КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра теоретичної кібернетики

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

ВИЯВЛЕННЯ QRS-КОМПЛЕКСУ З ВИКОРИСТАННЯМ ЛОКАЛЬНОГО СПЛАЙНУ ДРУГОГО ПОРЯДКУ

Виконав студент 4-го курсу

Андросюк Кирило Євгенович _____

(підпис)

Науковий керівник:

професор, доктор фіз.-мат. наук

Крак Юрій Васильович _____

(підпис)

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

Студент _____

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теоретичної кібернетики

«____» _____ 202__ р.,

протокол № _____

Завідувач кафедри

Ю. В. Крак _____

(підпис)

Київ-2021

1 РЕФЕРАТ

Обсяг роботи 45 сторінок, 31 ілюстрацій, 1 таблиця, 29 джерел посилань.

QRS-КОМПЛЕКС, ВИЯВЛЕННЯ, R-ЗУБЕЦЬ, ЕЛЕКТРОКАРДІОГРАМА, КУБІЧНИЙ СПЛАЙН.

Об’єктом роботи є процес виявлення QRS-комплексу.

Предметом роботи є програмний засіб для виявлення QRS-комплексу.

Метою роботи є створення програмного засобу для виявлення QRS-комплексу.

Методи розроблення: методи апроксимації та інтерполяції табличних функцій.

Інструменти розроблення: текстовий редактор Vim, мова програмування C++, науковий графічний пакет Grapher.

Результати роботи: виконано загальний огляд методів виявлення QRS-компексу, проаналізовано переваги та недоліки відповідних методів, створено програмну реалізацію алгоритму виявлення QRS-комплексу.

Програмна реалізація може використовуватись у мікроконтроллерах автоматизованих систем моніторингу пацієнтів задля отримання частоти сердцевого ритму з метою його подальшого аналізу.

1 ЗМІСТ

2 СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ЕКГ – електрокардіограма;

ВСР – варіабельність серцевого ритму;

3 ВСТУП

Електрокардіограма(ЕКГ) відображає електричну активність серця. Кожне скорочення серця виробляє електричний імпульс, який вловлюється електродами, розміщеними на шкірі. Серцебиття виробляє серію хвиль, які спричинені зміною напруги серцевих клітин.

Кожен серцевий цикл складається з двох стадій - скорочення та розслаблення, що в термінах електрики називається деполяризацією та реполяризацією. Деполяризація викликає швидку зміну потенціалу клітини (від -90 до 20 мВ), і ця зміна напруги викликає деполяризацію сусідніх клітин. Після деполяризації, клітини, внаслідок їх реполяризації, повертаються до стану спокою[1].

Електрокардіограма надає важливу інформацію про стан серця. Лікарі всього світу використовують ЕКГ для діагностики серцевих захворювань. На сьогоднішній день, інформація, що надається приладами аналізуючими серцеву діяльність, може бути легко оцифрована та оброблена комп'ютером. Таким чином, використовуючи потужність комп'ютеру, ми можемо виявити серцеві захворювання або аномалії.

Для аналізу сигналу ЕКГ важливо знати хвилі, що формують серцебиття, див. Рис. 1. Кожен такт ділиться на три етапи: деполяризація передсердь (P-хвиля), деполяризація шлуночків (QRS-комплекс) і, нарешті, реполяризація шлуночків (T-хвиля). Ці три стадії постійно повторюються в ЕКГ-сигналі, представляючи серцебиття протягом часу.

У дослідженнях, присвячених варіабельності серцевого ритму (ВСР), специфічні риси ЕКГ-хвилі визначаються з QRS-комплексу: як правило, час появи піку зубця R - з якого обчислюється тахограма - або тривалість QRS-комплексу - з якої знаходять частоту дихальних рухів [2].

