КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота

на здобуття ступеня магістра

за спеціальністю 122 Комп'ютерні науки

на тему:

ВІЗУАЛІЗАЦІЯ ТА РОЗМІТКА ДАНИХ ЕКГ ДЛЯ ВИДІЛЕННЯ ЕФЕКТИВНИХ ХАРАКТЕРИСТИЧНИХ ОЗНАК

Виконав студент 2-го курсу
Андрій ЛЯШКО _____

Науковий керівник:
професор, доктор фіз.-мат. наук

Юрій КРАК

Консультант:
доцент, кандидат технічних наук
Олексій ТКАЧЕНКО _____

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань

Студент _____

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування
«08» травня 2023 р., протокол № 16

Завідувач кафедри
Микола НІКІТЧЕНКО _____

Київ – 2023

РЕФЕРАТ

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел (14 найменувань) та 3 додатків. Робота містить 10 рисунків та 1 таблицю. Загальний обсяг становить 39 сторінок.

ЕКГ, ЕКГ ПО ХОЛТЕРУ, РОЗМІТКА ДАНИХ ЕКГ, НЕІНВАЗИВНИЙ АНАЛІЗ, СЕРЦЕВО-СУДИННІ ЗАХВОРЮВАННЯ, АНОМАЛІЇ СЕРЦЯ.

Об’єктом роботи є аналіз можливостей виявлення серцево-судинних захворювань за допомогою ЕКГ по Холтеру.

Предметом роботи є розробка системи, яка вирішує проблеми отримання даних ЕКГ, з їх подальшою обробкою та використанням.

Метою розробки є створення готової до розгортання системи, що надає можливості візуалізації та розмітки даних ЕКГ для виділення ефективних характеристичних ознак.

Інструменти розробки: середовище розробки PyCharm Community Edition, мова програмування Python, бібліотека PyEDFlib.

Результати розробки: проаналізовано та обрано оптимальні засоби для можливостей зчитування, розділення за ознаками та збереження даних ЕКГ по Холтеру.

Розроблена програма може бути корисна лікарям-кардіологам для виділення аномалій серцевого ритму для більш детального аналізу. Загалом програмне забезпечення може допомогти лікарю швидше поставити діагноз.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта, актуальність роботи та підстави для її виконання. Серцево-судинні захворювання є однією з найголовніших причин смертності в усьому світі [6]. Особливо гостро це проявляється останнім часом, коли люди ведуть менш рухливий спосіб життя, світом шириться епідемія та виникає багато інших збудників, які безпосередньо впливають на здоров’я серця. Саме тому виникає потреба в створенні програмних засобів, які здатні на початкових стадіях виявити хворобу для її профілактики або ефективного лікування.

Метою кваліфікаційної роботи є створення готової до розгортання системи, що надає можливості візуалізації та розмітки даних ЕКГ для виділення ефективних характеристичних ознак.

Об’єктом роботи є аналіз можливостей виявлення серцево-судинних захворювань за допомогою ЕКГ по Холтеру.

Предметом роботи є розробка системи, яка вирішує проблеми отримання даних ЕКГ, з їх подальшою обробкою та використанням.

Розмітка даних ЕКГ по Холтеру передбачає роботу з великими обсягами даних, тому потрібно мати ефективні алгоритми для вирішення поставлених задач. Перший розділ присвячений оцінці актуальності обраної теми, виявленні переваг та недоліків даного способу дослідження проблем з серцево-судинною системою та обґрунтування його ефективності. Наступний розділ передбачає обговорення етапів розробки програми для вирішення поставлених задач. В ньому буде прийняте рішення та його обґрунтування щодо вибору мови програмування. Останній, заключний розділ, де буде пояснено більшість концепцій, які вплинули на вибір одного чи іншого способу реалізації програмного продукту. Також буде розроблений кінцевий алгоритм та проілюстровано його роботу на

практиці. Наприкінці буде наведено короткий порівняльний аналіз створеної програми з вже наявними, а також буде пояснено в чому оригінальність даної системи, та в чому вона краща за порівнювану.

Можливі сфери застосування. Розроблена програма може бути корисна лікарям-кардіологам для виділення аномалій серцевого ритму для більш детального аналізу. Загалом програмне забезпечення може допомогти лікарю швидше поставити діагноз. Також кінцевий програмний продукт може бути корисний дослідникам, які будуть потребувати в аналізі розшифрованих даних для виявлення поведінки під час реєстрацій тих чи інших аномалій серцево-судинної системи.

РОЗДІЛ 1: ТЕОРЕТИЧНЕ ПІДҐРУНТЯ ДЛЯ РОЗРОБКИ АЛГОРИТМУ

В цьому розділі буде наведено основні теоретичні відомості, які необхідні для ілюстрації актуальності теми та іншим обґрунтуванням особливостей обраної теми роботи.

1.1 Способи та засоби для вимірювання серцевого ритму

На даний момент зберігається невтішна тенденція серцевих захворювань. За статистикою, наданою World Health Organization [7], у 2016 році від серцево-судинних захворювань померло 17,9 мільйонів людей, що складає 31% від загальної кількості померлих, з них 85% припадає на серцевий напад чи інсульт. Цьому сприяє малорухливий спосіб життя, постійний стрес, шкідливі звички, такі як алкоголь, куріння, вживання нездорової їжі.

Вченими було помічено, що саме під час епідемії COVID-19 статистика захворювання різко погіршилась [5], причому одразу для всіх вікових груп, зокрема особи віком 25-44 років, які раніше не були схильні до високого ризику серцевого нападу.

Якщо розглянути ситуацію в Україні [2], то за останні 30 років смертність від серцево-судинних захворювань зросла на 8%, що займає одне з перших місць в Європі за статистикою хвороб серця. За даними дослідження інституту когнітивного моделювання України (Рисунок 1), проведеними за січень-листопад 2022 року, найбільше смертей виникає через проблеми в системі кровообігу, зокрема ішемічною хворобою серця [8].

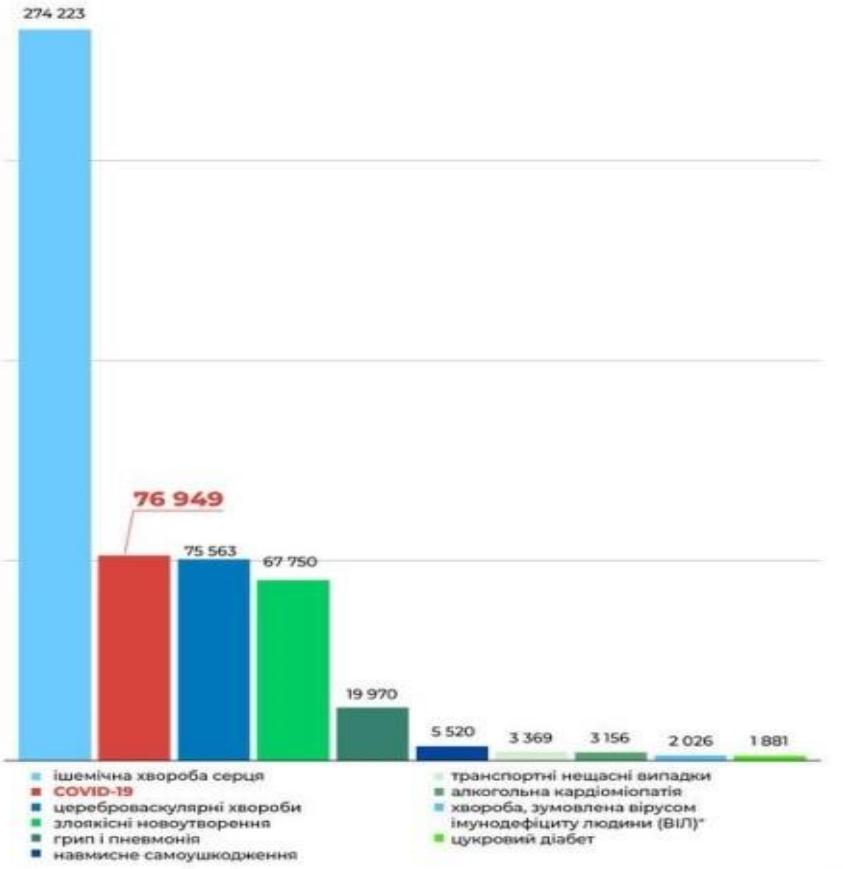

Рисунок.1. Кількість летальних випадків за причинами смерті в Україні.

Саме тому виникає потреба в створенні ефективних засобів для дослідження даної тематики [6]. Одним з перевірених методів є електрокардіографія. Зараз це один з найпоширеніших методів дослідження роботи серця. Він заснований на вимірюванні електричної активності серця. Прилад фіксує зміни електричних імпульсів та надає відображення серцевого ритму в вигляді графіку залежності сили електричного потенціалу від часу. Датчики виконані в вигляді присосок, які швидко встановлюються на потрібні зони. Графік можна отримати або на папері, чи переглянути на пристрої. Зазвичай вимірювання відбуваються на протязі

декількох десятків секунд. Перевагою є швидкість отримання даних, яка може грати вирішальну роль в критичних ситуаціях. Недоліком є те, що в багатьох випадках прилад не може дати повну картину хвороби, наприклад при аритмії, яка може проявитись в вузькому часовому діапазоні [4]. Крім того дослідження найчастіше відбувається в стані спокою, а не під час звичного навантаження.

Інший метод дослідження, на якому зосереджена ця робота – моніторинг за Холтером [3]. Основною відмінністю є більша тривалість спостережень. В пристрої Холтера найчастіше це 12, 24, 48 чи 72 години. Пристрій характеризується своєю зручністю та компактністю, не створює особливого дискомфорту при використанні.

Під час проведення процедури вимірювання даних Холтера потрібно заповнювати спеціальний щоденник в який буде записуватись інформація про фізичні навантаження, тривалість сну, прийом лікарських препаратів, можливий стрес та інші чинники, які можуть вплинути на показання серця. Крім звичайного способу проведення дня також потрібно проаналізувати роботу серця при посилених навантаженнях, наприклад біг, ходьба по сходах. Це потрібно для того, щоб лікар-кардіолог міг побачити при яких обставинах проявляється хвороба. Цей спосіб є дуже ефективним при аритмії чи ішемічних хворобах серця. Запис даних ведеться одразу за декількома каналами, найчастіше це 2-3, проте бувають випадки коли використовують до 12 каналів, найчастіше при першому проведенні процедури. Для точності показань потрібно максимально уникати впливу вібрацій, тримати комфортну температуру навколишнього середовища для тіла, не наближатись близько до джерел електромагнітного

випромінювання та хвиль. Також Холтер використовують для спостереження за пацієнтом після проведеного лікування в області серця.

