диференціальних рівнянь і розв'язку задач на вихід лінійних цифрових систем з сталими коефіцієнтами, що буде розглянуто в наступному пункті.

- **Лінійність:** z-перетворення є лінійним перетворенням, що означає
$$Z(ax_1(n) + bx_2(n)) = aZ(x_1(n)) + bZ(x_2(n))$$
де $x_1(n)$ і $x_2(n)$ - дискретні послідовності, а a і b - довільні константи

- **Зсув за часом:** для кожного z-перетворення Z(x(n)) існує z-перетворення x(n-m) таке що:
$$Z(x(n-m)) = z^{-m}Z(x(n))$$
, що називається затримкою або зсувом за часом. Дана властивість відіграє важливу роль у знаходженні перехідної функції з диференціальних рівнянь.

- **Згортка:** маючи дві послідовності $x_1(n)$ та $x_2(n)$, їхня згортка може бути визначена як:
$$x(n) = x_1(n) * x_2(n) = \sum_{k=0}^{\inf} x_1(n-k)x_2(k)$$
де * позначає лінійну згортку. В області z-перетворення маємо наступну формулу:
$$Z(x(n)) = Z(x_1(n))Z(x_2(n))$$

Зворотнє z-перетворення

Пряме z-перетворення дискретної послідовності x(n) і зворотнє z-перетворення функції Z(x(n)) позначаються відповідно:

$$X(z) = Z(x(n))$$

$$x(n) = Z^{-1}(X(z))$$

де Z() - це оператор z-перетворення, і $Z^{-1}()$ - зворотній оператор. Він може бути отриманий як мінімум трьома методами:

- **1.** Розкладом на прості дроби і подальшого пошуку значень у таблиці 1.7

- **2.** Розкладом степеневого ряду

- **3.** Методом невизначених коефіцієнтів

Перший випадок є найбільш вживаним і за загальними рисами не відрізняється від сетоду розкладу на прості дроби у методі для перетворення Лапласа. Розглянемо один приклад з використанням даного методу:

Приклад:

Нехай $X(z) = \dfrac{10z}{z^2 - z + 1}$

Потрібно знайти зворотнє z-перетворення

Розвязання:

Ввведемо нову змінну: довільну константу a

Тоді, цю константу можна підібрати таким чином, щоб виконувалась наступна рівність:

$$\frac{10z}{z^2 - z + 1} = \left(\frac{10}{sin(a)}\right) \frac{sin(a)z}{z^2 - 2zcos(a) + 1} \tag{1.7.1}$$

В лівій частині $sin(a)$ може приймати довільні, відмінні від нуля, значення з множини раціональних чисел. В той же час, щоб рівність виконувалась потрібно щоб:

$$-2cos(a) = -1$$

Звідси отримуємо:

$$cos(a) = 0.5$$

$$a = 60^\circ \text{ або } a = \frac{\pi}{3}$$

Підставляючи дані значення у ліву частину 1.7.1 отримаємо:

$$x(n) = \frac{10}{sin(a)} Z^{-1}\left(\frac{sin(a)z}{z^2 - 2zcos(a) + 1}\right) = \frac{10}{0.8666} sin\left(\frac{\pi n}{3}\right) = 11.547 sin\left(\frac{\pi n}{3}\right)$$

Використання z-перетворення для розвязку різницевих рівнянь

Для розв'язку різницевих рівнянь з заданими початковими значеннями, необхідно оперувати зсунутими за часом послідовностями y(n-

1), y(n-2), y(n-3), ..., y(n-m) і так далі. Використовуючи означення z-перетворення, маємо:

$$Z(y(n-1)) = \sum_{n=0}^{\inf} y(n-1)z^{-n}$$

$$= y(-1) + y(0)z^{-1} + y(1)z^{-2} + ...$$
$$= y(n-1) + z^{-1}(y(0) + y(1)z^{-1} + y(2)z^{-2} + ...)$$

Для простоти запису замінимо дужки в лівій частині на Y(z) Отримаємо:

$$Z(y(n-1)) = y(-1) + z^{-1}Y(z)$$

Аналогічний принцип для n-2:

$$Z(y(n-2)) = \sum_{n=0}^{\inf} y(n-2)z^{-n}$$

$$= y(-2) + y(-1)z^{-1} + y(0)z^{-2} + y(1)z^{-3} + ...$$
$$= y(-2) + y(-1)z^{-1} + z^{-2}(y(0) + y(1)z^{-1} + y(2)z^{-2} + ...)$$
$$Z(y(n-2)) = y(-2) + y(-1)z^{-1} + z^{-2}Y(z)$$
$$Z(y(n-m)) = y(-m0 + y(-m+1)z^{-1} + ... + y(-1)z^{-(m-1)} + z^{-m}Y(z)$$

де y(-m), y(-m+1), ..., y(-1) - початкові умови. Якщо ж припустити що всі початкові умови рівні нулю, тоді отримаємо:

$$Z(y(n-m)) = z^{-m}Y(z)$$

Що є аналогічним до властивості z-перетворень зсіві за часом. Загалом процедура знаходження розв'язку різницевих рівнянь описується наступними тезами:

- **1.** Застосувати z-перетворення до різницевого рівняння
- **2.** Відняти початкові умови
- **3.** Розв'язати рівняння в z-області
- **4.** Застосувати обернене z-перетворення до розв'язку

Розділ 2

Фільтри

2.1 Теорія цифрових фільтрів

2.1.1 Використання різницевих рівнянь у цифрових фільтрах

Нехай x(n) та y(n) - вхід та вихід системи ЦОС, відповідно. Зв'язок між входом і виходом можна зобразити за допомогою наступного різницевого рівняння:

$$y(n) = b_0x(n) + b_1x(n-1) + b_2x(n-2) + ... + b_kx(n-k) - \\ - a_1y(n-1) - a_2y(n-2) - ... - a_ly(n-l) \quad (2.1.1)$$

або ж

$$y(n) = \sum_{i=0}^{k} b_ix(n-i) - \sum_{j=1}^{l} a_jy(n-j) \quad (2.1.2)$$

де b_i, a_j - дійсні константи. Тобто, для обрахунку вихідного сигналу на n-му кроці, необхідно мати k попередніх вхідних значень і l попередніх вихідних. Дана система називається цифровим фільтром. Використаємо z-перетворення (X(z) - позначає z-перетворення для сигналу x(n), Y(z) - для сигналу y(n)):

$$Y(z) = b_0X(z) + b_1X(z)z^{-1} + ... + b_kX(z)z^{-k} - a_1Y(z)z^{-1} - ... - a_lY(z)z^{-l}$$

Звідси отримуємо функцію H(z), яка називається передатною функцією

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1z^{-1} + ... + b_kz^{-k}}{1 + a_1z^{-1} + ... + a_lz^{-l}}$$

За допомогою передатної функції можна визначити стабільність і АЧХ даного фільтру. Імпульсна характеристика (тобто реакція фільтру на імпульсний вхідний сигнал):

$$h(n) = Z^{-1}\{H(z)Z\{\delta(n)\}\} = Z^{-1}\{H(z)\}$$

Стрибкова характеристика - реакція фільтру на функцію Хевісайда:

$$X(z) = Z[\sigma(n)] = \frac{z}{z-1}$$

$$y(n) = Z^{-1}\{\frac{zH(z)}{z-1}\}$$

Введемо ще одне позначення. Нехай символом $\angle$ позначається аргумент комплексного числа. Тобто, комплексне число $z = r(cos(\varphi) + isin(\varphi))$ може бути записане як $r\angle\varphi$ або ж $|z|\angle\varphi$.

Для квантованого сигналу існує чіткий звязок між перетворенням Лапласа і z-перетворенням, і описується рівнянням:

$$z = e^{sT}$$

Де s - параметер перетворення Лапласа. Для того щоб оперувати лише раціональними числами доцільно зобразити цей параметр як сукупність дійсного і уявного числа:

$$z = e^{-\alpha T \pm i\omega T} = e^{-\alpha T}\angle \pm \omega T$$

Також, зручно замінити $\Omega = \omega T$. В такому разі, значення магнітуди АЧХ і зсуву по фазі буде відповідно $|H(e^{i\Omega}|$ і $\angle H(e^{i\Omega})$

Приклад:
Нехай задано коефіцієнти фільтру $a_i = [0.5, 0.5]$
Потрібно побудувати АЧХ і зсув по фазі
Розв'язання
Маємо загальний вигляд фільтру і його z-перетворення:

$$y(n) = 0.5x(n) + 0.5x(n-1)$$

$$Y(z) = 0.5X(z) + 0.5z^{-1}X(z)$$

Звідси можливо обчислити перехідну функцію:

$$H(z) = \frac{Y(z)}{X(z)} = 0.5 + 0.5z^{-1}$$

$$H(e^{i\Omega}) = 0.5 + 0.5cos(\Omega) - 0.5isin(\Omega)$$

Тому магнітуда АЧХ і її графік буде мати наступний вигляд:

$$|H(e^{i\Omega}) = \sqrt{(0.5 + 0.5cos(\Omega))^2 + (0.5sin(\Omega)^2)}$$

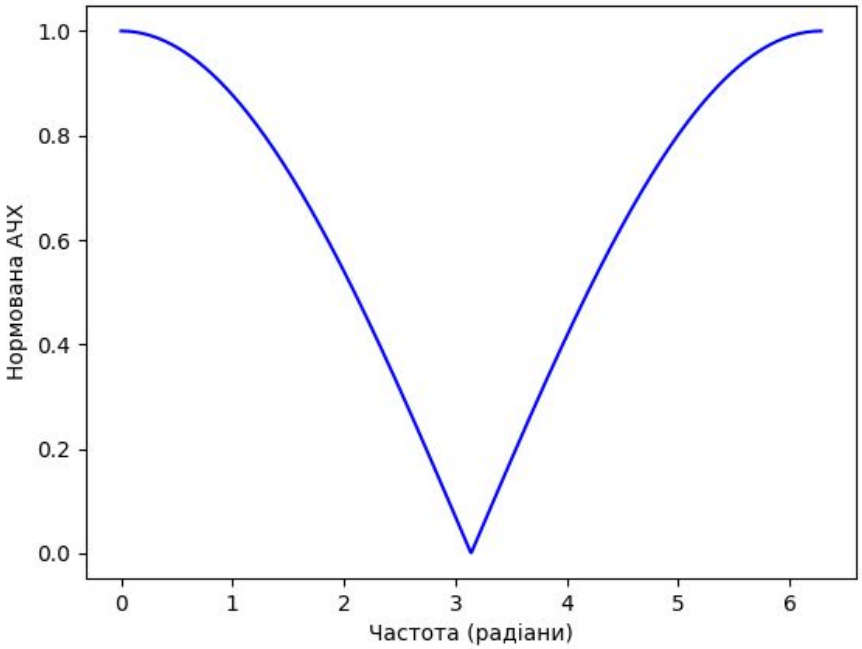

Рис. 2.1: Графік магнітуди АЧХ

Аналогічно для зсуву по фазі:

$$\angle H(e^{i\Omega}) = arctan\left(\frac{-0.5sin(\Omega)}{0.5 + 0.5cos(\Omega)}\right)$$