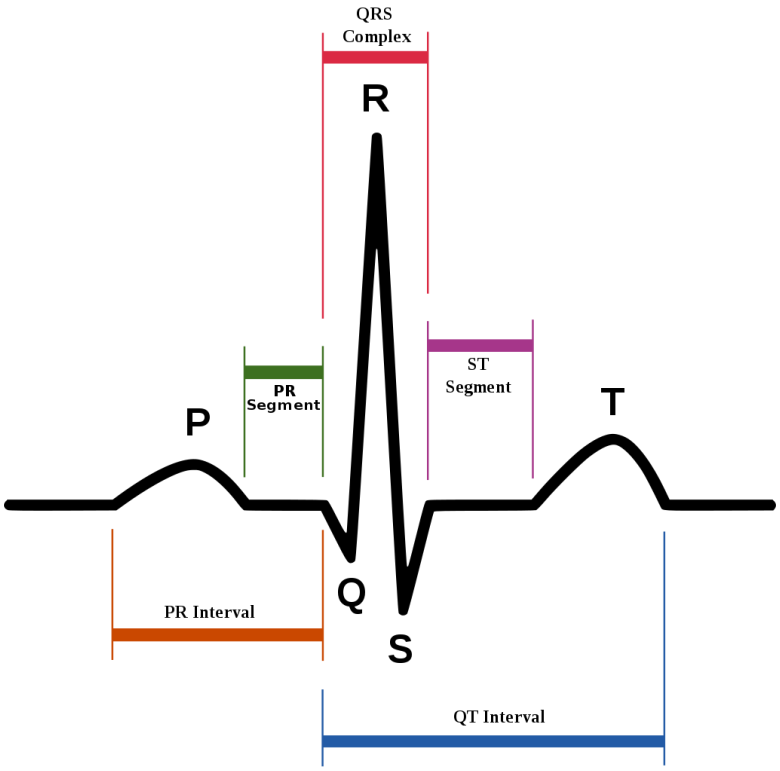

Рисунок 1 - Схематичне зображення нормальної ЕКГ

При обробці ЕКГ дуже важливо дуже точно виявити серцебиття, оскільки це основа для подальшого аналізу. Енергія серцебиття знаходиться в QRS-комплексі. Отже, точний детектор QRS важливий для аналізу ЕКГ. QRS-комплекс має частотний вміст від 10 до 25 Гц[3].

Виявлення QRS - важка задача, оскільки сигнал змінюється протягом часу, і в ньому можуть бути присутні різні типи шуму. Алгоритми виявлення QRS-комплексу вивчалися протягом десятків років. Методи діляться на апаратні та програмні за способом реалізації. Апаратні методи, набагато менш гнучкі та зручні, тому програмні методи перебувають під більшим обговоренням[4]. Програмне виявлення QRS було темою досліджень з вісімдесятих років. Якщо в перші роки ефективність алгоритмів визначалася його обчислювальним навантаженням і складністю, то сьогодні ефективність виявлення є основною метою.

Існують різні види програмних методів, такі як, методи, засновані на математичній моделі, методи, засновані на розпізнаванні образів, методи, засновані на вейвлет-перетворенні, і методи, засновані на нейронних мережах.

У більшості алгоритмів виявлення QRS існує два етапи: попередня обробка та прийняття рішень[5]. На етапі попередньої обробки до сигналу застосовуються різні методи, такі як лінійна та нелінійна фільтрація або згладжування, щоб послабити хвилі P і T, а також шум. Тим часом на стадії прийняття рішень найважливішим завданням є визначення порогових значень, а в деяких випадках і використання методів для відрізнення зубців T. Деякі алгоритми включають ще один етап прийняття рішень задля зменшення помилок.