Безсумнівною перевагою використання цих методів є те, що для їх проведення не потрібно спеціальної підготовки, також вони є інвазивними, тобто при використанні не потрібно хірургічне втручання. Для експрес аналізу слід використовувати електрокардіограф, який може надати загальні дані хвороби. Якщо лікар-кардіолог приймає рішення нестачу інформації, потрібно використовувати прилад Холтера, який дозволить побачити більш конкретну ситуацію та виявити ряд хвороб, які найчастіше проявляються в певний, вузький проміжок часу.

1.2 Особливості збереження даних, отриманих за допомогою Холтерівського моніторингу

Варто зазначити, що прилад Холтера абсолютно автономний, й під час вимірювань не потребує втручання. Вся інформація зберігається на вбудовану пам’ять, з якої після завершення можна отримати вихідні дані пацієнта, після чого їх можна конвертувати в зручний формат. Оптимальним форматом для подальшої роботи є .edf формат(European Data Format). Він є стандартизованим, що дозволяє зберігати багатоканальні біологічні та фізичні сигнали. Детальніше про формат можна переглянути на офіційному сайті [9]. Весь файл є зашифрованим стандартними алгоритмами шифрування, такі як 8 ascii, 16 ascii та інші.

Перерахуємо основну інформацію, яка знадобиться для створення графіків серцевого ритму. Найважливішими даними є числові характеристики сигналів та їх залежність від часу фіксування. Також потрібними показниками є кількість задіяних сигналів, кількість записування даних на одиницю часу, загальна протяжність вимірювань, особиста інформація пацієнта, інформація про дату та час проведення спостережень. Щоб не виникало помилок, рекомендується, щоб тривалість кожного запису даних була цілою кількістю секунд, а його розмір не перевищував 61440 байтів.

Однією з проблем, яка виникне при подальшій розробці програми – потреба в розшифруванні, зберіганні та аналізі великої кількості даних. Зазвичай пристрій Холтера записує близько 250-500 скорочень серця за секунду. В кожному файлі щонайменше 2-3 сигнали, а при першому проведенні процедури може бути досягати 12. Якщо порахувати загальну кількість даних сигналів на одного пацієнта, то вийде що програма повинна

обробити 3 сигнали * 250 точок в секунду * кількість секунд в добі, це близько $648 * 10^5$ точок. На практиці файл цього формату що містить стільки даних має розміри близько 100-150 мегабайт. Це означає що запуск та аналіз з подальшим створенням попереднього висновку потребує достатньо програмних потужностей та ефективних алгоритмів. В ході розробки алгоритму потрібно максимально зменшувати споживання ресурсів та часу, що необхідні для виконання аналізу.

Через великий інтервал вимірювання та велику кількість даних виникає ще одна проблема, коли потрібно буде відобразити графік серцевого ритму. Якщо прийняти, що в середньому серце робить 60 ударів в хвилину, то для того щоб їх було хоч якось видно, потрібно загальний інтервал поділити на рівні проміжки меншої довжини для їх адекватної візуалізації. Саме тому виникає програма, яка буде реалізувати ці можливості.

РОЗДІЛ 2: ВИБІР МОВИ ПРОГРАМУВАННЯ, ВИЯВЛЕННЯ НЕОБХІДНИХ ЗАВДАНЬ

В цьому розділі будуть описані основні ідеї та концепції для створення повноцінної програми, яка буде відповідати поставленим завданням . Спершу буде наведено порівняння найпопулярніших мов програмування, розказано про їх переваги та недоліки в різних аспектах та буде обрана оптимальна мова для реалізації алгоритму. В наступних частинах буде проведений порівняльний аналіз підходів, та буде обрано найоптимальніший.

2.1 Вибір мови програмування, обґрунтування вибору

Вибір мови програмування для розробки залежить від багатьох факторів. Насамперед мова має бути ефективна, тобто її використання дозволяє зменшити часові витрати на запуск та виконання, а також має використовуватись якнайменше пам’яті. Ще одна не менш важлива причина – багатофункціональність. Проект передбачає не лише обробку великої кількості даних, а й створення зручного графічного інтерфейсу з подальшою взаємодією з користувачем. Третій критерій вибору – простота мови програмування. Вона має бути легка для освоєння та при менших затратах на створення коду породжувати еквівалентний код.

Розглянемо декілька варіантів вибору та оберемо найоптимальніший з них. Обирати будемо з сучасних мов програмування, адже їх розробники піклуються про підтримку актуальних версій, що дозволяє користуватись їх перевагами та не боятись що з часом розроблену програму можна втратити та потрібно буде переносити на іншу платформу. В трійку

найпопулярніших мов входять Python, Java та C++ [12]. Зробимо коротке порівняння переваг та недоліків їх використання.

З точки зору швидкості Java та C++ є дещо швидшими в порівнянні з Python, адже ця мова є динамічною та інтерпретованою. Що стосується використання пам'яті, то найменш витратним в цьому питанні є C++, трохи кращим є Python. Також великою перевагою Python є створення коду. В порівнянні з іншими мовами його можна швидко опанувати навіть людині, яка нещодавно почала вивчати програмування. Також еквівалентний код є більш стислим та зрозумілим.

Найважливішим для розробки програми в цій роботі є можливості для дослідження. В цьому питанні Python є найкращим з трійки. В ньому доступна широка гамма відкритих бібліотек, зокрема великі бібліотеки для статистичних аналізів, добре розвинуті графічні інтерфейси, які дозволяють відображати дані в різних виглядах та з широким функціоналом.

Саме тому для розробки алгоритму була обрана мова Python [10], так в ній є все необхідне для обробки великих масивів даних, створення графічного представлення електрокардіограм.

2.2 Розшифрування файлу Холтера, збереження сигналів

Для виділення даних, отриманих за допомогою Холтерівського моніторингу, спочатку потрібно розшифрувати його вихідний файл. В ньому знаходиться основна інформація, така як кількість сигналів, загальна кількість зафіксованих скорочень серця та їх числова характеристика.

Існує декілька підходів в виокремленні даних. Першим, більш довготривалим та витратним по часу розробки є пряме розшифрування. Так як кожна стрічка вихідного файлу зашифрована в різному форматі, то потрібно точно знати яким способом сховані потрібні нам дані. Також треба відкривати вихідний файл декілька разів в різних форматах, переходити до середини файлу багато разів а також десь зберігати формати шифрування. Розробка з початку такої програми призведе до великих витрат на розробку та реалізації такого алгоритму.

Другий спосіб – використання готових бібліотек. Він є кращим з декількох причин. По-перше, не потрібно писати велику частину коду з самого початку, в більшості випадків достатньо обмежитись декількома командами для отримання бажаного результату. Є велика кількість популярних бібліотек, які постійно покращують швидкодію, використання пам'яті та зберігають актуальність. Тому для виконання цієї частини роботи було прийнято рішення користуватись готовими бібліотеками.

Однією з популярних, яка реалізує потрібні функції є PyEDFlib [11]. В неї є офіційний сайт, який містить коротку інструкцію з встановлення та використання. Налаштування потрібно зробити один раз, та вони займають декілька хвилин. Для використання потрібно встановити актуальну версію Python(3.5 або вище) та Numpy(1.9.1 або вище). Найчастіше всього при

встановленні середовища для запуску програм мовою Python вони встановлюються автоматично. Тож фактично встановлення зводиться до імпорту бібліотеки в програму.

Для подальшої розробки алгоритму я буду використовувати файли, які були взяті з вимірювання серцевого ритму реальних пацієнтів, що гарантовано містять підозрілі значення на проміжках з фібриляцією серця. Для прикладу візьмемо один файл, що містить залежність даних серцевого ритму від часу на трьох сигналах, зі щільністю точок 250 за секунду та інтервалом вимірювання в 24 години.

Якщо використовувати можливості PyEDFlib, то перша спроба від запуску до отримання першого значення займає близько 20-25 секунд. В випадку, якщо для кожного пацієнта програма буде запускатись лише один раз, то в будь-якому разі потрібно чекати стільки часу, проте при повторному запуску можна суттєво зменшити час на повторне виконання, зчитавши не всі дані, я їх частину, проте доцільність такої доробки буде розглянуто в наступному розділі при розробці та реалізації алгоритму. Також є потреба в розділенні даних за їх сигналами, з їх подальшою обробкою та використанням. Для того, щоб при кожному запуску можна було швидко отримати дані певного сигналу, потрібно створити нові файли, в які будуть записані його координати. Це повинно виконуватись для всіх нових пацієнтів, і для економії часу та зручності при подальшому виклику цього пацієнта можна використовувати вже розшифровані, записані в файли дані. Зрозуміло, що кожен файл повинен мати унікальне ім’я, щоб значення не перезаписувались, не зіпсувались та збереглись. Тож в цій частині роботи потрібно створити фрагмент коду, який буде розшифровувати файл, розбивати його за сигналами, записувати їх в файли з можливістю повторного використання.

2.3 Створення графічного відображення координат серцевого ритму

Наступною частиною роботи, яку потрібно реалізувати – створення зручного графічного інтерфейсу з можливістю візуалізації та перегляду координат серцевого ритму пацієнта за певний проміжок часу. При розробці концепцій попереднього розділу можна отримати будь-яку координату певного сигналу, загальну кількість точок, час вимірювання даних пацієнта. Для того, щоб створити графік, потрібно мати щонайменше дві координати, а саме час та сигнал.

Так як дані з файлів можливо отримувати лише по одному, це означає що для того щоб отримати сигнал на n-ному місці, потрібно зчитати попередні 0, 1, …, n-1 координати. З цього випливає, що при зчитуванні з файлу програма ніяк не зможе змінити порядок даних. Це означає що для того, щоб отримати координату часу достатньо звернутись до порядкового номеру певного елемента. Знаючи загальну кількість точок та загальний період вимірювання можна знайти кількість точок, що записуються Холтером за одну секунду. Для знаходження часу, який відповідає координаті я буду користуватись формулою:

$$t_{n,s} = (pos(n,s)) * \frac{len(x_s)}{t_s}$$ (Формула 1),

де $t_{n,s}$ – це час фіксації n-ї координати сигналу s, $pos(n,s)$ – поточна позиція n-ї координати сигналу s в масиві, $len(x_s)$ – загальна кількість координат сигналу s, t_s – загальна тривалість вимірювання. В мові Python початок відліку масиву чи інших контейнерів починається з 0, саме тому формулі до $pos(n,s)$ не додається 1 для того, щоб перша точка інтервалу починалась в нульовий момент часу. Тому коли користувач захоче отримати координати за період від t_1 до t_n, то результатом будуть

координати від t_1 включно до t_n не включно. Також це дозволить уникнути дублювання крайніх координат графіку при переході до нового інтервалу.

Тепер, коли є координати з двома змінними, можна створити графік. Проте якщо спробувати відобразити всі дані які зареєстрував Холтер, графік буде настільки щільним, що не буде зрозуміло який ритм у пацієнта. Також якщо генерувати графік від всього інтервалу, потрібно більше часу й пам’яті для запуску. Щоб цього уникнути потрібно вхідні дані розбити на рівні за розміром відрізки і працювати з ними.