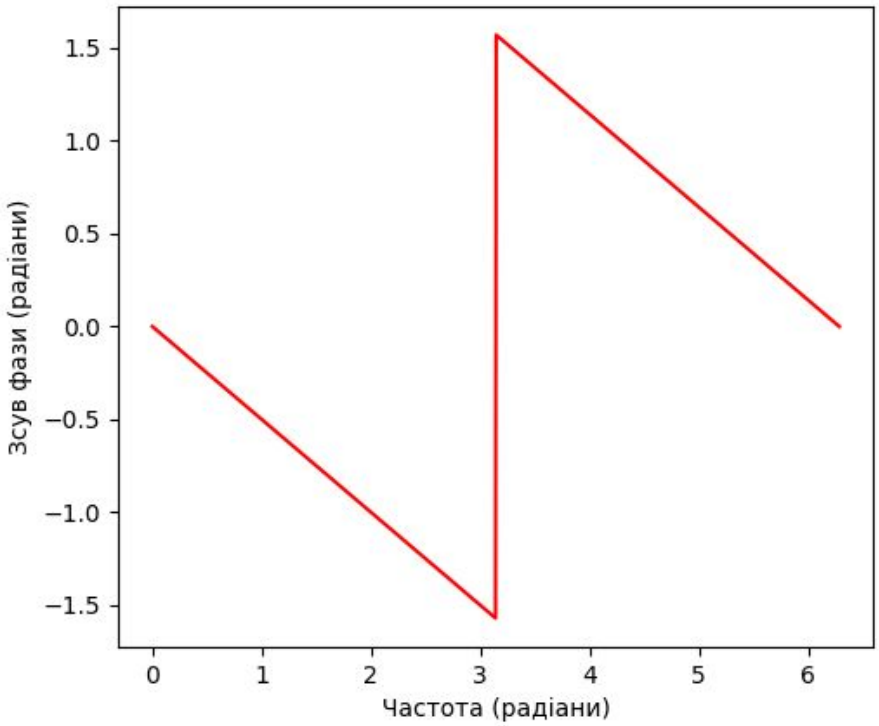

Рис. 2.2: Графік зсуву фази

Проте, не завжди зручно вимірювати відносну частоту в радіанах. Якщо відома частота дискретизації f_s, тоді можливо перевести розподіл АЧХ з радіан в Герци за допомогою наступної формули:

$$f = \frac{f_s \Omega}{2\pi}(Hz) \tag{2.1.3}$$

В той же час, розмірність радіанів частоти використовуються для позначення загальної характеристики фільтру, без конкретної привязки до частоти дискретизації.
Також, на практиці користуються логарифмічним позначенням АЧХ (що вимірюється в dB) замість нормованого коефіцієнту:

$$H = 20\log_{10}(H(e^{i\Omega}))(dB) \tag{2.1.4}$$

Оскільки Ω лежить на проміжку $[0, 2\pi]$, то за формулою 2.1.3 $f \in [0, f_s]$. Проте, при цифровій обробці інформації, фактична робота може виконуватись лише з частотами нижчими за частоту Найквіста, тобто $[0, \frac{f_s}{2}]$. Тому значеннями на $[\pi, 2\pi]$ можна знехтувати.

2.1.2 Різновиди цифрових фільтрів

За видом придушення АЧХ вхідного сигналу фільтри поділяються на:

- Фільтр низьких частот (low-pass filter *англ.*) - вид фільтру який придушує АЧХ з високою частотою, а ті що нижче заданої частоти зрізу - пропускає зазвичай без змін
- Фільтр високих частот (high-pass filter *англ.*) - вид фільтру, який навпаки, придушує сигнали з низькими частотами
- Смуговий фільтр (bandpass filter *англ.*) - вид фільтру, що пропускає частоти, які лежать в певному діапазоні, а решту - придушує. Отримається шляхом одночасного застосування фільтру низьких і високих частот, де частота зрізу ФНЧ більша за частоту зрізу ФВЧ
- Режекторний фільтр (band-reject filter *англ.*) - протилежний до попереднього, придушує АЧХ у певному діапазоні. Використовується для придушення небажаних сигналів, наприклад, у електрокардіографах або при обробці звуку.

Також, за характером різницевого рівняння фільтри поділяють на НкІХ та КІХ.
КІХ (FIR, finite impulse response, *англ.*) - різновид фільтру з скінченною імпульсною характеристикою. Заданий рівнянням 2.1.2 в якому всі $a_j == 0$. В такому випадку, наступне значення вихідного сигналу залежить лише від k попередніх значень вхідного сигналу. Якщо на вхід такого фільтру подати одиничний імпульс, то починаючи з k+1-го значення вихідний сигнал буде дорівнювати нулю. Різницева функція має наступний вигляд:

$$H(z) = \sum_{i=0}^{k} b_i z^{-i}$$

НкІХ (IIR, infite impulse response, *англ.*) - вид фільтру з нескінченною імпульсною характеристикою. Також називається рекурсивними фільтром, оскільки для обрахунку вихідного сигналу використовується як попередні значення вхідного так і вихідного сигналів. На відміну від КІХ фільтрів, може бути не стабільним. Передатна функція виглядає наступним чином:

$$H(z) = \frac{b_0 + b_1 z^{-1} + ... + b_k z^{-k}}{1 + a_1 z^{-1} + ... + a_l z^{-l}}$$

Однією із головних проблем НкІХ фільтрів є округлення чисел з плаваючою точкою, що є результатом послідовних множень. Більш дедально ця проблема розглядалась в наступній праці [7].

При виборі фільтру для обробки дискретизованого сигналу інженери або науковці використовують наступні показники:

- $\boldsymbol{A_{pass}}$ - різниця між максимальним і мінімальним значенням у смузі пропускання
- $\boldsymbol{A_{stop}}$ - різниця між максимальним і мінімальним значенням у смузі подавлення
- $\boldsymbol{F_{pass}}$ - (частота зрізу) - гранична частота смуги пропускання
- $\boldsymbol{F_{stop}}$ - гранична частота для смуги подавлення
- $\boldsymbol{N}$ - кількість доданків необхідних для обчислення вихідного сигналу

Зазвичай, збільшення кількості коефіцієнтів покращує решту характеристик, проте також суттєво збільшує час на обробку сигналу, що може бути неприпустимо у системах що працюють у режимі реального часу

2.2 Покращення характеристик сигналу

При використанні КІХ фільтрів доволі часто на частотному спектрі спостерігається виникнення бічних пелюсток. Воно пов'язане з тим, що кількість квантувань сигналу набагато менша за кількість точок при перетворенні Фур'є. Одним із рішень є збільшити довжину оброблюваного сигналу, проте це збільшить час необхідний на передачу і отримання сигналу. Інший спосіб - це використання віконних функцій.

Наприклад, на вхід подається гармонійний сигнал з частотою 1Hz. При використанні 50 точок квантування частотний розподіл буде виглядати наступним чином:

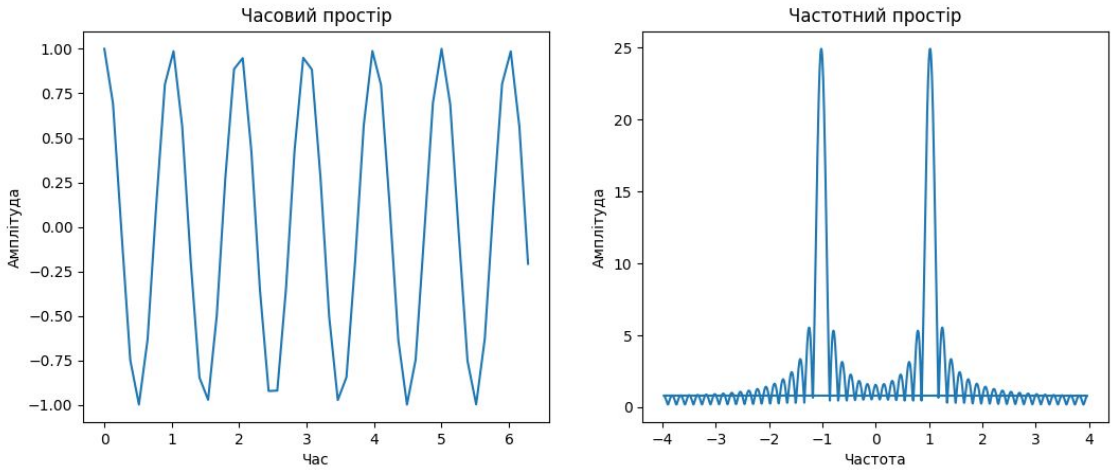

Рис. 2.3: Графік магнітуди АЧХ

Якщо ж використати віконну функцію Блекмена домножити на вхідний сигнал, то отримаємо більш гладкий частотний спектр:

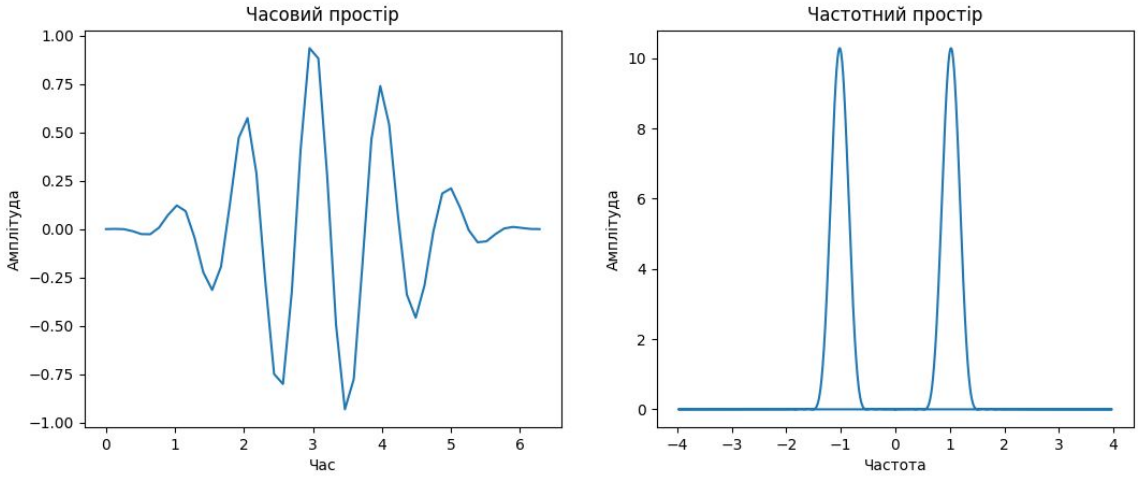

Рис. 2.4: Графік магнітуди АЧХ

2.3 Різновиди цифрових фільтрів та їх застосування

2.3.1 Фільтр рухомого середнього

Є одним з найростіших, а також, одним з найбільш розповсюджених фільтрів. За характером передатної функції належить до класу КІХ фільтрів, за методом придушення АЧХ - фільтр низьких частот. Дуже широко використовується у фінансах, обробці зображень і, також, у

обробці сигналів. Основна задача даного методу полягає у "зглажувані кутів тобто для того щоб зменшити вплив різких, малозначимих сигналів на подальшу обробку. Частотна характеристика фільтру порядку 4 має наступний вигляд:

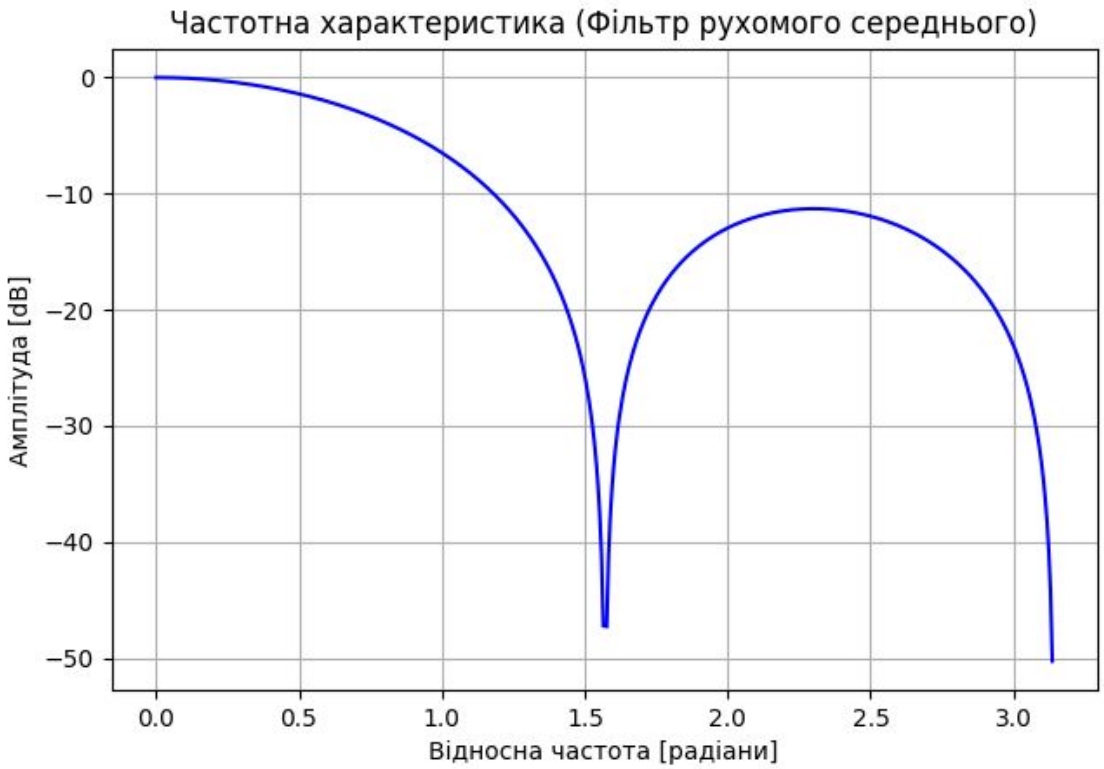

Рис. 2.5: Частотна характеристика

Конкретні приклади використання даного фільтру на практиці можна знайти в уроках програмної обробки зображень (https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html) або ж у проектах з відкритим вихідним кодом. Наприклад, набір програм Klampt використовує рухоме середнє для тестування вхідних даних: https://github.com/krishauser/Klampt/blob/master/Python/klampt/control/blocks/core.py#L717 Більш детально про використання фільтру рухомого середнього у галузі економіки описано в наступній праці: [8]

Переваги: висока швидкість роботи, простота у реалізації

Недоліки: Дуже вузька смуга пропускання, велика різниця між граничною частотою смуги пропускання і смуги затухання, коливання частоти в смузі пропускання.

Висновок: враховуючи частотні характеристики, даний фільтр не варто використовувати в якості фільтра низьких частот. Проте він може бути використаний в ситуації, коли потрібно досягти плавних пе-

реходів між близькими значеннями, наприклад у обробці аудіо або зображень.

2.3.2 Фільтр Баттерворта

Вперше описаний британським інженером і фізиком Стівеном Баттервортом у статті «Про теорію фільтруючих підсилювачів»[9]. За характером передатної функції належить до класу КІХ фільтрів, за методом придушення АЧХ - фільтр низьких частот, проте може виступати у якості фільтру високих частот, смуговим або режекторним. Є найбільш поширеним з усіх НкІХ фільтрів. Частотна характеристика фільтру порядку 4 має наступний вигляд:

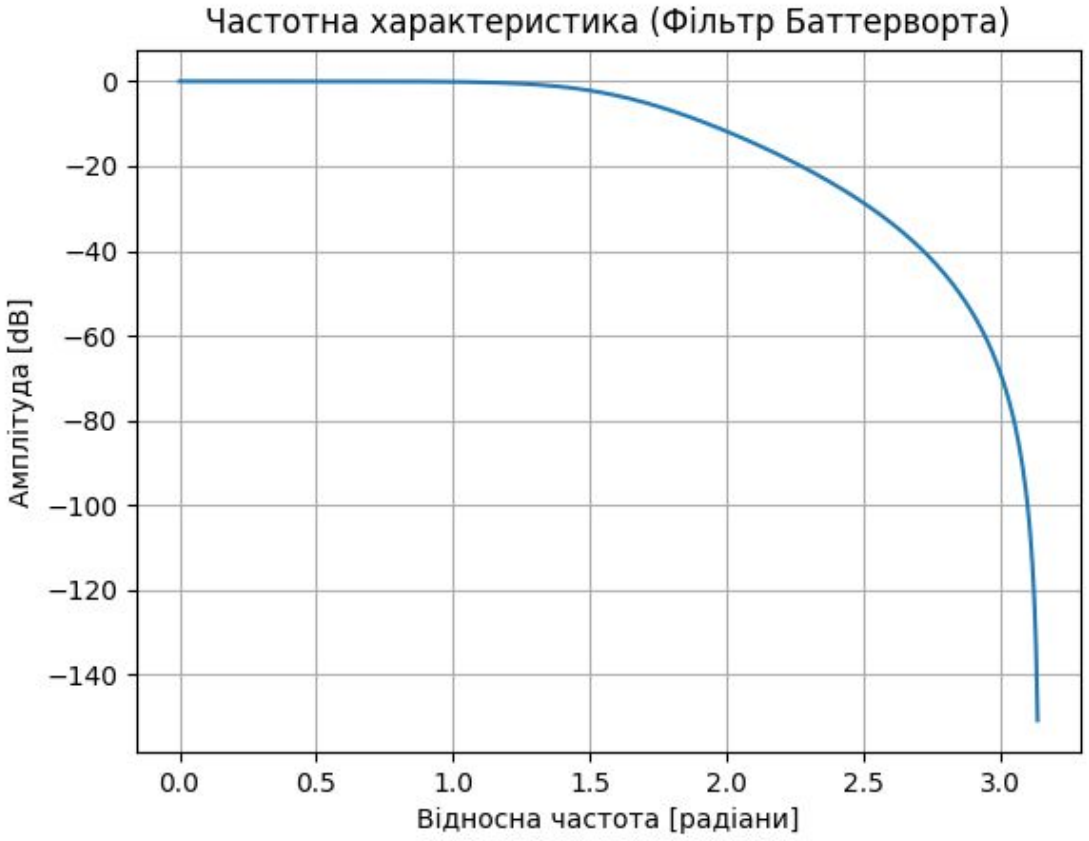

Рис. 2.6: Частотна характеристика

Як видно з графіку, смуга пропускання є дуже плоскою, в той час як перехід від смуги пропускання до смуги затухання є плавним.

У публічно доступних проектах з відкритим вихідним кодом можна знайти багато використань даного фільтру. Наприклад, ось некомерційний проект від компанії Google, що аналізує букви у вимові вико-

ристовує фільтр Баттерворта 2-го роду для обробки аудіо: https://github.com/google/audio-to-tactile/blob/main/src/dsp/butterworth.h#L16, або ж проект від компанії Microsoft, що аналізує ходу людей і може використовуватись для контролю лікування різного роду хвороб опорно-рухового апарату: https://github.com/microsoft/GaitAndBalanceApp/blob/master/Analysis/Trajectory.cs#L288

Переваги: Дуже плоска смуга пропускання

Недоліки: Велика різниця між частотою пропускання і затухання, відсутність смуги пропускання

Висновок: Фільтр Баттерворта повинен бути застосований в тих випадках, коли необхідно зберегти АЧХ і ФЧХ смуги пропускання без змін. Меншої різниці між частотою пропускання і затухання можна досягти шляхом збільшення порядку фільтру.