4 ОГЛЯД МЕТОДІВ ВИЯВЛЕННЯ QRS-КОМПЛЕКСУ

Більшість засобів виявлення QRS розділені на два етапи: етап попередньої обробки та етап прийняття рішень. Майже всі алгоритми використовують етап фільтрації перед виявленням для того, щоб усунути шум і зменшити амплітуду хвиль P і T для полегшення подальшого виявлення[6]. Деякі алгоритми застосовують банк високочастотного і низькочастотного фільтрів, який відомий як смуговий фільтр. Після попередньої обробки сигналу, QRS-комплекс виявляється шляхом встановлення порогу сигналу, де порог може бути фіксованим або адаптивним. Нарешті, більшість алгоритмів використовують додатковий етап прийняття рішень, де правила прийняття рішень застосовуються для зменшення помилкових спрацьовувань. Протягом останніх 30 років було запропоновано багато алгоритмів для виявлення QRS-комплексу. Існує багато підходів - від штучних нейронних мереж чи генетичних алгоритмів до вейвлет-перетворень, банків фільтрів, евристичних методів чи методів машинного навчання[3].

Підходи:

• Алгоритми використовуючі похідну: алгоритми, засновані на фільтрах та похідних[7]. Вони часто використовують фільтр високих частот, а похідна використовується для визначення максимального нахилу, який відповідає комплексу QRS.

• Алгоритми, засновані на цифрових фільтрах: інші алгоритми використовують більш досконалі фільтри[8][9]. Два різні фільтри обробляють ЕКГ, низькочастотний та високочастотний, з різною частотою відсікання, формуючи смугово-фільтрований сигнал. Також порогові значення порівнюються адаптивно[10][11].

• Вейвлети: підходи, засновані на вейвлетах, розкладають сигнал на різні компоненти для аналізу сигналу в різних смугах частот[12][13][14]. Після цього застосовуються фіксовані пороги.

• Нейронні мережі: нейронні мережі використовуються для прогнозування поточних значень сигналу з минулих, і тому застосовують відповідні фільтри для послаблення шуму[14][15].

• Приховані моделі Маркова: ПММ моделюють послідовність даних відповідно до основного ланцюга Маркова[16].

• Генетичні алгоритми: вони мають намір отримати оптимальні поліноміальні фільтри для етапу попередньої обробки та параметри для етапу прийняття рішень[17].

Алгоритм Пана-Томпкінса був великим проривом на момент його публікації. Він використовував найновіші на той час методи, і це алгоритм, який може швидко адаптуватися до змін сигналу і має ефективне виявлення навіть у шумних сигналах.

Алгоритм Гамільтона і Томпкінса був опублікований наступного року, і він дуже схожий на алгоритм Пана-Томпкінса. Він використовує той самий препроцесор з невеликими змінами, але з абсолютно різними правилами прийняття рішень. Теоретично це трохи покращує виявлення порівняно з попереднім алгоритмом.

Алгоритм на основі фазорів - робота, опублікована в 2010 році, яка характеризується своєю надійністю, низькою обчислювальною вартістю та математичною простотою. Результати навіть кращі за результати двох інших алгоритмів, згідно авторам.

Традиційно, ефективне виявлення QRS-комплексу досягається шляхом передискретизації сигналу ЕКГ. Цей підхід дає велику кількість даних. Крім того, ЕКГ можна взяти на частоті Найквіста, але ця процедура вимагає інтерполяції sinc(x) дискретизованого сигналу[18], що надзвичайно трудомістко. Однак в обох випадках потрібно зберігати великий обсяг даних або вимагати великих обчислювальних потужностей. Ці вимоги можуть бути непрактичними в конкретних областях, наприклад, у програмах телемоніторингу, або там, де сигнали повинні записуватися системами з низьким енергоспоживанням та недорогими системами.

У роботі [19] пропонується обчислювально ефективний метод ідентифікації часу появи піку зубця R або тривалості QRS-комплексу з гарною точністю на ЕКГ, отриманій при низьких частотах дискретизації. Метод заснований на ідентифікації та локальній реконструкції QRS-комплексу за допомогою інтерполяції, отриманої за допомогою процедури нульового заповнення. Параметри, що цікавлять, походять лише від QRS-комплексу, і цей компонент хвилі ЕКГ має частотний вміст від 10 до 25 Гц [3], що дозволяє використовувати низькі частоти дискретизації.