Методом підстановки було виявлено, що оптимальним рішенням є розбиття загального інтервалу на менші, тривалістю 1 хвилина. По-перше, графік буде чітко видно навіть без наближення. Ще одна вагома причина – менше використання ресурсів комп’ютеру чи пристрою де програма буде запущена. За одну хвилину при створенні одного графіку в середньому потрібно розмістити 250 точок на секунду * 60 секунд в хвилині. Також дослідним шляхом було з’ясовано, що для чіткої візуалізації графіку серцевого ритму достатньо показати 50 точок на секунду. Це дозволить зменшити час переходу до іншого інтервалу та зробити його майже непомітним при використанні. Також варто зазначити, що при розробці потрібно писати код так, щоб в будь-який момент його можна було доповнити та змінити, щоб інший функціонал не постраждав. Тож при реалізації алгоритму основні змінні краще розмістити в загальному класі, щоб була можливість керувати основними змінними з одного місця, щоб вона так само працювала вже з новими значеннями. Наприклад користувач захоче змінити тривалість інтервалів графіку, це можна буде зробити змінивши одну змінну на всю програму. Отже метою розробки цього розділу є створення графічного інтерфейсу, який буде аналізувати попередньо заповнені масиви даних та виводити на екран потрібний їх сегмент.

2.4 Збереження даних за обраний період часу

Припустимо, що при перегляді графіків користувач знаходить потенційно підозрілий інтервал, наприклад на ньому чітко видно підозрілі значення або інші, властивості, яких потрібно виділити. Саме тому потрібно додати функціонал, який буде зберігати поточні дані, наприклад у файли. Після збереження не потрібно аналізувати загальний інтервал, а достатньо передивитись лише виділені значення. Після проходження загального інтервалу вимірювання буде отримана система файлів, яка буде містити у собі дані де були зареєстровані аномалії, тобто час початку, час завершення, інші важливі змінні та самі дані.

На початковому етапі реалізації програми користувач повинен сам знаходити підозрілі проміжки. В подальшому цей функціонал можна розширити, створивши методи, які будуть автоматично виділяти дані, й поступово можна додавати різні функції, які будуть краще знаходити особливості.

Також варто зазначити, що жодна програма не може брати на себе відповідальність за встановлення кінцевого діагнозу, адже є хоч і невисока, проте ймовірність помилки. Саме тому програма може надати функціонал для розшифрування, зручного відображення графіків з можливістю збереження даних, а лікар спираючись на цю інформацію має проаналізувати надані результати та прийняти остаточне рішення.

На даному етапі реалізації цього розділу потрібно доповнити програму так, щоб вона могла зберігати дані за певний проміжок часу, а саме відповідні координати всіх сигналів та зберігати відповідні графіки, які будуть відповідати цим координатам.

РОЗДІЛ 3: РОЗРОБКА АЛГОРИТМУ ВІЗУАЛІЗАЦІЇ ТА РОЗМІТКИ ДАНИХ ЕКГ. СТВОРЕННЯ ПРОГРАМИ

Цей розділ складається з розробки алгоритмів, необхідних для реалізації завдань, що вказані вище. В ньому будуть коротко описані основні концепції, вибір оптимальних з точки зору простоти та ефективності. Наприкінці наведено короткий огляд програми та її можливостей.

3.1 Розшифрування файлу Холтера та збереження сигналів. Реалізація

Для розшифрування файлів Холтера я буду користуватись бібліотекою PyEDFlib [4]. Для того, щоб можна було швидко коригувати значення змінних був створений клас Variables.py (Додаток А). В ньому будуть знаходитись основні змінні, які можна швидко модифікувати й змінювати спосіб збереження чи візуалізації даних. Для роботи з файлами в класі виділені константи, які містять у собі посилання на розміщення файлу Холтера, який потрібно розшифрувати, кількість точок за секунду які будуть використовуватись, розширення файлів, які будуть генеруватись. За замовчуванням відповідні змінні ініціалізовані наступними значеннями:

file_directory = "D:/holter_files/2@1951-01-31.edf" – посилання на локальне місце зберігання файлу Холтера

get_points_in_one_sec = 10 – яку кількість точок зберігати. Тобто щоб дізнатись кількість записуваних точок за секунду, потрібно початкову кількість точок за секунду розділити на це значення .

text_type = ".txt" – текстове розширення генерованих файлів.

Також створимо файл, що реалізовуватиме логіку зчитування та збереження відповідних сигналів при першому запуску файлу цього пацієнта. Назвемо його fileManager.py (додаток 1). Коротко опишу логіку роботи алгоритму з цього класу. На вхід конструктору потрібно передати тільки назву файлу. В подальшому для генерації нових файлів, які розділять вхідний зашифрований файл за сигналами, буде використовуватись його назва, тобто 2@1951-01-31.edf (зображення вмісту на Рисунку 7).

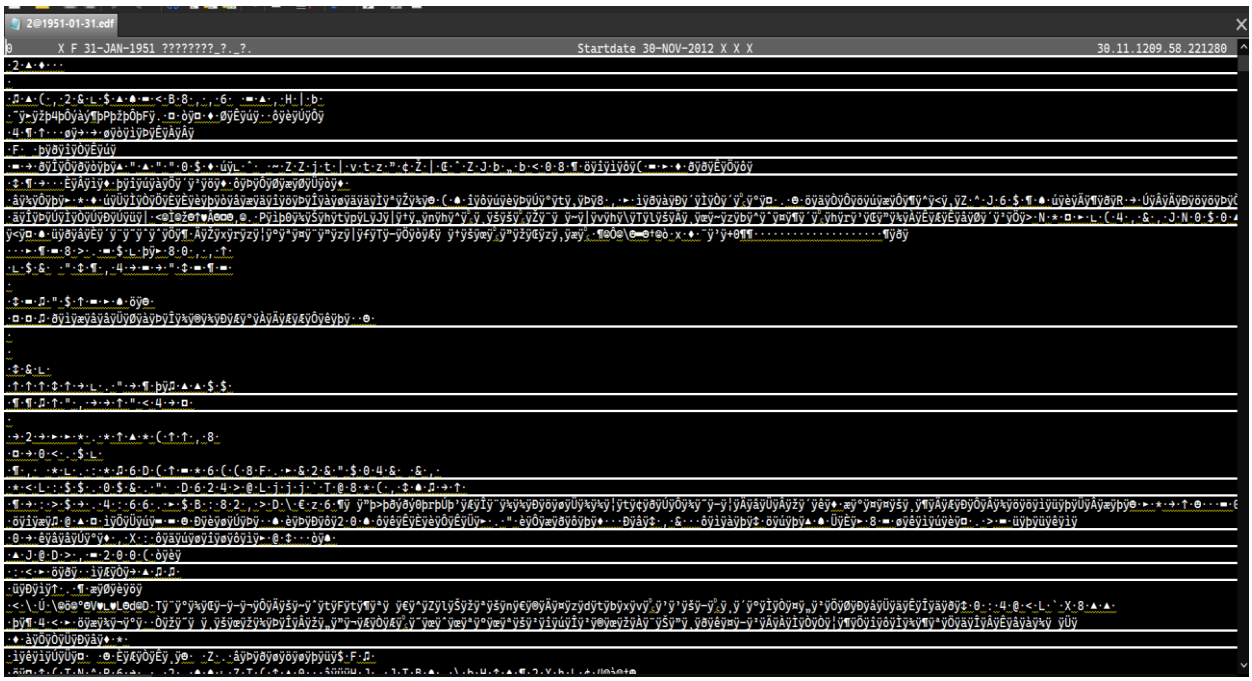

Рисунок 7. Вміст вихідного файлу Холтера перед розшифруванням

Також до неї буде додаватись стрічка "_generated_" щоб користувач бачив що вони були згенеровані. Далі для формування назви файлу додаємо номер сигналу який буде записуватись та в кінці додаємо розширення файлу, яке потрібно генерувати. Така генерація відбувається для кожного сигналу з вхідного файлу Приклад назви файлу з назвами змінних для сигналу 0 буде створений наступним чином:

2@1951-01-31.edf_generated_0.txt

В файл записуються всі дані поточного сигналу (ілюстрація після розшифрування на Рисунку 8).

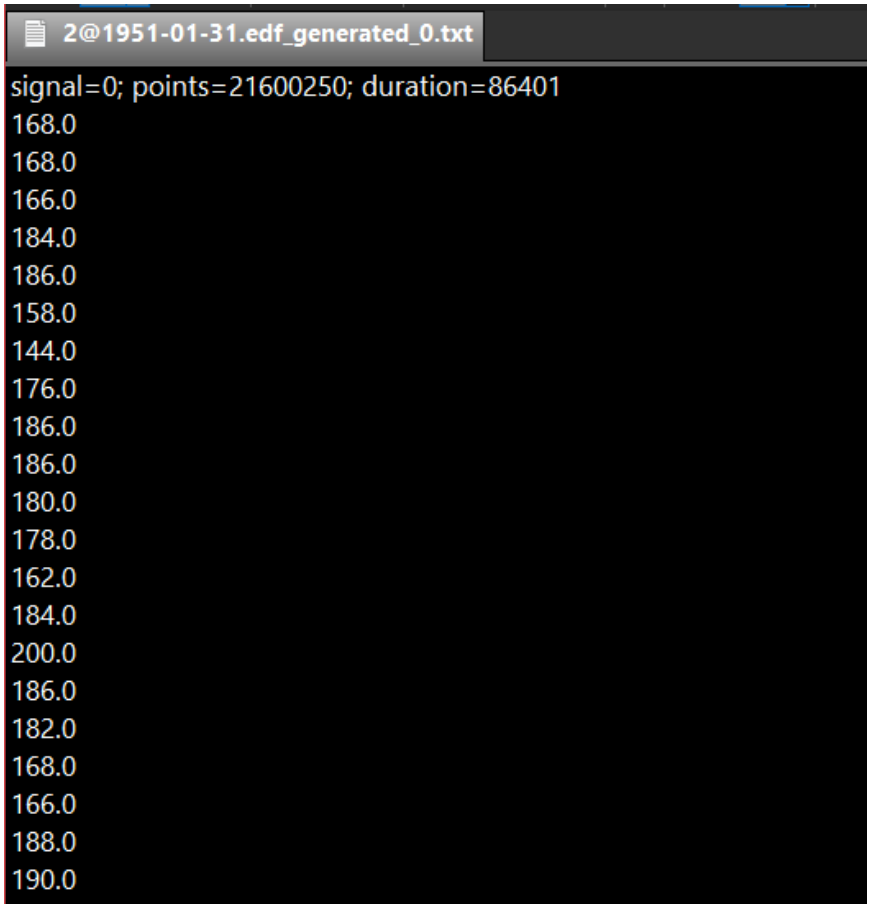

Рисунок. 8. Вміст вихідного файлу Холтера після розшифрування

Також в цьому класі є службовий метод, який при першому запуску генерує та наповнює ці файли, а при кожному наступному перевіряє чи всі файли коректно створені, якщо так то відповідає позитивно, якщо ні, то потрібно заново створити ці файли та їх заповнити. Значення що повертається потрібне щоб показати, що алгоритм успішно відпрацював і не виявив помилок.