Розділ 3

Практична частина

3.1 Інструменти

Для практичного виконання задач поставлених даною темою найкраще підходять маніпуляції саме з електромагнітними сигналами. Для досягнення цілі існує два шляхи: генерація і збір сигналів за допомогою електричних приборів (наприклад, осцилятора) або ж симуляція повного циклу електричного кола на спеціалізованих програмних платформах. З найпоширеніших засобів можна виділити наступні:

- SPICE - опенсоурсна консольна програма для симуляції зміни напруги і струму на різних ділянках електричного струму. Дуже часто використовується в якості бекенду для інших графічних програм
- EasyEDA - безкоштовна онлайн платформа для дизайну друкованих плат, побудови електричних кіл та їх симуляції. Однією з особливостей є тісна інтеграція з виробниками деталей, що дозволяє користувачам в декілька кліків замовити необхідні запчастини або ж готову друковану плату. На відмінну від більшості аналогів, має простий, зручний і гарний дизайн.
- GNU Radio - набір для розробки програм з відкритим вихідним кодом. Для побудови ланцюга системи ЦОС використовує функціональні блоки. На вхід може подаватись сигнал з підключених пристроїв або ж з файлу
- мова програмування Python - скриптова мова програмування, дуже зручна для написання невеликих проектів. Містить потрібні

бібліотеки для роботи з великими масивами чисел (numpy), науковими і технічними обчисленнями (scipy), а також для роботи з форматами збереження цифрових сигналів (SigMF)

Враховуючи всі переваги і недоліки перелічених систем для виконання практичної частини я обрав саме мову програмування Python, оскільки саме гнучкість даного інструменту є необхідною умовою при роботі зі складними структурами. Для написання та тестування програмного коду використовувалось інтегроване середовище розробки PyCharm [10].
Також для роботи необхідні дані, над якими проводиться аналіз. Оскільки я не маю SDR приймача, то було прийнято рішення використати вже зібрані дані, які знаходяться в публічному просторі, а саме з URL адреси https://www.sdrplay.com/iq-demo-files/.

3.2 Обробка сигналу

Спектр демодульованого FM-сигналу буде виглядати наступним чином:

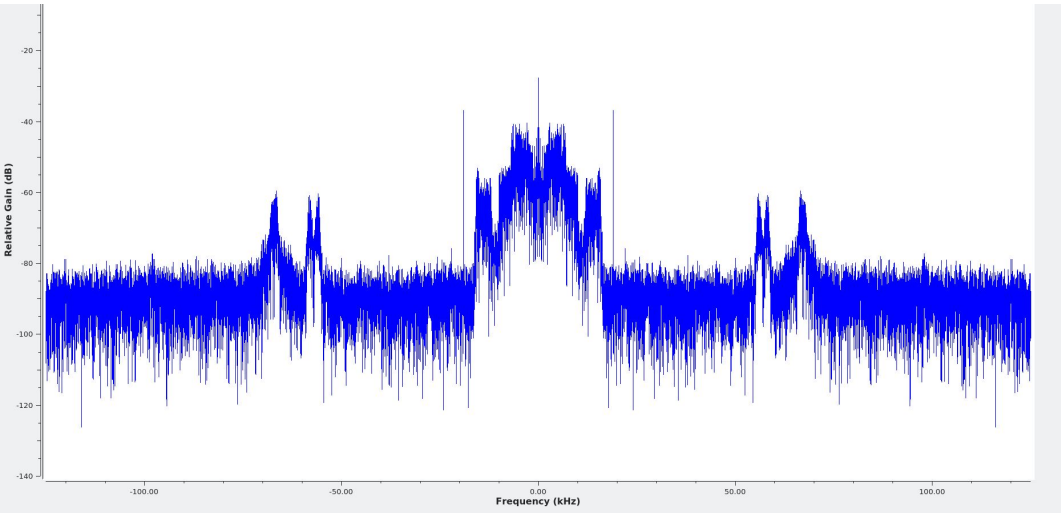

Рис. 3.1: Спектр демодульованого сигналу

На основі цього можна зробити висновок, що FM сигнал складається з декількох сигналів:

- 0-16 kHz - Потужний сигнал
- 19 kHz - Імпульсний сигнал

- 20-54 kHz - Начебто відсутність будь-якого сигналу
- 56 kHz і 58 kHz - Два доволі потужних, но вузьких за спектром сигнали
- 60-73 kHz - Однопелюсковий сигнал

Далі спостереження наочно корелюють з одним із можливих видів смуг FM-сигналу:

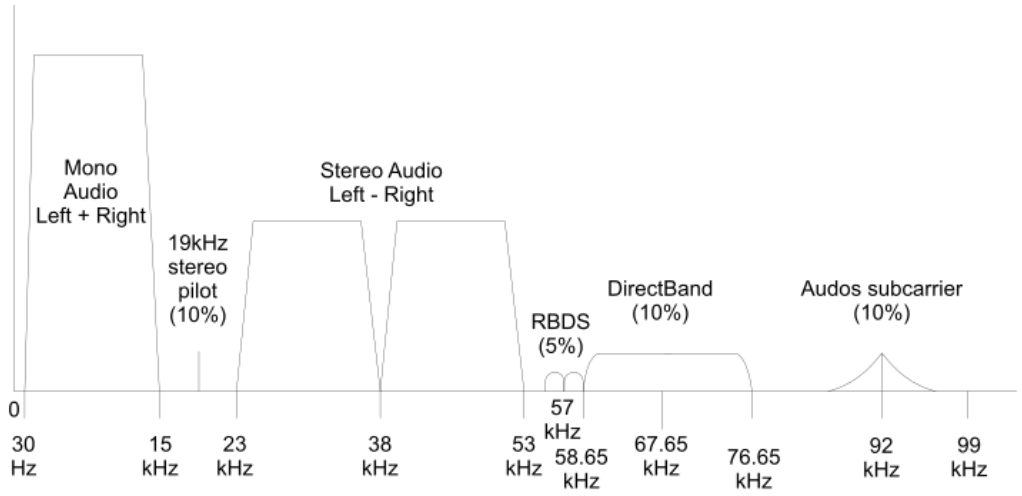

Рис. 3.2: Спектр можливого розміщення сигналів (взято з сайту *wikipedia.en*)

Варто зазначити, що сигнали RBDS та DirectBand не впливають на звучання аудіо сигналу, а лише слугують носіями для метаданих. Як логічний наслідок, не всі FM-радіосигнали мають дані смуги. Імпульсний сигнал використовується для декодування стерео аудіо.

Для демодуляції використовується Квадратурна демодуляція.

3.2.1 RDS протокол

Даний протокол детально описаний в даному документі на 132 сторінки [11].

Інформація передається у формі бітів, зі швидкістю 1187.5 біт/с. Для транспортування використовуються групи розміром 104 біти. Кожна група містить 4 блоки, які, в свою чергу складаються з корисного навантаження в 16 бітів і CRC хеш-суми в 10 бітів. З метою синхронізації, до хеш-суми додано офсети слів A, B, C, D та E з наступними значеннями[12]:

Рис. 3.3: Значення, що використовуються для синхронізації окремих блоків

Далі, двійковий сигнал закодовується методом різниці. До отриманого цифрового сигналу застосовується SRRC фільтр, з метою обрізання зайвих частот, які присутні в цифрових сигналах. Отриманий сигнал модулюється використовуючи бінарну фазову маніпуляцію.

3.2.2 Принцип роботи

Отриманий сигнал збережений у IQ форматі, що був описаний у розділі 1.5. Він демодулюється за допомогою квадратурної демодуляції, що в результаті дає сигнал з дійсними значеннями, спектр якого зображений на 3.1.

Отримання аудіо сигналу

Наступни кроком може бути використання фільтру низьких частот з частотою зрізу в 19kHz, провести процедуру пониження квантування, переведення масиву до типу даних, з яким зможуть працювати більшість аудіо-відтворювачів і запис масиву у файл. Даний код реалізовано в процедурі $sound\_output(...)$

```python
def sound_output(x, srate):
    # plot_waterfall(x, sample_rate=srate, freq_center=0)
    # Filter-out everything except mono audio
    x = si.sosfilt(si.butter(2, 20e3, fs=srate, output="sos"), x)

    # plot_waterfall(x, sample_rate=srate, freq_center=0)

    # downsample by 6 to get mono audio
    x = si.decimate(x, 6)
    sample_rate_audio = srate//6

    # normalize volume so its between -1 and +1
    x /= np.max(np.abs(x))

    # convert from float to int16s
    x *= 32767
    x = x.astype(np.int16)

    wavfile.write('fm.wav', int(sample_rate_audio), x)
    # os.system("aplay fm.wav")
```

Рис. 3.4: Отримання аудіо-запису з демодульованого сигналу

Отримання RDS сигналу

Для початку необхідно виокремити смугу RDS зі всього сигналу. Оскільки смуга не лежить біля нульової частоти, її потрібно змістити. Для цього необхідна властивість перетворення Фур'є зсуву частот. Далі процедура аналогічна попередньому пункту. Даний код реалізовано в функції $get\_rds\_from\_fm(...)$:

```python
def get_rds_from_fm(x, sample_rate):
    # Freq shift to the center of the rds
    N = len(x)
    f_o = -57e3
    t = np.arange(N) / sample_rate
    x = x * np.exp(2j * np.pi * f_o * t)  # down shift
    # plot_waterfall(x, sample_rate, 0)

    x = si.lfilter(si.firwin(numtaps=101, cutoff=7.5e3, fs=sample_rate), [1], x)
    # plot_waterfall(x, sample_rate, 0)

    # Resample to 19kHz
    x = si.resample_poly(x, 19, 250)
    new_sample_rate = 19e3
    # plot_waterfall(x, sample_rate, 0)
    return x, new_sample_rate
```

Рис. 3.5: Код зміщення RDS смуги до нульової частоти

Отримання цифрового сигналу з аналогового

Як було сказано раніше, сигнал кодується за допомогою алгоритму BPSK, що є частковим випадком фазової модуляції. З метою часової синхронізації використовується алгоритм Мюллера-Маллера. Для корекції фази використовується алгоритм Костки. На виході отримано два сузір'я символів, кожен елемент з яких позначає один двійковий символ.