5 ІНТЕРПОЛЮЮЧІ СПЛАЙНИ

Кубічні інтерполюючі сплайни класу C^1 – засіб кусково-заданої поліноміальної інтерполяції табличних функцій. Алгоритми їх побудови, обґрунтування, а також ілюстрації різних властивостей наведені в численних роботах[20][21][22]. Серед недоліків кубічних сплайнів класу C^2 ,вони, як правило, не зберігають монотонності вхідних даних, що призводить до коливань побудованих функцій та можуть робити неконтрольовані відхилення значень побудованих функцій.

Одним із способів усунути недоліки глобальних кубічних сплайнів є використання локальних сплайнів. У роботі [23] запропоновано кубічний кусково-заданий поліноміальний інтерполяційний сплайн гладкості C^1. Поліном для вибраного відрізка будується із заданих значень функції та перших похідних на її кінцях. На відміну від глобального кубічного інтерполяційного сплайна класу C^2, сплайн представлений у роботі менше піддається впливу відхиленнь і дозволяє будувати функції майже без коливань.

Локальнй сплайн Catmull-Rom, який є особливим випадком сімейства кубічних сплайнів, був запропонований у роботі [24]. Перші похідні на кінцях відрізка побудовані як скінченні різниці, отримані зі значень інтерполюючої табличної функції. У [25] був побудований підклас сплайнів Катмулл-Рома, який містить параметри форми. Ці параметри використовуються для зміни форми кривої, незалежно від значень інтерпольованої функції. У роботі [26] проведено аналіз різних способів параметризації кубічної кривої Катмулла-Рома. Показано, що вибір параметрів є важливим у програмах. Використання параметрів форми у сплайнах Катмулл-Рома також є предметом роботи [27].

Таким чином, як випливає з аналізу існуючих методів інтерполяції, вирази для обчислення похідних отримують на основі комбінації значень інтерпольованих функцій. У запропонованому підході використовуються значення інтерпольованої функції та скінченних різниць, отримані на основі

значень у контрольних точках. Це дозволяє нам сказати, що побудований сплайн (у глобальній та локальній інтерпретаціях) управляється за допомогою деяких точок або відрізків полігональної лінії, що з'єднують ці точки.

6 ПОСТАНОВКА ПРОБЛЕМИ ТА ОПИС РЕАЛІЗОВАНОГО АЛГОРИТМУ

Нехай на інтервалі [a, b] визначена полігональна лінія. Абсциси вершин цієї лінії задані вузлами сітки:

$$\Delta_\tau: \ a = \tau_1 < \tau_2 < \ldots < \tau_N = b$$

Ординати вершин позначені F_i (контрольні точки в термінах кривих Безьє). Будується інтерполюючий сплайн, який задовольняє наступним умовам: кути нахилу зв'язків полігональної лінії визначають значення перших похідних сплайну у вузлах x_i сітки $\Delta_x: \ \tau_1 = x_1 < x_2 < \ldots < x_{N+1} = \tau_N$, такі, що $\tau_{i-1} < x_i < \tau_i, i = \overline{2, N}, f_i^{'} = \frac{(F_i - F_{i-1})}{h_i}$, де $h_i = \tau_i - \tau_{i-1}$, i $= \overline{2, N}$.

Значення інтерпольованої функції у вузлах сітки Δ_x визначаються наступним чином: $f_i = \frac{[F_{i-1}(h_i - \mu_i) + F_i\mu_i]}{h_i}$, де $\mu_i = \tau_i - x_i$. У цьому випадку значення μ_i відіграють роль параметрів форми кривої. Таким чином, задовольняються рівності $x_i - \tau_{i-1} = h_i - \mu_i$.