Після цього потрібно навчити програму витягувати звідти дані. Для цього було створено окремий клас container_Manager.py (Додаток В). В ньому реалізовано можливість виділення інформації, яка записувалась в файли за допомогою можливостей попереднього класу. Цей клас дозволяє витягнути з файлів інформацію про тривалість вимірювань, кількість даних за секунду та самі дані. При створенні класу кожен раз перевіряється чи всі файли успішно знайдені, якщо ні, то відбувається повторне створення й перезапис, якщо так то заповнюються контейнери для подальшої роботи з даними. Також для того, щоб не наповнювати масиви всіма даними, використовується константа get_points_in_one_sec, яка за замовчуванням рівна 10. Вона регулює яку кількість значень потрібно залишити, тобто якщо в файлі щільність 250 значень в хвилину, то буде обиратись кожне 10 значення та записуватись в масив даних. Значення за замовчуванням було отримано протягом запуску програми, й було визначене як найоптимальніше з точки зору наповнення даними та швидкості. З використанням цієї змінної графік буде добре розрізнятись та не буде так багато витрачатись пам’яті та часу на запис в масиви та розміщення цих даних на графіку. Варто зазначити, що це значення потрібно обирати кратне кількості значень в секунду, адже в протилежному випадку можуть зустрічатись інтервали з різною кількістю значень, і в подальшому обрахунок часу фіксації даних з певного сигналу може збитись.

Також при розробці потрібно було визначитись як будуть записуватись дані. Перший спосіб – при створенні класу записувати всі значення. Цей метод дозволяє ініціалізувати всі потрібні дані на початку й потім користуватись будь-якими з них, але якщо користувачу не потрібно буде переходити до сильно віддалених графіків, то він програє наступному.

Інший метод – поступова ініціалізація. Цей спосіб хороший тим, що при виклику перших інтервалів графіка він буде працювати в рази швидше та буде використовуватись менше пам’яті. Проте він буде не ефективний при виклику сильно віддалених від початку інтервалів, адже специфіка використання файлів вимагає, що для отримання координати на місці n обов’язково потрібно прочитати всі попередні 1, 2, …, n-1. Отже коли ми запустимо програму та спробуємо створити графік, користувач швидко отримає графік першого інтервалу, проте при різкому переході до останнього програма може зависнути через велику кількість даних які потрібно одночасно записати. В порівнянні з попереднім способом, цей є більш нестабільним. Також якщо буде реалізуватись автоматична перевірка інтервалів на критичні значення(тобто інтервали з явними даними хвороби), то все одно потрібно буде пройти весь загальний інтервал. Тому на даному етапі було вирішено проводити повну ініціалізацію до того, як буде створено графік першого інтервалу. Якщо в майбутньому потрібно буде виконати поступову ініціалізацію, то це можна буде легко зробити, доповнивши вже наявний функціонал.

3.2 Розробка алгоритму візуалізації даних ЕКГ з подальшим збереженням даних

Наступний етап в розробці програми - створення графічного відображення отриманих даних. Для цього створимо новий клас graph_Manager.py (Додаток С), в якому буде відображатись розміщення переданих в нього даних. На вхід можна передавати масиви, які були попередньо ініціалізовані, але якщо користувач при запуску готової програми хоче побачити тільки графік, то йому не обов’язково знати які дані при цьому передаються, тож кращим рішенням буде всередині класу graph_Manager.py викликати та створювати всі потрібні методи та змінні для його роботи. Для цього будуть використовуватись попередні реалізовані класи container_Manager.py та fileManager.py.

На даному етапі доступна інформація про ініціалізовані масиви, кожен з яких містить дані кожного сигналу. Кількість значень кожного сигналу обов’язково співпадає, отже за формулою 1 з розділу 2.3 можна кожному значенню поставити у відповідність числову характеристику часу, в який відбувався удар серця. Так як вимірювання ритму відбувається протягом декількох діб, то потрібно перейти до однієї системи числення часу для всіх даних. Тож числовою характеристикою часу удару буде значення реєстрації серцевого скорочення в секундах, в точності до 8 знаків після коми. Для того, щоб в подальшому можна було переходити до інших інтервалів введемо наступні змінні:

delta_time = 60 – довжина інтервалів в секундах, на які буде розділений загальний інтервал

startTime = 0 – час, що відповідає початку відліку поточного інтервалу.

finishTime = startTime + delta_time – час кінця поточного інтервалу. Він рахується в залежності від початку інтервалу та довжини одного інтервалу.

Кожен раз при переході до іншого інтервалу обраховується очікуване значення startTime, і відносно нього рахується інші змінні. Тепер кожен раз при переході до іншої частини рахується початок та кінець інтервалу, після цього з вхідних масивів відсіюються потрібні значення та додаються на графік. Для створення графічного відображення даних я буду користуватись стандартними бібліотеками Python [9], а саме numpy, matplotlib.pyplot та matplotlib.widgets. Щоб під час роботи програми можна було переходити до будь-якого інтервалу чи обирати різні графіки або сигнали, я користувався можливостями бібліотеки matplotlib.widgets(Button, Slider, CheckButtons). Використавши їх було створено зручний графічний інтерфейс, який може видавати криву серцебиття певного сигналу(в меню є можливість обирати будь-який з наявних сигналів) за визначений проміжок часу з можливістю масштабування, прокрутки, переходу до наступного та попереднього інтервалу, переходу до заданого інтервалу.

На графіку можна бачити координатну площину, в якій вісь x відповідає за час реєстрації даних, а вісь y за числове представлення скорочення серця. Так як в одному інтервалі розміщуються значення за одну хвилину, то ілюстрація масштабу графіка відображається в секундах. На віддалених від початку інтервалах шкала значень часу повинна співпадати. Для цього в меню додана шкала, яка відповідає за початок відліку даного інтервалу, а для того щоб отримати на якій секунді було зареєстроване значення, потрібно до значення шкали додати значення координати x. Для підрахунку часу початку відліку певного інтервалу була створена наступна функція:

```
def convertSecondsToTime(s):
  hours = s // 3600
  minutes = (s - hours * 3600) // 60
  seconds = s - hours * 3600 - minutes * 60
  milis = (s - hours * 3600 - minutes * 60 - seconds) * 100
  return "Current time: " + str(hours) + "h. " + str(minutes) + "m. " + str(seconds) + "s. " + str(milis) + "ms"
```

В ній використовуються лише базові арифметичні операції та операція конкатенації строк. Вона викликається при кожній зміні графіку(при переході до іншого інтервалу), тому вона не несе великого навантаження на швидкість виконання програми.

Також потрібно уточнити логіку вибору масштабу за координатою y. Так як дані моніторингу за Холтером збираються під час руху, деякі дані можуть сильно відрізнятись від інших, іноді більше ніж в 10 разів. В Python масштабом графіка обирається максимальне значення для верхнього порогу, та мінімальне значення для нижнього. Це може призвести до вибору масштабу, при якому графік буде погано видно. Саме тому для визначення оптимального значення верхнього та нижнього порогу створена наступна функція:

```
def predictionLimits(data, amount_of_period=1):
  res = []
  mean_val = np.mean(data)
  st_dev = np.std(data)
```

```
if amount_of_period < 1:
    amount_of_period = 1
res.append(mean_val - amount_of_period * st_dev)
res.append((mean_val + amount_of_period * st_dev))
return res
```

За основу взято припущення, що при нормальному розподілі деяка кількість даних гарантовано знаходиться в інтервалі [середнє –n*стандартне відхилення, середнє +n*стандартне відхилення], де n – це деяке невід'ємне число, при збільшенні якого в інтервал потрапляє більша кількість значень. Ці два значення обчислюються лише один раз при створенні першого графіку, після чого зберігаються на весь діапазон вхідних значень. Для прикладу, при обрахунку порогових значень масштабу для n = 3 на графіку будуть відображатись 99.7% значень, що гарантує що всі важливі дані будуть збережені, і навіть деякі викиди не будуть відображатись. При потребі значення n можна збільшити, тоді більше даних потрапить на графік.

До функціоналу програми також варто додати відображення декількох графіків, створених за різними формулами. Дані різних пацієнтів можуть відображатись в різному масштабі, або при створенні графіків виникає проблема, що підозрілі значення важко виявити, або потрібно якось порівняти різні графіки. Саме для цього вводять функції стандартизації, щоб більш явно виділити потрібні характеристики значень з можливістю їх порівняння. Зробити це не важко, потрібно лише задати функції, які будуть обробляти вхідні дані та отримувати нові, які будуть змінюватись за заданими в них законах. В програмі додано дві додаткові функції, які

стандартизують дані. Перша з них перетворює вхідні за допомогою z-Score. Для кожного переданого значення обраховуються нове за таким алгоритмом:

```
def normalize_zscore(input_data):
    out_data = []
    disp_sqrt = np.sqrt(np.var(input_data))
    mean_v = np.mean(input_data)
    for i in input_data:
        out_data.append((i - mean_v) / disp_sqrt)
    return out_data
```

Для обрахунків використовуються можливості стандартної бібліотеки numpy, щоб обчислити значення дисперсії, її квадратного кореня та середнього значення.

Інша функція отримується за допомогою MinMax алгоритму. Реалізація мовою Python:

```
def normalize_mean(input_data):
    out_data = []
    min_v = np.min(input_data)
    down = np.max(input_data) - min_v
    for i in input_data:
        out_data.append((i - min_v) / down)
```

return out_data

В основі також взяті можливості бібліотеки numpy. Обидві функції не потребують збереження після виконання програми, адже кожен раз при запуску зберігаються звичайні, незмінювані дані які можна підставити в потрібну функцію та отримати за необхідним законом. В програмі додано можливість переглядати стандартний графік та стандартизований по першому та другому закону, хоч один з них, так і декілька разом. За необхідністю легко доповнити функціонал програми, додавши інші закони або модифікувавши наявні.

Наступна корисна функція – можливість збереження окремих інтервалів. Для початку це потрібно для ручного збереження користувачем, в подальшому це може виконуватись автоматично. Виконавши попередні завдання, на кожному етапі роботи програми окремо зберігаються дані, які відображаються на графіку. Для їх збереження знову ж таки скористаємось збереженням даних у файл. Для цього потрібно реалізувати логіку створення файлів, з якої можна однозначно дізнатись для якого вхідного файлу було викликано збереження та за який проміжок часу збереглися дані.