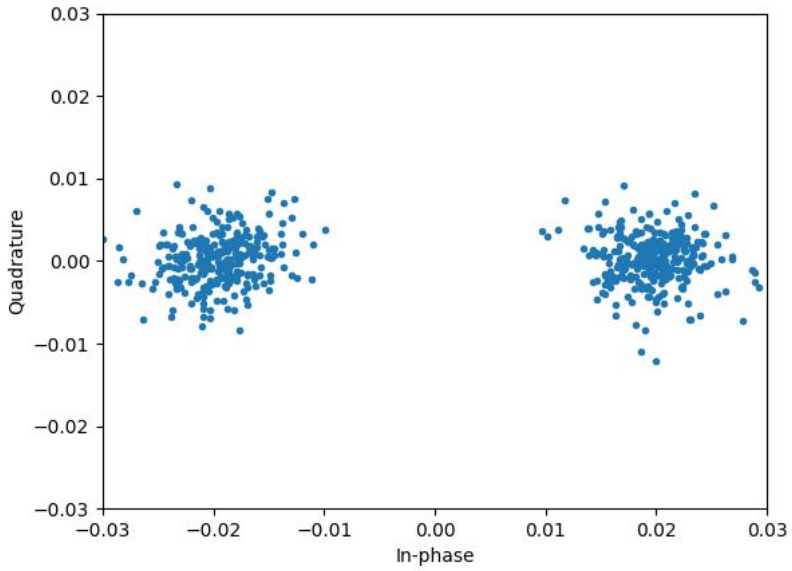

Рис. 3.6: Отримано два чітких сузір'я. Отже можна продовжити подальшу обробку

Якщо ж отримані сузір'я пересікаються між собою, то подальша обробка сигналу неможлива, потрібно вибрати інше значення для параметру mu з алгоритму Мюллера-Маллера.

Синхронізація на рівні блоків

Модифікована версія алгоритму CRC який використовується у протоколі RDS реалізована у функції $calc\_syndrome(...)$. Спочатку необхідно знайти межі будь-якого блоку. Для цього біти по-одному зчитуються в регістр, далі рахується хеш-сума і порівнюється з відомими значеннями сигнатури блоку. Коли встановлено синхронізацію на рівні блоків, необхідно провести ітерацію до того моменту, поки не знайдеться блок A, що є першим в кожній групі. Далі, корисне навантаження з кожного блоку заноситься в вихідний масив. Якщо ж при цьому ви-

никла велика кількість блоків, хеш-сума яких не збігається, необхідно заново провести процедуру синхронізації

```
blocks_counter += 1
if blocks_counter == 50:
    if wrong_blocks_counter > 25:
        # This many wrong blocks must mean we lost sync
        print("Lost Sync (Got ", wrong_blocks_counter, " bad blocks on ", blocks_counter, " total)")
        synced = False
        presync = False
    else:
        pass
    blocks_counter = 0
    wrong_blocks_counter = 0
```

Рис. 3.7: Велика кількість пошкоджених блоків скидає стан програми до моменту синхронізації

Парсинг радіо-тексту

В стандарті прописано декілька типів інформації що може бути передана. Наприклад, інформація про дорожні затори, погоду, альтернативні частоти або ж довільний текст. Оскільки в даних, отриманих з публічних джерел, я не знайшов семплів, що передають якусь іншу інформацію окрім тексту. Тому, мій програмний засіб на даний момент підтримує лише тип радіотексту.

Структура групи даного типу представлена двома варіантами:

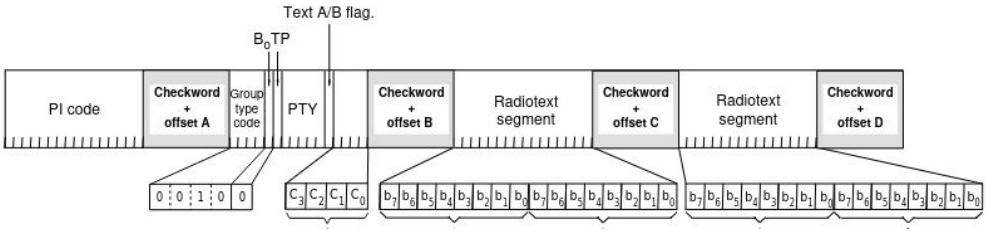

Рис. 3.8: Підтип A

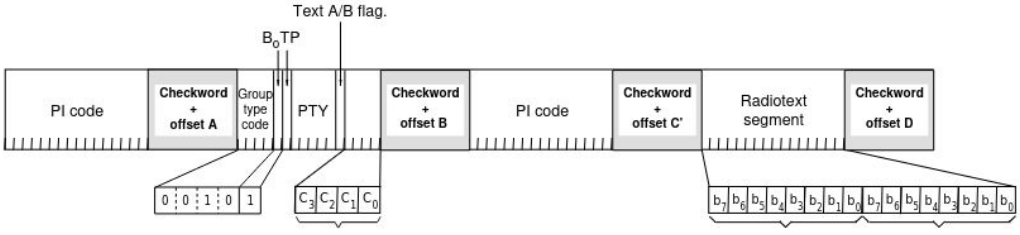

Рис. 3.9: Підтип B

Відповідно до документації, дані поля мають наступний зміст:

- **PI code** - Коди для індентифікації програми. Перші чотири біта визначають країну (Україна має код 0x6), наступні 4 - тип програми за площею передачі (інтернаціональні, національні або регіональні) і останні 8 бітів - тип радіостанції, що визначаються кожною країною окремо.

- **Group Type code** - Тип групи. Для радіо-тексту даний параметр має значення 0x2

- $\mathbf{B_0}$ - Визначає підтип групи. Якщо $\mathbf{B_0} == 0$, то дана група має підтип A, інакше - B

- **PTY** - Тип програми що відтворюється на аудіо-сигналі. Може позначати новини, спорт, культуру, легку музику, джаз, тощо.

- **Text A/B flag** - Визначає спеціальний A/B символ, зміна якого призводить до стирання тексту з екрану

- **Text segment address code** - Визначає адресу тексту, що передається. Для B підтипу 2 байти записуються послідовно за адресою $address * 2$. Для A підтипу передаються 4 байти тексту і записуються вони починаючи з адреси $address * 4$.

Таким чином, розмір буферу в який записується даний текст має 64 байти для підтипу A і 32 байти для підтипу B.

Код для парсингу радіо-тексту реалізовано в функції $parse\_rds\_radiotext(...)$. Варто зазначити, що дана радіостанція для стирання тексту з екрану радіо-приймача використовує не A/B символ, а ASCII символ повернення каретки "\r". Результат виконання програмного комплексу має наступний вигляд:

Рис. 3.10: Результат виконання програми. Синім позначено метадані декодовані з сигналу. Зеленим - радіо-текст, що передавався в поточній групі

Повний текст, що передається протоколом RDS є нічим іншим як гаслом компанії “WayFM Uplifting. Upbeat. Real”

Розділ 4

Висновок

В випускній кваліфікаційній роботі було:

- Розглянуто повний шлях кодування-передачі-прийому-декодування електромагнітних сигналів.
- Створенно шаблонні коди для демонстрації різних аспектів цифрової обробки сигналів, такі як фільтрація, застосування віконної функції, демодуляція.
- Розглянуто мету їх застосування в рельному світі і вплив на вхідний сигнал.
- Описано теорію побудови цифрових фільтрів і використання z-перетворення.

В ході виконання практичної частини даної роботи було написано програмну реалізацію FM-радіо приймача. Дана реалізація дозволяє не лише отримувати, декодувати і відтворювати аудіо сигнал, а й працювати з метаданими. Наприклад, було повністю декодовано текст, що передавався, використовуючи протокол RDS.

Підхід, заснований на програмній реалізації замість апаратної, має більш широкий спектр можливостей. Моя реалізація містить дві ключові особливості: передача довільної інформації з використанням двійкової системи і перевірка цілісності пакетів за допомогою хеш-суми. Перша дозволяє передавати будь-яку числову інформацію, з можливістю її шифрування сучасними алгоритмами. Друга - забезпечує відкидання всіх пакетів даних, які з довільної причини прийшли пошкодженими. Дані критерії є надважливими для громадської, промислової або ж військової галузей.

Перелік використаних джерел

[1] НАЦІОНАЛЬНА ТАБЛИЦЯ розподілу смуг радіочастот України - Режим доступу: https://www.kmu.gov.ua/npas/25976178

[2] H. NYQUIST. Certain Topics in Telegraph Transmission Theory / Harry Nyquist // Transactions of the A.I.E.E. - 1928 - Сторінки - Режим доступу: https://web.archive.org/web/20130926031230/http://www.ieee.org/publications_standards/publications/proceedings/nyquist.pdf

[3] Claude E. Shannon. Communication in the Presence of Noise - 1924 - Режим доступу: https://web.archive.org/web/20100208112344/http://www.stanford.edu/class/ee104/shannonpaper.pdf

[4] Cooley, James W.; Tukey, John W. "An algorithm for the machine calculation of complex Fourier series" - 1965 - Режим доступу: https://www.ams.org/journals/mcom/1965-19-090/S0025-5718-1965-0178586-1/S0025-5718-1965-0178586-1.pdf

[5] datasheet TX-2B - Режим доступу: https://www.digchip.com/datasheets/parts/datasheet/433/TX-2B-pdf.php

[6] AM1139S V1.3 - AM1139S RC car SOP16 All in one controller for 4-CH RC car 2.7V 6.8V 2-channel H-Bridge 3-LEDs control - Режим доступу: https://www.datasheetarchive.com/whats_new/692fc4a213a1a674690ae952a3a2bf06.html

[7] Anastasia Volkova: Towards reliable implementation of digital filters - Режим доступу: https://theses.hal.science/tel-01916214/document

[8] Johan Boissard: Applications and Uses of Digital Filters in Finance - Режим доступу: https://ethz.ch/content/dam/ethz/special-interest/mtec/chair-of-entrepreneurial-risks-dam/documents/dissertation/master%20thesis/MAS_Johan_Boissard_Dec12.pdf

[9] S. Butterwort: On the Theory of Filter Amplifiers - Режим доступу: https://www.changpuak.ch/electronics/downloads/On_the_Theory_of_Filter_Amplifiers.pdf

[10] PyCharm Guide - https://www.jetbrains.com/pycharm/guide/

[11] European Committee for Electrotechnical Standardization: Specification of the radio data system (RDS) for VHF/FM sound broadcasting in the frequency range from 87,5 to 108,0 MHz - Режим доступу: http://www.interactive-radio-system.com/docs/EN50067_RDS_Standard.pdf

[12] READBEST RDS Encoder - Режим доступу: https://www.pira.cz/rds/readbest.pdf

Додаток А

Вихідний код програми, що використовувалась при написанні теоретичної частини

Лістинг А.1: **Теоретична частина**

```
# import numpy as np
# import matplotlib.pyplot as plt
# import scipy.fftpack as fft
#
# n = 64
# t = np.linspace(0, 1, n, endpoint=True)
# ds = 3*np.sin(2*np.pi*t)
#
# tt = np.linspace(0, 1, 32)
#
# plt.plot(t, ds, "-", linewidth=2)
# plt.stem(tt, 3*np.sin(2*np.pi*tt), "o")
#
# plt.show()

from scipy import signal
import matplotlib.pyplot as plt
import numpy as np

# Для параграфу по цифровому перетворенню Фурє'
if 1:
    plt.style.use('seaborn-poster')


    def FFT(x):
        N = len(x)

```

```
28          if N == 1:
29              return x
30          else:
31              X_even = FFT(x[::2])
32              X_odd = FFT(x[1::2])
33              factor = \
34                  np.exp(-2j * np.pi * np.arange(N) / N)
35
36              X = np.concatenate( \
37                  [X_even + factor[:int(N / 2)] * X_odd,
38                   X_even + factor[int(N / 2):] * X_odd])
39              return X
40
41
42      def DFT(x):
43          N = len(x)
44          n = np.arange(N)
45          k = n.reshape((N, 1))
46          # Матриця n на k
47          e = np.exp(-2j * np.pi * k * n / N)
48          # Добуток матриці на стовпчик
49          X = np.dot(e, x)
50
51          return X
52
53      # Частота квантування
54      sr = 16
55      # Інтервал квантування
56      ts = 1.0 / sr
57      t = np.arange(0, 2, ts)
58      # Сигнал - це сума двох синусоїдних з різними амплітудами і частотами
59      freq = 1
60      x = 3 * np.sin(2 * np.pi * freq * t)
61      freq = 4
62      x += np.sin(2 * np.pi * freq * t)
63
64      # Add noise
65      # x += np.random.rand(len(x))
66      # freq = 7
67      # x += 0.5 * np.sin(2 * np.pi * freq * t)
68
69      plt.figure(figsize=(8, 6))
70      plt.plot(t, x, 'r')
71      # plt.stem(t, x, "o")
72      plt.ylabel("Амплітуда")
73      plt.xlabel("Час")
74      plt.show()
75
76
77      X = DFT(x)
78
79      # Обрахунок значень на осі частот
80      N = len(X)
81      n = np.arange(N)
82      T = N/sr
83      freq_axis = n/T - sr/2 + 0.5
```

```
84
85      plt.subplot(121)
86      reform_X = np.concatenate((abs(X/N)[0:N//2][::-1], abs(X/N)[N
        //2:][::-1]), axis=None)
87      plt.plot(freq_axis, abs(reform_X))
88      plt.xlabel('Частотний␣розподіл')
89      plt.ylabel("Коефіцієнт␣Фурє'")
90
91
92      n_oneside = N//2
93      f_oneside = freq_axis[:n_oneside] + sr/2 - 0.5
94      X_oneside =X[:n_oneside]/n_oneside
95
96      plt.subplot(122)
97      plt.stem(f_oneside, abs(X_oneside), 'b', \
98               markerfmt="␣", basefmt="-b")
99      plt.xlabel('Частота')
100     plt.ylabel('Амплітуда')
101     plt.tight_layout()
102     plt.show()
103
104     phase_spectrum = [np.arctan(k.imag / k.real) for k in X]
105     print(phase_spectrum)
106
107     # result = np.zeros(N)
108     # for k in range(n_oneside):
109     #     result += abs(X_oneside[k]) * np.sin(2 * np.pi * freq[k] * t
        )
110     x_new = np.zeros(len(t))
111     for n in range(len(t)):
112         tmp_result = 0
113         for k in range(len(t)):
114             tmp_result += X[k] * np.exp(2j * np.pi * k * n / N)
115
116         x_new[n] = tmp_result/N
117
118     plt.subplot(121)
119     plt.plot(t, x, 'r')
120     plt.subplot(122)
121     plt.plot(t, x_new, "-b")
122     plt.xlabel("Time")
123     plt.ylabel("Ampl")
124     plt.show()
125
126
127 # Амплітудна і частотна модуляція
128 if 0:
129     n = 10244
130     t = np.linspace(0, 1, n, endpoint=True)
131     period = 2*np.pi*t
132     orig_freq = 3
133     modul_freq = 15
134     original_signal = np.cos(period*orig_freq)
135
136
137     amplitude_signal = 1 * (1+0.25*original_signal) * np.cos(period*
```

```
    modul_freq)
138
139     frequency_signal = 1 * np.cos(period*modul_freq + 12/orig_freq *
    np.sin(period*orig_freq))
140
141     # tt = np.linspace(0, 1, 32)
142
143     plt.plot(t, original_signal+3, "b-", linewidth=2)
144     plt.plot(t, amplitude_signal, "r-")
145     plt.plot(t, frequency_signal-3, "g-")
146     # plt.plot(t, original_signal)
147     # plt.xlim([0, n-1])
148     # plt.stem(tt, np.sin(2 * np.pi * tt), "o")
149     plt.show()
150
151
152
153 # Процес квантизації
154 if 0:
155     n = 100
156     t = np.linspace(0, 1, n, endpoint=True)
157     t2 = np.linspace(0, 1, n//2, endpoint=True)
158     period = 2 * np.pi * t
159     orig_freq = 3
160     modul_freq = 15
161     original_signal = np.cos(period * orig_freq)
162     quantized = np.cos(2 * np.pi * t2 * orig_freq)
163
164     plt.plot(t, original_signal, "b-")
165     plt.stem(t2, quantized, "k")
166     plt.bar(t2, quantized, align="edge", alpha=0.5, width=0.02, label=
    'Bar␣Plot', edgecolor="k", color="w")
167
168     plt.ylabel("Amplitude")
169     plt.xlabel("Time")
170     plt.show()
171
172
173 # АЧХ та зсув фази
174 if 0:
175     n = 1000
176     t = np.linspace(0, 2*np.pi, n, endpoint=True)
177     freq_resp = np.sqrt((0.5+0.5*np.cos(t))**2 + (0.5*np.sin(t))**2)
178     phase_resp = np.arctan((-0.5*np.sin(t)) / (0.5 + 0.5*np.cos(t)))
179
180     plt.subplot(121)
181     plt.plot(t, freq_resp, "b-")
182     plt.ylabel("Нормована␣АЧХ")
183     plt.xlabel("Частота␣радіани()")
184     plt.subplot(122)
185     plt.plot(t, phase_resp, "r-")
186     plt.ylabel("Зсув␣фази␣радіани()")
187     plt.xlabel("Частота␣радіани()")
188     plt.show()
189
190     from scipy import signal
```

```
191
192     b = [0.5, 0.5] #signal.firwin(80, 0.5, window=('kaiser', 8))
193     b = [0.25, 0.25, 0.25, 0.25]
194     w, h = signal.freqz(b) #, fs=400)
195
196     # fig = plt.figure()
197     plt.title('Частотна характеристика Фільтр( рухомого середнього)')
198     # ax1 = fig.add_subplot(111)
199
200     plt.plot(w, 20 * np.log10(abs(h)), 'b')
201     plt.ylabel('Амплітуда [dB]')
202     plt.xlabel('Відносна частота радіани[]')
203
204     # ax2 = ax1.twinx()
205     angles = np.unwrap(np.angle(h))
206     # plt.plot(w, angles, 'g')
207     # plt.ylabel('Angle (radians)', color='g')
208     plt.grid()
209     # plt.axis('tight')
210     plt.show()
211
212
213
214 # Віконні функції
215 if 0:
216     n = 50
217     freq = 1
218     t = np.linspace(0, 2 * np.pi, n, endpoint=True)
219     orig_x = np.cos(2*np.pi*freq*t)
220     orig_x2 = 0.1*np.cos(2*np.pi*2*t)
221     x = orig_x + orig_x2
222     plt.subplot(121)
223     plt.plot(t, x)
224     plt.ylabel("Амплітуда")
225     plt.xlabel("Час")
226     plt.title("Часовий простір")
227
228     f_x = np.fft.fft(x, 1028)
229     f_y = np.fft.fftfreq(len(f_x), 1/(n/(max(t)-min(t))))
230
231     # plt.plot(f_y, f_x.real, f_y, f_x.imag)
232     plt.subplot(122)
233     plt.title("Частотний простір")
234     plt.ylabel("Амплітуда")
235     plt.xlabel("Частота")
236     plt.plot(f_y, abs(f_x))
237     plt.show()
238
239     window = signal.windows.blackman(len(x))
240
241     x = x*window
242     plt.subplot(121)
243     plt.plot(t, x)
244     plt.ylabel("Амплітуда")
245     plt.xlabel("Час")
246     plt.title("Часовий простір")
```

```
247
248     f_x = np.fft.fft(x, 1028)
249     f_y = np.fft.fftfreq(len(f_x), 1 / (n / (max(t) - min(t))))
250
251     # plt.plot(f_y, f_x.real, f_y, f_x.imag)
252     plt.subplot(122)
253     plt.title("Частотний␣простір")
254     plt.ylabel("Амплітуда")
255     plt.xlabel("Частота")
256     plt.plot(f_y, abs(f_x))
257     plt.show()
258
259
260
261
262
263 # Фільтр Баттерворта
264 if 0:
265     b, a = signal.butter(3, 0.5, 'low')
266
267     w, h = signal.freqz(b, a)
268
269     plt.plot(w, 20 * np.log10(abs(h)))
270
271     plt.title('Частотна␣характеристика␣Фільтр(␣Баттерворта)')
272     plt.xlabel('Відносна␣частота␣радіани[]')
273     plt.ylabel('Амплітуда␣[dB]')
274     plt.grid(which='both', axis='both')
275     plt.show()
```