Для подальшої реалізації була створена змінна gen_files_dir = "D:/holter_files/file", яка регулює в яку папку потрібно зберігати дані. Для ідентифікації пацієнта буде використовуватись назва вихідного файлу Холтера. Всередині будуть створюватись нові папки з іменами, які відповідають початку відліку інтервалу, що буде туди записуватись. Для цього створена функція, яка переводить час в секундах в загальноприйнятий формат часу з годинами, хвилинами, секундами та

мілісекундами, яка є більш коротким представленням шкали часу з меню програми.

Якщо виникає потреба в збереженні інтервалу, це означає що на певному сигналі були знайдені підозрілі значення, тому може зустрітись випадок, коли в інших сигналах краще показано ту чи іншу особливість, саме тому при збереженні інтервалу я буду зберігати окремо всі сигнали з нього. Так як в одного пацієнта не можуть співпадати сигнали, то при створенні файлів для конкретного пацієнта їх можна зберігати за таким макетом, як “signal{n}.{ext}”, де замість n підставляється номер сигналу, а замість ext – файлове розширення в якому потрібно зберігати дані. Також буде збереження графіків окремих сигналів в форматі .pdf, тому що в цьому випадку вони будуть зберігатись як векторне зображення, його використання дозволяє масштабувати зображення без втрати його чіткості.

3.3 Огляд роботи програми

Після всіх необхідних налаштувань середовища запуску програми можна перейти до налаштувань програми. Всі дані за замовчуванням підібрані так, щоб програму можна було запустити з будь-якого комп'ютеру. Все що потрібно, це прописати явний шлях до місця зберігання файлу пацієнта. Для ілюстрації роботи програми так само буде взятий файл 2@1951-01-31.edf з розділу 3. Після цього можна переходити до головного класу main.py та запустити. Якщо всі дані вказано вірно, то маємо отримати наступні повідомлення в консоль(Рисунок 2).

```
D:/holter_files/2@1951-01-31.edf_generated_0.txt just created
D:/holter_files/2@1951-01-31.edf_generated_1.txt just created
D:/holter_files/2@1951-01-31.edf_generated_2.txt just created
computing!
please wait
current status is 1
container 0 successfully filled...
container 1 successfully filled...
container 2 successfully filled...
```

Рисунок. 2. Перший запуск програми

Перші три повідомлення показують, що допоміжні файли успішно створені й виводяться абсолютні посилання, які можна записати в пошукову строку пристрою та відкрити. Як можна бачити, дані цього пацієнта складаються з трьох сигналів. Повідомлення "current status is 1" означає, що всі дані успішно отримані та ініціалізація всіх потрібних параметрів пройшла успішно. Останні три повідомлення сигналізують, що наповнення масивів даними завершилось й пройшло успішно. Наступні запуски програми відбуваються швидше, адже не потрібно ще раз розбивати загальні дані за сигналами. На Рисунку 3 зображено графік серцевого ритму пацієнта за перші 60 хвилин.

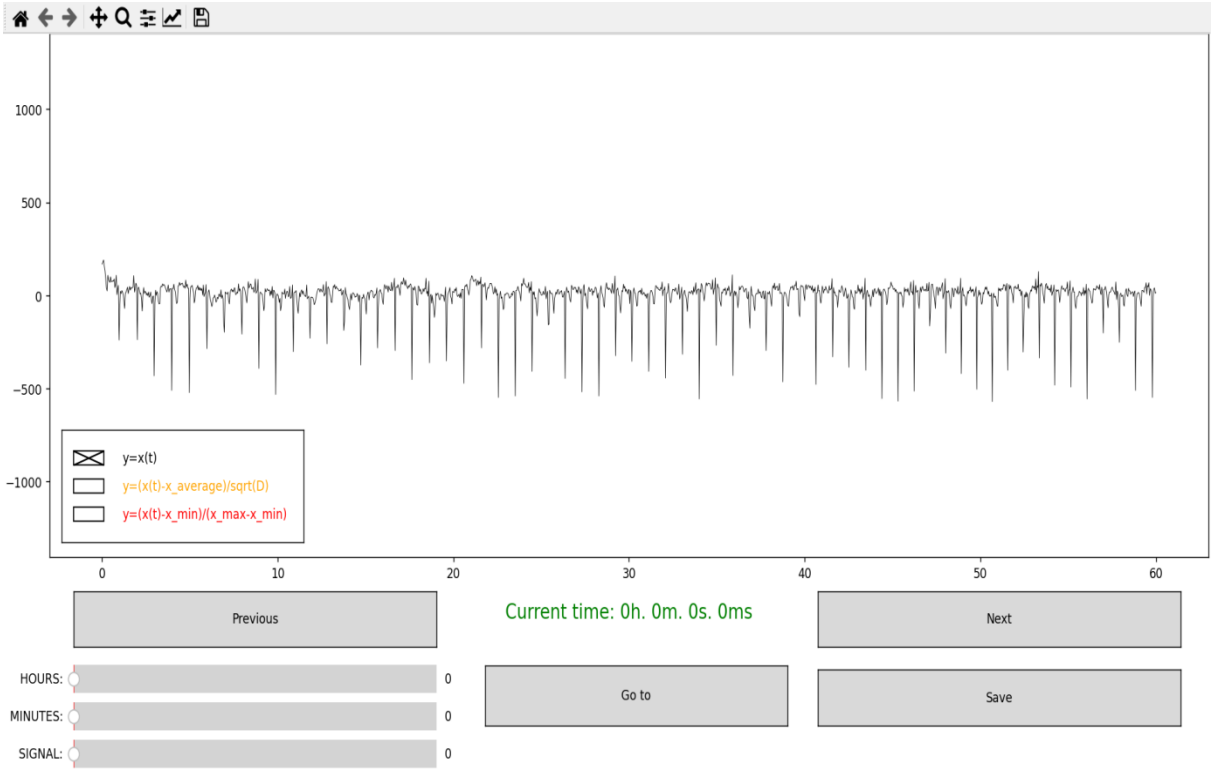

Рисунок. 3. Головне меню при першому запуску програми

Розглянемо детальніше меню програми. Масштаб графіку обирається автоматично відносно всіх значень, що знаходяться на ньому й не змінюється при переході до інших інтервалів. Посередині відображається час, який відповідає за початок відліку. Для того, щоб точно дізнатись в який момент часу відбулось скорочення серця, необхідно до цього часу додати координату x, яка відповідає за це значення. Для переходу до наступного чи попереднього інтервалу можна використовувати кнопки “Previous” та “Next” відповідно. Якщо потрібно перейти до будь-якого іншого інтервалу можна скористатись шкалою “HOURS” та “MINUTES”, встановивши потрібне значення та натиснути на кнопку “Go to”. Для вибору поточного сигналу можна скористатись шкалою “SIGNAL”, встановивши доступне значення. Тепер спробуємо перейти до даних,

отриманих з 4 години 54 хвилини, які знаходяться в першому сигналі. Для цього обираємо потрібні значення на шкалах та натискаємо “Go to”. Як можна бачити, дані успішно відображені(Рисунок 4).

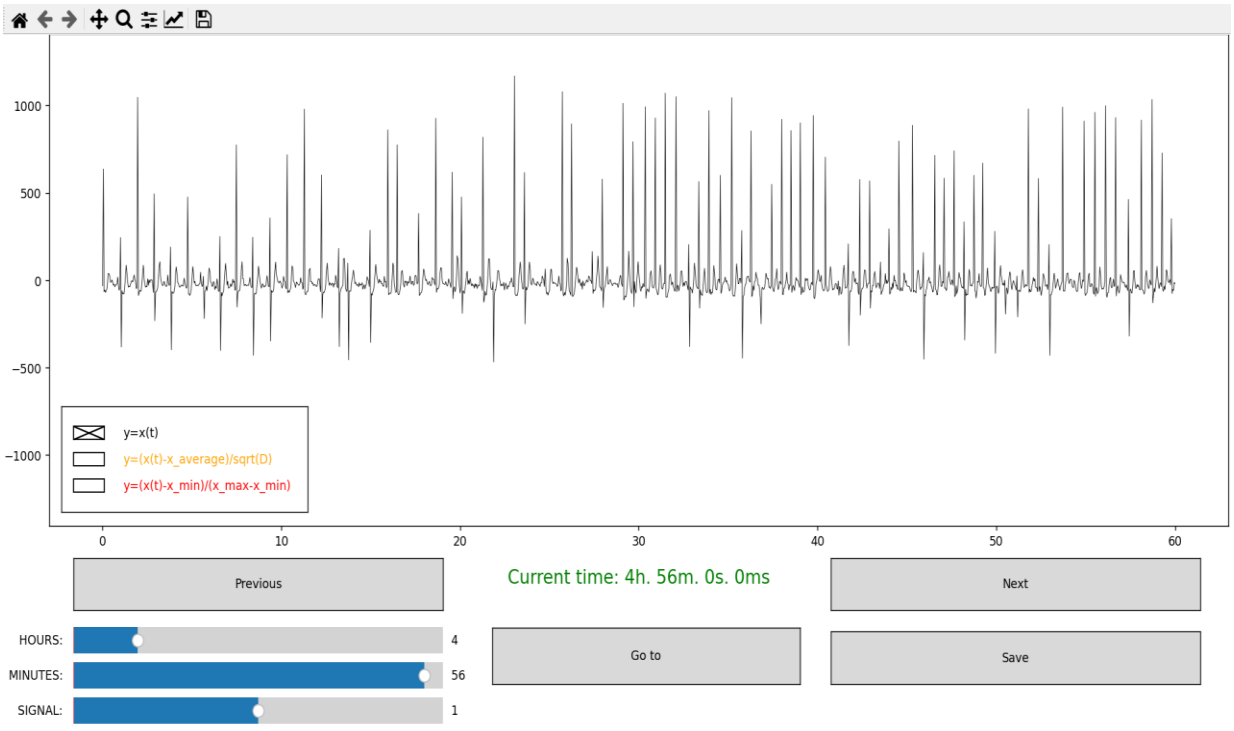

Рисунок. 4. Перехід до віддалених інтервалів графіку

Тепер емулюємо ситуацію, в якій лікар-кардіолог хоче зберегти інтервали з підозрілими значеннями. Припустимо, що потрібно зберегти перші десять хвилин другої години загального інтервалу, для цього потрібно обрати проміжки та натискати “Save”. Отримуємо результат, що зображений на Рисунку 5 та Рисунку 6.