Додаток Б

Вихідний код програми, що використовувалась при написанні практичної частини частини

Лістинг Б.1: **Головний файл для парсингу FM сигналу** (main.py)

```
1 import numpy as np
2 from helpers import *
3
4 ##############
5 # Init Data #
6 ##############
7 samples = np.fromfile('SDRuno_20200907_184033Z_88110kHz.dat', dtype=np
      .complex64)
8
9 # Limit sample count for very-large files
10 samples = samples[:4000000]
11 sample_rate = 250e3
12 center_freq = 99.5e6
13
14
15
16 ####################
17 # Spectre Handling #
18 ####################
19 # Quadrature demod
20 # https://wiki.gnuradio.org/index.php/Quadrature_Demod#
      Mathematical_Description
21 x = 0.5 * np.angle(samples[0:-1] * np.conj(samples[1:]))
22
23 # Parse mono audio
24 sound_output(x.copy(), sample_rate)
25
```

```
# Get rds baseband
x, sample_rate = get_rds_from_fm(x, sample_rate)


####################
# BPSK Demodulate #
####################
x = sync(x)
bits = (np.real(x) > 0).astype(int)  # 1's and 0's
# Differential decoding
bits = (bits[1:] - bits[0:-1]) % 2
bits = bits.astype(np.uint8)


################
# RDS Decoder #
################
# http://www.interactive-radio-system.com/docs/EN50067_RDS_Standard.
    pdf
bytes_out = parse_rds_bitstream(bits)



################
# RDS Parser  #
################
parse_rds_radiotext(bytes_out)
```