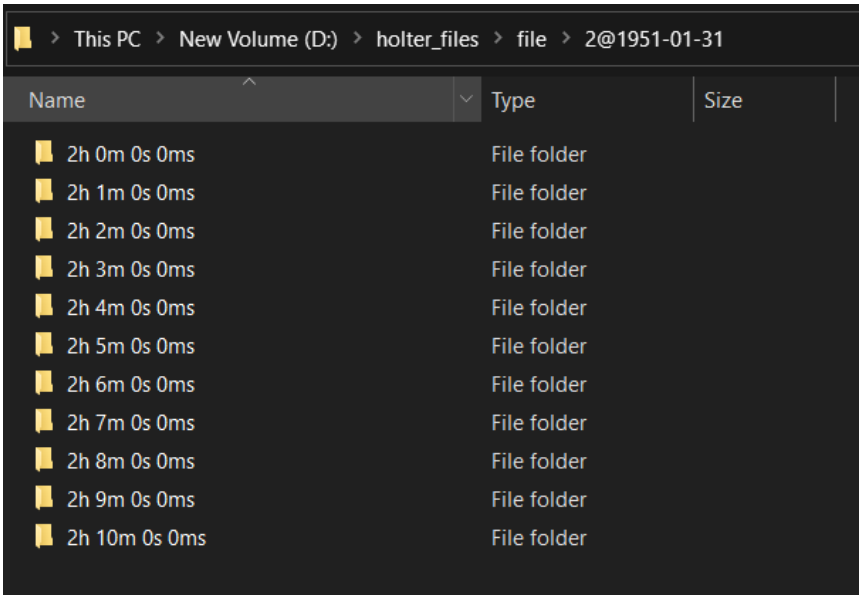

Рисунок. 5. Демонстрація структури файлової системи

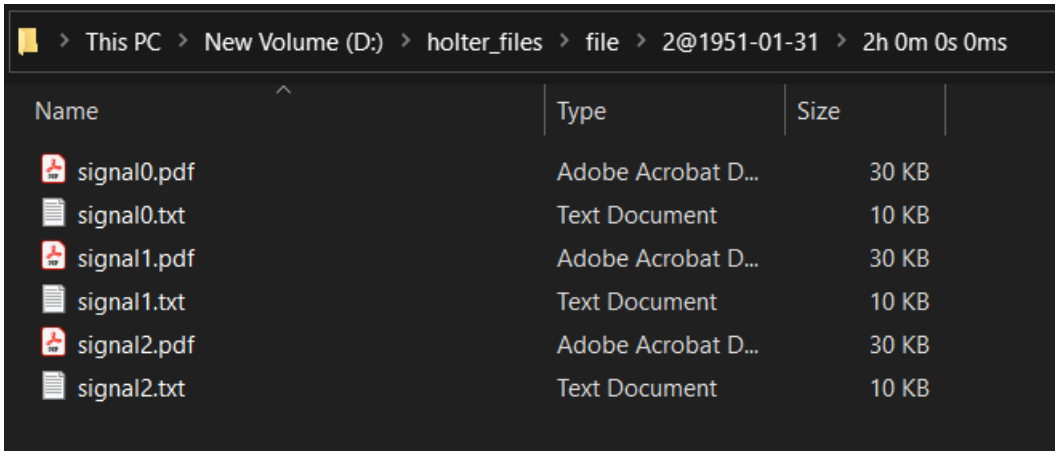

Рисунок 6. Вміст файлової системи

3.4 Порівняння програми з конкурентами

Потрібно зазначити, що існують інші, схожі за функціоналом програми, які можуть виконувати деякі поставлені задачі. Розглянемо одну з них, яка називається Online EDF Viewer - Bilal Zonjy [13]. Програма створена для розшифрування та виведення на екран інформації, що зашифрована в файлах .edf формату. Переглядач написаний за допомогою JavaScript, HTML і CSS. Він повністю працює на стороні клієнта за допомогою браузера (для розгортання сторінки потрібен Інтернет, але потім його можна запускати в автономному режимі).

Для демонстрації роботи я буду використовувати той самий приклад з попереднього розділу (2@1951-01-31.edf). Нагадаю, що в ньому зберігаються 250 даних на секунду для кожного з трьох сигналів за одну добу. Особливістю програми є те, що при завантаженні файлу пацієнта потрібно стабільне підключення до інтернету, від якого залежить час, який потрібно витратити на генерацію графіку. Для запуску потрібно завантажити файл пацієнта та обрати що потрібно зобразити. Від завантаження демонстраційного прикладу і до створення графіку в середньому займає близько 2 хвилин. На тому самому пристрої моя програма займає близько 20-25 секунд. Тож першою перевагою можна виокремити менший час на обробку даних та отримання графічного відображення, а також вищу стабільність і меншу кількість вимог. Недоліком є вужчий список пристроїв, де її можна запустити. Програму даної кваліфікаційної роботи зручно використовувати локально, наприклад на комп’ютерах, в той час як Online EDF Viewer можна запускати на мобільних пристроях.

Тепер можна розглянути детальніше меню програми. На Рисунку 9 можна побачити головне меню програми.

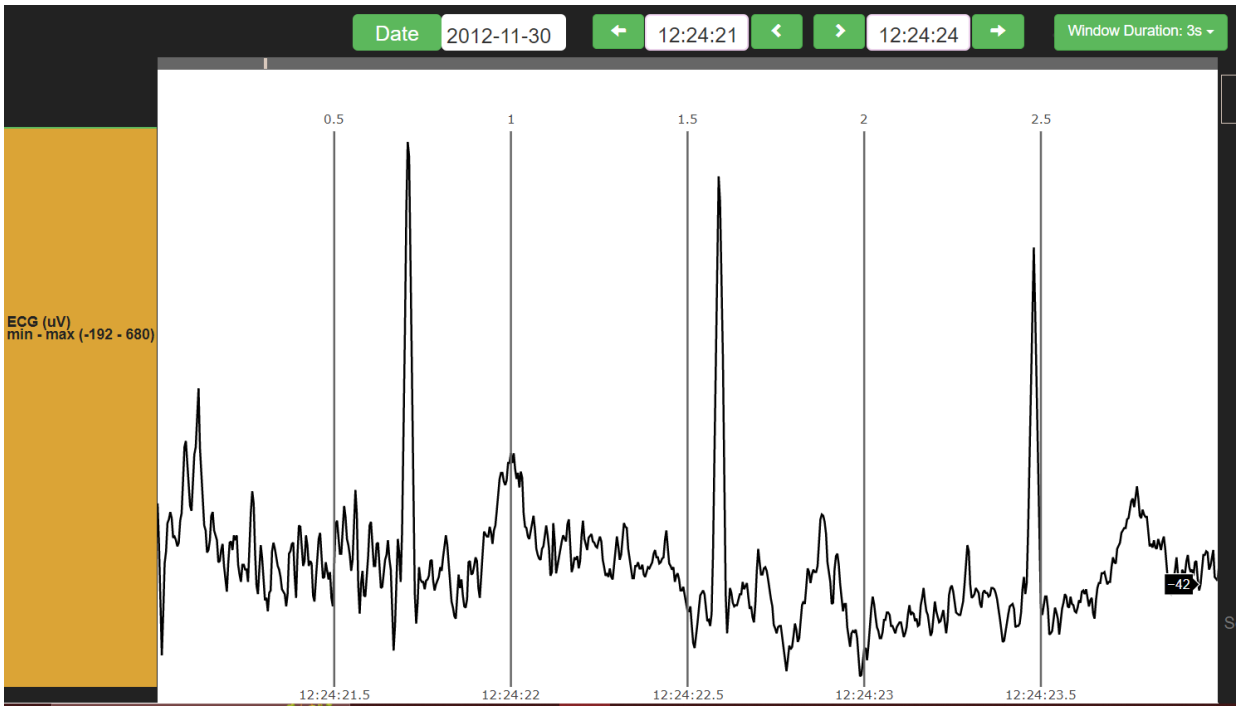

Рисунок 9. Online EDF Viewer, головне меню

За замовчуванням тривалість одного інтервалу – 7 секунд, проте є можливість обирати довжину від 1 до 60 секунд. Також можна навести мишкою на екран та побачити координати нанесених точок. При повторному запуску програми потрібно ще раз завантажувати файл.

В цієї програми є декілька інших функцій, які можуть фільтрувати значення, проте основною є можливість перегляду серцевого ритму пацієнта за певний час. Можна виокремити ще один недолік – вибір масштабу. Для кожного інтервалу він обирається повторно, і після переходу до інших інтервалів більшість даних можуть відображатись як в верхній частині екрану, так і в нижній, щоразу стрибаючи вгору та вниз. В моїй

програмі такої проблеми немає, адже вибір масштабу відбувається один раз за допомогою аналізу всіх даних за вже визначеним алгоритмом.

В програмі немає можливості вибору кількості даних на секунду, які відображаються, тому при довгих інтервалах виникає проблема, коли дані відображаються щільно, й без збільшення важко розпізнати графік, проте розробник надав можливість виділяти окремі частини даних або збільшувати їх масштаб. На Рисунку 10 можна бачити як буде зображено певний інтервал довжиною в 60 хвилин.

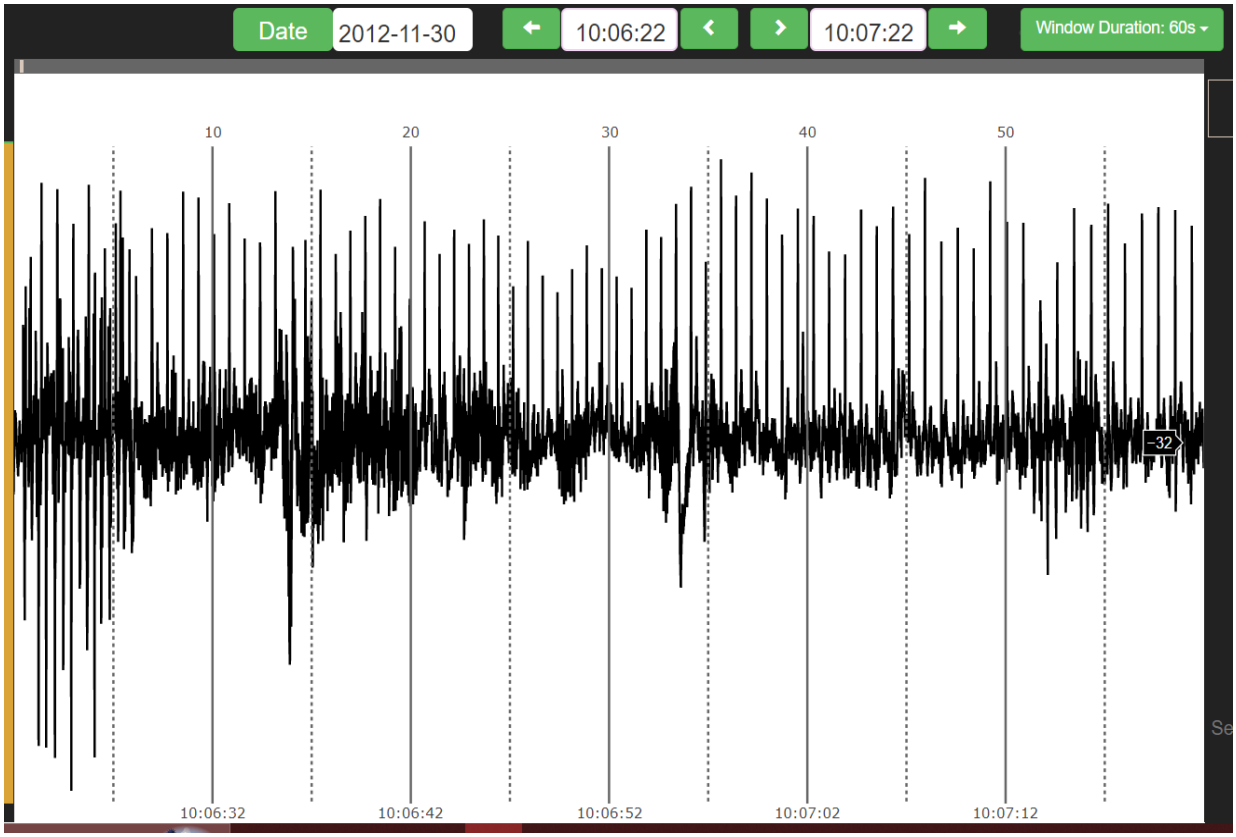

Рисунок 10. Online EDF Viewer, демонстрація 2

Можна підсумувати отриману інформацію у вигляді порівняльної таблиці, де буде коротко описано підходи в роботі програм (Таблиця 1).

Програма/особливості	Програма кваліфікаційної роботи	Online EDF Viewer - Bilal Zonjy
Візуалізація даних ЕКГ	Так	Так
Перехід до віддалених інтервалів	Так	Так
Вибір довжини інтервалів	Так	Так
Вибір щільності даних	Так	Ні
Чисельне представлення координат на графіку	Так	Так
Масштабування графіку	Так	Так
Збереження даних	Так	Ні
Фільтрація даних	Ні	Так
Час створення графіку	1 хвилина при першому запуску, 20-25 секунд кожен наступний	Стабільно, близько 2 хвилин
Програмні вимоги	Python 3.5+, Numpy1.9.1+	Браузер + інтернет

Таблиця 1. Порівняльна характеристика програм.

Як можна бачити, для поставлених задач створена програма відповідає більше, розширює функціонал Online EDF Viewer з певними вдосконаленнями. Можна виділити декілька переваг, такі як менший час виконання та більша кількість тонких налаштувань, які допоможуть краще візуалізувати дані для різних випадків, а також розширений функціонал. Недоліком є менша кросплатформенність, тому що для запуску виникає потреба в завантаженні програмного середовища локально, в той час як Online EDF Viewer не потребує встановлення та налаштування програмного середовища користувачем.

ВИСНОВКИ

Отже під час виконання кваліфікаційної роботи було проведено дослідження щодо особливостей збору інформації серцево-судинної системи. проаналізовано та обрано оптимальні засоби для можливостей зчитування, розділення за ознаками та збереження даних ЕКГ по Холтеру. Підсумовуючи отримуємо, що аналіз серцевих захворювань здійснюється шляхом виявлення початкових стадій хвороби за допомогою електрокардіограми, зокрема моніторингу по Холтеру. ЕКГ є головним неінвазійним методом дослідження роботи серця та визначення його аномалій. При наявності хвороби серця своєчасно зроблена ЕКГ може запобігти серцевому нападу або і смерті.

З огляду це виникає необхідність використання алгоритмічних та програмних засобів попередньої обробки великих масивів даних. В даній кваліфікаційній роботі розроблено програмне забезпечення, яке реалізує наступні можливості:

- виділення даних окремих відведень (дані кожного відведення створюються парою датчиків) з вихідного файлу Холтера;
- розподіл отриманої інформації за її ознаками та збереження відповідних сигналів в окремих файлах;
- розбиття даних за вказаними часовими інтервалами з їх подальшою візуалізацією та можливостями масштабування, прокрутки, збереження.

В даній роботі було використано можливості середовища розробки PyCharm Community Edition, мова програмування Python, бібліотека PyEDFlib. Розроблена програма може бути корисна лікарям-кардіологам для виділення аномалій серцевого ритму для більш детального аналізу. Загалом програмне забезпечення може допомогти лікарю швидше поставити діагноз.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Krak, O. Stelia, A. Pashko, M. Efremov, O. Khorozov. Electrocardiogram Classification Using Wavelet Transformations // Conference Paper, February 2020: https://www.researchgate.net/publication/341243120
2. Сіренко Ю. М. Стан проблеми серцево-судинної захворюваності та смертності в Україні № 2 (258) додаток 1. 2022: с. 11-14.
3. B. Paudel, K. Paudel, The Diagnostic Significance of the Holter Monitoring in the Evaluation of Palpitation//Original Article, 01 march 2013: p. 480-483.
4. J. G. Andrade, L. Macle, A. Verma, Atrial Fibrillation Guidelines// Canadian Cardiovascular Society, 2018, p. 1-31.
5. I. Vosko, A. Zirlik, H. Bugger, Impact of COVID-19 on Cardiovascular Disease //Review, 11 February 2023: https://www.mdpi.com/1999-4915/15/2/508, p. 1-31.
6. Ляшко А., Візуалізація та розмітка даних ЕКГ для виділення ефективних характеристичних ознак: матеріали XXI міжнар. наук.-прак. конф. "Шевченківська весна - 2023", м. Київ, 14 квітня 2023, с. 93-94.
7. World Health Organization [Електронний ресурс] URL: https://www.who.int/health-topics/cardiovascular-diseases
8. Інститут когнітивного моделювання [Електронний ресурс] URL: https://www.cognitive.com.ua/
9. European Data Format, overview [Електронний ресурс] URL: https://www.edfplus.info/
10. Python 3.11.2 documentation [Електронний ресурс] URL: https://docs.python.org/3/
11. Python EDF/BPF reader [Електронний ресурс] URL: https://pyedflib.readthedocs.io/en/latest/

12. The best programming languages in 2022 [Електронний ресурс] URL: https://www.halo-lab.com/blog/top-programming-languages
13. Online EDF Viewer - Bilal Zonjy [Електронний ресурс] https://bilalzonjy.github.io/EDFViewer/EDFViewer.html

ДОДАТОК А

container_Manager.py:

```
import os
from main_program import Variables
import fileManager
class Container:
    def filling_containers(self):
        self.current_status = fileManager.createServiceFiles(self.dir)
        print("current status is " + str(self.current_status))

    def __init__(self, directory):
        self.dir = directory
        self.current_status = 0
        self.delta = Variables.FilesConstant.get_points_in_one_sec
        self.filling_containers()
        self.points_number = 0
        self.time_duration = 0

    # get operation time and length
    def fill_input_param(self, input_line):
        num_list = []
        num = ''
        for char in input_line:
            if char.isdigit():
                num = num + char
            else:
                if num != '':
                    num_list.append(int(num))
                    num = ''
        if num != '':
            num_list.append(int(num))
        self.points_number = num_list[1]
        self.time_duration = num_list[2]
```

```
    @staticmethod
    def getSecondsToTime(time_sec, points_in_sec):
        return str(time_sec / points_in_sec)

    def writeFileToListAndDate(self, output_list0, out_date, file_number):
        if not self.current_status:
            return 0
        reading_path = self.dir + fileManager.ending + str(file_number) +
Variables.FilesConstant.text_type
        check_file = os.path.isfile(reading_path)
        if check_file:
            inp_file = open(reading_path)
            j = -1
            self.fill_input_param(inp_file.readline())
            point_in_second = (self.getPointsAmount() / self.time_duration)
            for i in inp_file:
                j = j + 1
                if j % self.delta == 0:
                    out_date.append(float(self.getSecondsToTime(j, point_in_second).strip()))
                    output_list0.append(float(i.strip()))
            inp_file.close()
            print("container "+str(file_number)+" successfully filled...")
            return 1
        print("An error occurred")
        return 0

    def writeFileToList(self, output_list0, file_number):

        if not self.current_status:
            return 0
        reading_path = self.dir + fileManager.ending + str(file_number) +
Variables.FilesConstant.text_type
        check_file = os.path.isfile(reading_path)
        if check_file:
```

```
            inp_file = open(reading_path)
            j = -1
            self.fill_input_param(inp_file.readline())
            point_in_second = (self.getPointsAmount() / self.time_duration)
            for i in inp_file:
                j = j + 1
                if j % self.delta == 0:
                    output_list0.append(float(i.strip()))
            inp_file.close()
            print("container " + str(file_number) + " successfully filled...")
            return 1
        print("An error occurred")
        return 0

    def getDuration(self):
        return self.time_duration

    def getPointsAmount(self):
        return self.points_number
import os.path
import numpy as np
```

fileManager.py:

```
import pyedflib as edfl
from main_program import Variables

ending = "_generated_"
def write_file(dir_name):
    fr = edfl.EdfReader(dir_name)
    n = fr.signals_in_file

    sigbufs = np.zeros((n, fr.getNSamples()[0]))
    for i in np.arange(n):
        sigbufs[i, :] = fr.readSignal(i)
```

```
    for i in range(n):
        with open(dir_name + ending + str(i) + Variables.FilesConstant.text_type, 'w') as f:

            f.write("signal=" + str(i) + "; points=" + str(len(sigbufs[i])) + "; duration=" + str(
                fr.getFileDuration()) + "\n")

            for j in range(len(sigbufs[i])):
                f.write(str(sigbufs[i][j]) + "\n")
        print(dir_name + ending + str(i) + Variables.FilesConstant.text_type + " just created")


def createServiceFiles(file_dir):
    if not os.path.isfile(file_dir):
        print("file doesn`t exist")
        return -1
    is_all_created = True
    n = edfl.EdfReader(file_dir).signals_in_file
    for i in range(n):
        check_file = os.path.isfile(file_dir + ending + str(i) + Variables.FilesConstant.text_type)
        if not check_file:
            write_file(file_dir)
            is_all_created = False
    if is_all_created:
        print("files had been created")
    else:
        print("computing!\n" + "please wait")
    return 1


def save_input_container(file_directory, file_name, st_time, fin_time, y_s):
    if not os.path.isdir(file_directory):
        print("incorrect file directory;")
        return 0
    with open(file_directory + "/" + file_name, 'w') as f:
```

```
    f.write("START TIME: " + st_time + "\n")
    f.write("FINISH TIME: " + fin_time + "\n")
    for i in y_s:
        f.write(str(i) + "\n")
path = os.path.join(file_directory + "/" + file_name)
print("saved to "+path)
```