Лістинг Б.2: **Допоміжні функції для парсингу FM сигналу** (helpers.py)

```
# Basis
import numpy as np
import scipy.signal as si
# Plots
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
# For the audio
from scipy.io import wavfile
import os


def to_logarithmic(elem):
    return 10 * np.log10(elem ** 2)


def plot_dft(x, srate):
    PSD = to_logarithmic(np.abs(np.fft.fftshift(np.fft.fft(x))))
    PSD = PSD[::100]
    PSD = PSD[len(PSD) // 2:]
    PSD = PSD - np.max(PSD)
    f = np.linspace(0, srate / 2, len(PSD)) / 1e3
    plt.plot(f, PSD)
    plt.axis([0, 125, -55, 1])
    plt.xlabel("Частота␣[kHz]")
    plt.ylabel("Амплітуда␣[dB]")
```

```
26      plt.show()
27
28
29  def plot_waterfall(samples, sample_rate, freq_center, fft_size=1024,
        chunks=500):
30      num_rows = len(samples) // fft_size #len(x) // fft_size # // is an
         integer division which rounds down
31      spectrogram = np.zeros((chunks, fft_size))
32      div = 1e3
33      xlimits = [(-sample_rate/2+freq_center)/div, (sample_rate/2+
        freq_center)/div]
34
35      for i in range(num_rows):
36          new_slice = samples[i*fft_size: (i+1)*fft_size] #signal.
        read_samples(i*fft_size, fft_size)
37          tmp_idx = i % chunks
38          spectrogram[tmp_idx,:] = to_logarithmic(np.abs(np.fft.fftshift
        (np.fft.fft(new_slice))))
39          if (i+1) % chunks == 0:
40              plt.imshow(spectrogram, aspect='auto', extent=[xlimits[0],
         xlimits[1], 0, len(samples)/sample_rate])
41              plt.xlabel("Частота␣[kHz]")
42              plt.ylabel("Час␣с[]")
43              plt.show()
44
45              spectrogram = np.zeros((chunks, fft_size))
46
47      plt.imshow(spectrogram, aspect='auto', extent=[xlimits[0], xlimits
        [1], 0, len(samples)/sample_rate])
48      plt.xlabel("Частота␣[kHz]")
49      plt.ylabel("Час␣с[]")
50      plt.show()
51
52
53  def sound_output(x, srate):
54      # plot_waterfall(x, sample_rate=srate, freq_center=0)
55      # Filter-out everything except mono audio
56      x = si.sosfilt(si.butter(2, 20e3, fs=srate, output="sos"), x)
57
58      # plot_waterfall(x, sample_rate=srate, freq_center=0)
59
60      # downsample by 6 to get mono audio
61      x = si.decimate(x, 6)
62      sample_rate_audio = srate//6
63
64      # normalize volume so its between -1 and +1
65      x /= np.max(np.abs(x))
66
67      # convert from float to int16s
68      x *= 32767
69      x = x.astype(np.int16)
70
71      wavfile.write('fm.wav', int(sample_rate_audio), x)
72      # os.system("aplay fm.wav")
73
74
```

```
75 def get_rds_from_fm(x, sample_rate):
76     # Freq shift to the center of the rds
77     N = len(x)
78     f_o = -57e3
79     t = np.arange(N) / sample_rate
80     x = x * np.exp(2j * np.pi * f_o * t)  # down shift
81     # plot_waterfall(x, sample_rate, 0)
82 
83     x = si.lfilter(si.firwin(numtaps=101, cutoff=7.5e3, fs=sample_rate
       ), [1], x)
84     # plot_waterfall(x, sample_rate, 0)
85 
86     # Resample to 19kHz
87     x = si.resample_poly(x, 19, 250)
88     new_sample_rate = 19e3
89     # plot_waterfall(x, sample_rate, 0)
90     return x, new_sample_rate
91 
92 
93 def plot_in_time(x):
94     subset = x[0:2000]
95     fig, ax = plt.subplots()
96     fig.set_tight_layout(True)
97     line, = ax.plot([0, 0, 0, 0, 0], [0, 0, 0, 0, 0], '.')
98     ax.axis([-0.02, 0.02, -0.02, 0.02])
99     subset = np.concatenate(
100         (np.zeros(100), subset))  # Add zeros at the beginning so that
        when gif loops it has a transition period
101 
102     def update(i):
103         i = int(i)
104         line.set_xdata([np.real(subset[i * 10:(i + 2) * 10])])
105         line.set_ydata([np.imag(subset[i * 10:(i + 2) * 10])])
106         return line, ax
107 
108     anim = FuncAnimation(fig, update, frames=np.arange(0, len(subset)
       / 10 - 2), interval=20)
109     anim.save('/tmp/constellation-animated.gif', dpi=80, writer='
       imagemagick')
110 
111 
112 
113 def sync(x):
114     # Mueller and Muller clock synchronization
115     # https://dsp.stackexchange.com/questions/75202/clock-recovery-
       using-mueller-and-muller-adds-noise-affecting-evm-or-snr-two-cas
116     samples = x  # for the sake of matching the sync chapter
117     samples_interpolated = si.resample_poly(samples, 32, 1)  # we'll
       use 32 as the interpolation factor, arbitrarily chosen
118     sps = 16
119     mu = 12.9  # initial estimate of phase of sample
120     out = np.zeros(len(samples) + 10, dtype=np.complex64)
121     out_rail = np.zeros(len(samples) + 10,
122                         dtype=np.complex64)  # stores values, each
       iteration we need the previous 2 values plus current value
123     i_in = 0  # input samples index
```

```
124     i_out = 2  # output index (let first two outputs be 0)
125     while i_out < len(samples) and i_in + 32 < len(samples):
126         out[i_out] = samples_interpolated[i_in * 32 + int(mu * 32)]  #
        grab what we think is the "best" sample
127         out_rail[i_out] = int(np.real(out[i_out]) > 0) + 1j * int(np.
        imag(out[i_out]) > 0)
128         x = (out_rail[i_out] - out_rail[i_out - 2]) * np.conj(out[
        i_out - 1])
129         y = (out[i_out] - out[i_out - 2]) * np.conj(out_rail[i_out -
        1])
130         mm_val = np.real(y - x)
131         mu += sps + 0.01 * mm_val
132         i_in += int(np.floor(mu))  # round down to nearest int since
        we are using it as an index
133         mu = mu - np.floor(mu)  # remove the integer part of mu
134         i_out += 1  # increment output index
135     x = out[2:i_out]  # remove the first two, and anything after i_out
        (that was never filled out)
136
137     # Animate constellation
138     # plot_in_time(x)
139
140     # Fine freq sync
141     samples = x
142     N = len(samples)
143     phase = 0
144     freq = 0
145     # These next two params is what to adjust, to make the feedback
        loop faster or slower (which impacts stability)
146     alpha = 100.0
147     beta = 0.5
148     out = np.zeros(N, dtype=np.complex64)
149     # freq_log = []
150     for i in range(N):
151         out[i] = samples[i] * np.exp(-1j * phase)  # adjust the input
        sample by the inverse of the estimated phase offset
152         error = np.real(out[i]) * np.imag(out[i])  # This is the error
        formula for 2nd order Costas Loop (e.g. for BPSK)
153
154         # Advance the loop (recalc phase and freq offset)
155         freq += (beta * error)
156         # freq_log.append(freq * sample_rate / (2*np.pi)) # convert
        from angular velocity to Hz for logging
157         phase += freq + (alpha * error)
158
159         # Optional: Adjust phase so its always between 0 and 2pi,
        recall that phase wraps around every 2pi
160         while phase >= 2 * np.pi:
161             phase -= 2 * np.pi
162         while phase < 0:
163             phase += 2 * np.pi
164
165     # Draw constellation
166     plt.plot(np.real(out[1000:1600]), np.imag(out[1000:1600]), '.')
167     plt.xlim([-0.030, +0.030])
168     plt.ylim([-0.030, +0.030])
```

```
169     plt.ylabel("Quadrature")
170     plt.xlabel("In-phase")
171     plt.show()
172     return out
173
174
175 # Constants
176 # Page 64
177 syndrome = [383, 14, 303, 663, 748]
178 offset_pos = [0, 1, 2, 3, 2]
179 # Page 59
180 #               A,    B,    C,    D,    C'
181 offset_word = [252, 408, 360, 436, 848]
182
183 # Annex F of RBDS Standard Table F.1 (North America) and Table F.2 (
        Europe)
184 #               Europe                    North America
185 pty_table = [["Undefined", "Undefined"],
186              ["News", "News"],
187              ["Current␣Affairs", "Information"],
188              ["Information", "Sports"],
189              ["Sport", "Talk"],
190              ["Education", "Rock"],
191              ["Drama", "Classic␣Rock"],
192              ["Culture", "Adult␣Hits"],
193              ["Science", "Soft␣Rock"],
194              ["Varied", "Top␣40"],
195              ["Pop␣Music", "Country"],
196              ["Rock␣Music", "Oldies"],
197              ["Easy␣Listening", "Soft"],
198              ["Light␣Classical", "Nostalgia"],
199              ["Serious␣Classical", "Jazz"],
200              ["Other␣Music", "Classical"],
201              ["Weather", "Rhythm␣&␣Blues"],
202              ["Finance", "Soft␣Rhythm␣&␣Blues"],
203              ["'Childrens␣Programmes", "Language"],
204              ["Social␣Affairs", "Religious␣Music"],
205              ["Religion", "Religious␣Talk"],
206              ["Phone-In", "Personality"],
207              ["Travel", "Public"],
208              ["Leisure", "College"],
209              ["Jazz␣Music", "Spanish␣Talk"],
210              ["Country␣Music", "Spanish␣Music"],
211              ["National␣Music", "Hip␣Hop"],
212              ["Oldies␣Music", "Unassigned"],
213              ["Folk␣Music", "Unassigned"],
214              ["Documentary", "Weather"],
215              ["Alarm␣Test", "Emergency␣Test"],
216              ["Alarm", "Emergency"]]
217 pty_locale = 1  # set to 0 for europe which will use first column
        instead
218
219 # page 72, Annex D, table D.2 in the standard
220 coverage_area_codes = ["Local",
221                        "International",
222                        "National",
```

```
223                         "Supra-regional",
224                         "Regional␣1",
225                         "Regional␣2",
226                         "Regional␣3",
227                         "Regional␣4",
228                         "Regional␣5",
229                         "Regional␣6",
230                         "Regional␣7",
231                         "Regional␣8",
232                         "Regional␣9",
233                         "Regional␣10",
234                         "Regional␣11",
235                         "Regional␣12"]
236
237
238 # Annex B, page 64
239 def calc_syndrome(x, mlen):
240     reg = 0
241     plen = 10
242     for ii in range(mlen, 0, -1):
243         reg = (reg << 1) | ((x >> (ii - 1)) & 0x01)
244         if (reg & (1 << plen)):
245             reg = reg ^ 0x5B9
246     for ii in range(plen, 0, -1):
247         reg = reg << 1
248         if (reg & (1 << plen)):
249             reg = reg ^ 0x5B9
250     return reg & ((1 << plen) - 1)  # select the bottom plen bits of
      reg
251
252
253 # 1187.5 bps / 104 bits = 11.4 groups/sec, or 45.7 blocks/sec */
254 def parse_rds_bitstream(bits):
255     # Initialize all the working vars we'll need during the loop
256     synced = False
257     presync = False
258
259     wrong_blocks_counter = 0
260     blocks_counter = 0
261     group_good_blocks_counter = 0
262
263     reg = np.uint32(0)
264     lastseen_offset_counter = 0
265     lastseen_offset = 0
266
267     # the synchronization process is described in Annex C, page 66 of
      the standard */
268     bytes_out = []
269     for i in range(len(bits)):
270         # in C++ reg doesn't get init so it will be random at first,
      for ours its 0s
271         # It was also an unsigned long but never seemed to get
      anywhere near the max value
272         # bits are either 0 or 1
273         reg = np.bitwise_or(np.left_shift(reg, 1),
274                             bits[i])  # reg contains the last 26 rds
```

```
    bits. these are both bitwise ops
275         if not synced:
276             reg_syndrome = calc_syndrome(reg, 26)
277             for j in range(5):
278                 if reg_syndrome == syndrome[j]:
279                     if not presync:
280                         lastseen_offset = j
281                         lastseen_offset_counter = i
282                         presync = True
283                     else:
284                         if offset_pos[lastseen_offset] >= offset_pos[j
    ]:
285                             block_distance = offset_pos[j] + 4 -
    offset_pos[lastseen_offset]
286                         else:
287                             block_distance = offset_pos[j] -
    offset_pos[lastseen_offset]
288                         if (block_distance * 26) != (i -
    lastseen_offset_counter):
289                             presync = False
290                         else:
291                             print('Sync␣State␣Detected')
292                             wrong_blocks_counter = 0
293                             blocks_counter = 0
294                             block_bit_counter = 0
295                             block_number = (j + 1) % 4
296                             group_assembly_started = False
297                             synced = True
298                 break  # syndrome found, no more cycles
299
300         else:  # SYNCED
301             # wait until 26 bits enter the buffer */
302             if block_bit_counter < 25:
303                 block_bit_counter += 1
304             else:
305                 good_block = False
306                 dataword = (reg >> 10) & 0xffff
307                 block_calculated_crc = calc_syndrome(dataword, 16)
308                 checkword = reg & 0x3ff
309                 if block_number == 2:  # manage special case of C or C
    ' offset word
310                     block_received_crc = checkword ^ offset_word[
    block_number]
311                     if (block_received_crc == block_calculated_crc):
312                         good_block = True
313                     else:
314                         block_received_crc = checkword ^ offset_word
    [4]
315                         if (block_received_crc == block_calculated_crc
    ):
316                             good_block = True
317                         else:
318                             wrong_blocks_counter += 1
319                             good_block = False
320                 else:
321                     block_received_crc = checkword ^ offset_word[
```

```
    block_number]  # bitwise xor
322                     if block_received_crc == block_calculated_crc:
323                         good_block = True
324                     else:
325                         wrong_blocks_counter += 1
326                         good_block = False
327
328                 # Done checking CRC
329                 if block_number == 0 and good_block:
330                     group_assembly_started = True
331                     group_good_blocks_counter = 1
332                     bts = bytearray(8)  # 8 bytes filled with 0s
333                 if group_assembly_started:
334                     if not good_block:
335                         group_assembly_started = False
336                     else:
337                         # raw data bytes, as received from RDS. 8 info
     bytes, followed by 4 RDS offset chars: ABCD/ABcD/EEEE (in US)
    which we leave out here
338                         # RDS information words
339                         # block_number is either 0,1,2,3 so this is
    how we fill out the 8 bytes
340                         bts[block_number * 2] = (dataword >> 8) & 255
341                         bts[block_number * 2 + 1] = dataword & 255
342                         group_good_blocks_counter += 1
343                         # print('group_good_blocks_counter:',
    group_good_blocks_counter)
344                     if group_good_blocks_counter == 5:
345                         # print(bts)
346                         bytes_out.append(bts)  # list of len-8 lists
    of bytes
347                 block_bit_counter = 0
348                 block_number = (block_number + 1) % 4
349                 blocks_counter += 1
350                 if blocks_counter == 50:
351                     if wrong_blocks_counter > 25:
352                         # This many wrong blocks must mean we lost
    sync
353                         print("Lost␣Sync␣(Got␣", wrong_blocks_counter,
     "␣bad␣blocks␣on␣", blocks_counter, "␣total)")
354                         synced = False
355                         presync = False
356                     else:
357                         pass
358                     blocks_counter = 0
359                     wrong_blocks_counter = 0
360     return bytes_out
361
362
363 def parse_rds_radiotext(bytes_inp):
364     radiotext_AB_flag = 0
365     radiotext = ['␣'] * 65
366     first_time = True
367     for bytes in bytes_inp:
368         block_0 = bytes[1] | (bytes[0] << 8)
369         block_1 = bytes[3] | (bytes[2] << 8)
```

```
370         block_2 = bytes[5] | (bytes[4] << 8)
371         block_3 = bytes[7] | (bytes[6] << 8)
372 
373         group_type = (block_1 >> 12) & 0xf
374         AB = (block_1 >> 11) & 0x1  # b if 1, a if 0
375 
376         program_identification = block_0  # "PI"
377 
378         program_type = (block_1 >> 5) & 0x1f  # "PTY"
379         pty = pty_table[program_type][pty_locale]
380 
381         pi_area_coverage = (program_identification >> 8) & 0xf
382         coverage_area = coverage_area_codes[pi_area_coverage]
383 
384         pi_program_reference_number = program_identification & 0xff  #
     just an int
385 
386         if first_time:
387             print("\nPTY:", pty)
388             print("program:", pi_program_reference_number)
389             print("coverage_area:", coverage_area)
390             print("\nReceived␣RadioText:")
391             first_time = False
392 
393         # Only support for the radiotext type
394         if group_type == 2:
395             # when the A/B flag is toggled, flush your current
    radiotext
396             if radiotext_AB_flag != ((block_1 >> 4) & 0x01):
397                 radiotext = ['␣'] * 65
398             radiotext_AB_flag = (block_1 >> 4) & 0x01
399             text_segment_address_code = block_1 & 0x0f
400             if AB:
401                 radiotext[text_segment_address_code * 2] = chr((
    block_3 >> 8) & 0xff)
402                 radiotext[text_segment_address_code * 2 + 1] = chr(
    block_3 & 0xff)
403             else:
404                 radiotext[text_segment_address_code * 4] = chr((
    block_2 >> 8) & 0xff)
405                 radiotext[text_segment_address_code * 4 + 1] = chr(
    block_2 & 0xff)
406                 radiotext[text_segment_address_code * 4 + 2] = chr((
    block_3 >> 8) & 0xff)
407                 radiotext[text_segment_address_code * 4 + 3] = chr(
    block_3 & 0xff)
408             res = ''.join(radiotext)
409             if res.count("␣") != 65:
410                 res = res.replace("\x0d", "\\r")
411                 print(res)
412         else:
413             pass
```