ДОДАТОК В

graph_Manager.py:

```
import os

import StaticMethods
import numpy as np
from main_program import container_Manager as cm, Variables
import matplotlib.pyplot as plt
import pylab
import fileManager as fm
from matplotlib.widgets import Button, Slider, CheckButtons

draw_first_plot = True
draw_second_plot = False
draw_third_plot = False
c = ['black', 'orange', 'red']



class Graph:
    delta_time = Variables.GraphConstant.delta_time
    startTime = 0
    finishTime = startTime + delta_time

    current_xs = []
    current_ys0 = []
    current_ys1 = []
    current_ys2 = []
    lim = 0
    on_changed_sign = Variables.FilesConstant.current_signal
    file_n = ""

    def getCurrentYS(self, s=Variables.FilesConstant.current_signal):
        if s == 0:
            return self.current_ys0
```

```
    if s == 1:
        return self.current_ys1
    return self.current_ys2

def file_init(self):
    self.file_n = Variables.GraphConstant.gen_files_dir
    if not os.path.isdir(self.file_n):
        os.mkdir(self.file_n)
    mas = str.split(Variables.FilesConstant.file_directory, "/")
    n = str(mas[len(mas) - 1])
    n = n[0:len(n) - 4]
    self.file_n = self.file_n + "/" + n
    if not os.path.isdir(self.file_n):
        os.mkdir(self.file_n)

def start_init(self):

    n = cm.Container(Variables.FilesConstant.file_directory)
    n.writeFileToList(self.current_ys0, 0)
    n.writeFileToList(self.current_ys1, 1)
    n.writeFileToListAndDate(self.current_ys2, self.current_xs, 2)

    limit = (StaticMethods.predictionLimits(self.getCurrentYS(), 8))
    limit[0] = np.abs(limit[0])
    limit[1] = np.abs(limit[1])
    self.lim = np.max(limit)

    self.file_init()

def init_value(self, x_s, y_s):

    current_position = 0
    while self.current_xs[current_position] < self.startTime:
        current_position = current_position + 1
```

```
    while self.current_xs[current_position] <= self.finishTime:
        x_s.append((self.current_xs[current_position]) - self.startTime)
        y_s.append(self.getCurrentYS(self.on_changed_sign)[current_position])
        current_position = current_position + 1

def init_to_file(self, x_s, y0_s, y1_s, y2_s):
    current_position = 0
    while self.current_xs[current_position] < self.startTime:
        current_position = current_position + 1
    while self.current_xs[current_position] <= self.finishTime:
        x_s.append((self.current_xs[current_position]) - self.startTime)
        y0_s.append(self.getCurrentYS(0)[current_position])
        y1_s.append(self.getCurrentYS(1)[current_position])
        y2_s.append(self.getCurrentYS(2)[current_position])
        current_position = current_position + 1

def gen_files(self):

    next_path = self.file_n + "/" + str(StaticMethods.sec_to_time_short(self.startTime))
    if not os.path.isdir(next_path):
        os.mkdir(next_path)
    global sign_slider
    s = self.on_changed_sign
    for i in range(3):
        self.on_changed_sign = i
        sign_slider.set_val(i)
        self.redrawFigure()
        fig.savefig(next_path + "/" + "signal" + str(
            self.on_changed_sign) + Variables.FilesConstant.screen_type)
    self.on_changed_sign = s
    sign_slider.set_val(s)
    x_s0 = []
    y_s_0 = []
    y_s_1 = []
```

```
    y_s_2 = []
    self.init_to_file(x_s0, y_s_0, y_s_1, y_s_2)
    start_t = StaticMethods.sec_to_time_short(self.startTime)
    finish_t = StaticMethods.sec_to_time_short(self.finishTime)
    fm.save_input_container(next_path, "signal" + str(0) + ".txt", start_t, finish_t, y_s_0)
    fm.save_input_container(next_path, "signal" + str(1) + ".txt", start_t, finish_t, y_s_1)
    fm.save_input_container(next_path, "signal" + str(2) + ".txt", start_t, finish_t, y_s_2)

def redrawFigure(self):
    x = []
    y = []
    self.init_value(x, y)
    ax1.clear()
    global draw_first_plot, draw_second_plot, draw_third_plot, c
    if draw_first_plot:
        ax1.plot(x, y, linewidth=0.5, color=c[0])
    if draw_second_plot:
        ax1.plot(x, StaticMethods.normalize_zscore(y), linewidth=0.5, color=c[1])
    if draw_third_plot:
        ax1.plot(x, StaticMethods.normalize_mean(y), linewidth=0.5, color=c[2])

    ax1.set_ylim([-self.lim, self.lim])
    add_time(ax1, StaticMethods.convertSecondsToTime(self.startTime))
    pylab.draw()

def changeSlider(self):
    hour = self.startTime // 3600
    hour_slider.set_val(hour)
    minute_slider.set_val((self.startTime - hour * 3600) // 60)
    self.redrawFigure()

def next(self, event):
    if self.finishTime + self.delta_time <= self.current_xs[len(self.current_xs) - 1]:
        self.startTime = self.delta_time + self.startTime
```

```
            self.finishTime = self.delta_time + self.finishTime
            self.changeSlider()

    def prev(self, event):
        if self.startTime - self.delta_time >= 0:
            self.startTime = self.startTime - self.delta_time
            self.finishTime = self.finishTime - self.delta_time
            self.redrawFigure()
            self.changeSlider()

    def saveToFile(self, event):
        self.gen_files()

    def set_time(self, hour, minute):
        self.startTime = hour * 3600 + minute * 60
        self.finishTime = self.startTime + self.delta_time

    def buttonGoTo(self, event):
        self.set_time(hour_slider.val, minute_slider.val)
        self.redrawFigure()

    def sign_slider(self, event):
        global sign_slider
        self.on_changed_sign = sign_slider.val
        self.redrawFigure()


def add_time(ax, curr_time):
    ax.text(0.5, -0.125, curr_time,
            verticalalignment='bottom', horizontalalignment='center',
            transform=ax1.transAxes,
            color='green', fontsize=15)
```

```
def add_plot_menu():
    rax = pylab.axes([0.05, 0.32, 0.2, 0.15])
    global check
    global c
    check = CheckButtons(rax, ('y=x(t)', 'y=(x(t)-x_average)/sqrt(D)', 'y=(x(t)-x_min)/(x_max-x_min)'),
(draw_first_plot, draw_second_plot, draw_third_plot))
    [rec.set_color(c[i]) for i, rec in enumerate(check.labels)]


def func(label):
    if label == 'y=x(t)':
        global draw_first_plot
        draw_first_plot = not draw_first_plot
    elif label == 'y=(x(t)-x_average)/sqrt(D)':
        global draw_second_plot
        draw_second_plot = not draw_second_plot
    elif label == 'y=(x(t)-x_min)/(x_max-x_min)':
        global draw_third_plot
        draw_third_plot = not draw_third_plot
    global gr
    gr.redrawFigure()


def start_plot():
    global gr
    gr = Graph()
    gr.start_init()
    global fig
    fig = plt.figure(figsize=(15, 7))
    global ax1
    ax1 = fig.add_subplot()
    gr.redrawFigure()

    add_time(ax1, StaticMethods.convertSecondsToTime(gr.startTime))
```

```
fig.subplots_adjust(left=0.04, right=0.998, top=1.0, bottom=0.3)

axes_button_add = pylab.axes([0.675, 0.18, 0.3, 0.075])
axes_button_remove = pylab.axes([0.06, 0.18, 0.3, 0.075])
axes_button_save_to_file = pylab.axes([0.675, 0.075, 0.3, 0.075])
axes_slider1 = pylab.axes([0.06, 0.1, 0.3, 0.075])
axes_slider2 = pylab.axes([0.06, 0.05, 0.3, 0.075])
axes_button_go_to = pylab.axes([0.4, 0.075, 0.25, 0.08])
axes_sign_slider = pylab.axes([0.06, 0.0, 0.3, 0.075])

button_add = Button(axes_button_add, 'Next')
button_remove = Button(axes_button_remove, 'Previous')
button_save_to_file = Button(axes_button_save_to_file, 'Save')

global hour_slider, minute_slider, sign_slider
hour_slider = Slider(axes_slider1, "HOURS: ", 0, 23, 0, valstep=1)
minute_slider = Slider(axes_slider2, "MINUTES: ", 0, 59, 0, valstep=1)
button_go_to = Button(axes_button_go_to, "Go to")
sign_slider = Slider(axes_sign_slider, "SIGNAL: ", 0, 2, 0, valstep=1)
sign_slider.set_val(Variables.FilesConstant.current_signal)
sign_slider.on_changed(gr.sign_slider)

button_add.on_clicked(gr.next)
button_remove.on_clicked(gr.prev)
button_save_to_file.on_clicked(gr.saveToFile)
button_go_to.on_clicked(gr.buttonGoTo)
add_plot_menu()
global check
check.on_clicked(func)
pylab.show()
```

ДОДАТОК С

StaticMethods.py:

```
import numpy as np


def getMinimum(input_double):
    res = input_double[0]
    for i in input_double:
        if res > i:
            res = i
    return res


def getMaximum(input_double):
    res = input_double[0]
    for i in input_double:
        if res < i:
            res = i
    return res


# enter amount_of_period  percent of filtered data
# 1                 68
# 2                 95
# 3                 99.7

def predictionLimits(data, amount_of_period=1):
    res = []
    mean_val = np.mean(data)
    st_dev = np.std(data)
    if amount_of_period < 1:
        amount_of_period = 1
    res.append(mean_val - amount_of_period * st_dev)
    res.append((mean_val + amount_of_period * st_dev))
```

```
    return res


def convertSecondsToTime(s):
    hours = s // 3600
    minutes = (s - hours * 3600) // 60
    seconds = s - hours * 3600 - minutes * 60
    milis = (s - hours * 3600 - minutes * 60 - seconds) * 100
    return "Current time: " + str(hours) + "h. " + str(minutes) + "m. " + str(seconds) + "s. " + str(milis) +
"ms"


def sec_to_time_short(s):
    hours = s // 3600
    minutes = (s - hours * 3600) // 60
    seconds = s - hours * 3600 - minutes * 60
    milis = (s - hours * 3600 - minutes * 60 - seconds) * 100
    return str(hours) + "h " + str(minutes) + "m " + str(seconds) + "s " + str(milis) + "ms"


def timetosec(hours, minutes, seconds=0, milis=0):
    return hours * 3600 + minutes * 60 + seconds + milis / 100


def normalize_zscore(input_data):
    out_data = []
    disp_sqrt = np.sqrt(np.var(input_data))
    mean_v = np.mean(input_data)
    for i in input_data:
        out_data.append((i - mean_v) / disp_sqrt)
    return out_data


def normalize_mean(input_data):
```

```
    out_data = []
    min_v = np.min(input_data)
    down = np.max(input_data) - min_v
    for i in input_data:
        out_data.append((i - min_v) / down)
    return out_data
```

Variables.py

```
class FilesConstant:
    file_directory = "D:/holter_files/2@1951-01-31.edf"
    current_signal = 1
    get_points_in_one_sec = 10
    screen_type = ".pdf"
    text_type = ".txt"
class GraphConstant:
    delta_time = 60
    gen_files_dir = "D:/holter_files/file"
```

main.py:

```
from main_program import graph_Manager as gm
if __name__ == "__main__":
    gm.start_plot()
```