У [28] представлений алгоритм побудови та отримані умови існування та унікальності кубічної сплайн-кривої дефекту 2 в інтервалі [a, b], які задовольняють наступні умови:

$$S(x_i) = f_i$$

$$S(x_i) = f_i$$

$$S'(x_i) = f_i^{'}, \quad i = \overline{2, N}$$

Точки τ_i - це вузли сплайну, а точки x_i - вузли інтерполяції. Позначаючи ϕ_i, $i = \overline{1, N}$ невідомі значення сплайна в точках τ_i, система рівнянь для їх визначення записується у вигляді:

$$A_i\phi_{i-1} - (B_i^{(1)} + B_i^{(2)})\phi_i + C_i\phi_{i+1} = \Phi_i, \ i = \overline{2, N-1},$$

де

$$A_i = \frac{\mu_i^2}{(h_i - \mu_i)^2 h_i},\ B_i^{(1)} = \frac{2h_i + \mu_i}{\mu_i h_i},\ B_i^{(2)} = \frac{3h_{i+1} - \mu_{i+1}}{(h_{i+1} - \mu_{i+1})h_{i+1}},\ C_i = \frac{(h_{i+1} - \mu_{i+1})^2}{h_{i+1}\mu_{i+1}^2},$$

$$\Phi_i = f_i \frac{h_i}{(h_i - \mu_i)\mu_i} + f_i \frac{(h_i - 2\mu_i)h_i}{(h_i - \mu_i)^2 \mu_i} + \frac{F_i - F_{i-1}}{(h_i - \mu_i)} + f_{i+1} \frac{h_{i+1}}{(h_{i+1} - \mu_{i+1})\mu_{i+1}} -$$
$$- f_{i+1} \frac{(h_{i+1} - 2\mu_{i+1})h_{i+1}}{(h_{i+1} - \mu_{i+1})\mu_{i+1}^2} - \frac{F_{i+1} - F_i}{\mu_{i+1}}.$$

Для закриття системи рівнянь необхідно додати умови:

$$\phi_1 = F_1,\ \phi_N = F_N.$$

У цьому випадку сплайн записується так:

$$S(x) = \phi_{i-1} \frac{(x - x_i)(x - \tau_i)}{(\tau_{i-1} - x_i)(\tau_{i-1} - \tau_i)} + \phi_i \frac{(x - x_i)(x - \tau_{i-1})}{(\tau_i - x_i)(\tau_i - \tau_{i-1})} + f_i \frac{(x - \tau_i)(x - \tau_{i-1})}{(x_i - \tau_i)(x_i - \tau_{i-1})} +$$
$$+ Q(x - \tau_i)(x - \tau_{i-1})(x - x_i),$$

$$Q = -\phi_{i-1} \frac{1}{(h_i - \mu_i)^2 h_i} + \phi_i \frac{1}{h_i \mu_i^2} - f_i \frac{h_i - 2\mu_i}{(h_i - \mu_i)^2 \mu_i^2} - \frac{F_i - F_{i-1}}{h_i \mu_i (h_i - \mu_i)}$$

для $x \in [\tau_{i-1}, \tau_i], i = \overline{2, N}$.

Для спрощення подальших перетворень розглядаються однорідні сітки Δ_τ та Δ_x.

Візьмемо $h_i = h, \quad i = 2,\ldots,N-1,\ \tau_i = x_{i+1/2} = (x_{i+1} + x_i)/2, i = 1,..N-1.$

У цьому випадку сплайн набуває вигляду:

$$S(x) = -\phi_{i-1} \frac{4}{h^3}(x - x_i)^2 (x - x_{i+1/2}) - f_i \frac{4}{h^2}(x - x_{i-1/2})(x - x_{i+1/2}) +$$

$$+ \phi_i \frac{4}{h^3}(x - x_i)^2 (x - x_{i-1/2}) - f_i' \frac{4}{h^2}(x - x_i)(x - x_{i-1/2})(x - x_{i+1/2}).$$

Система рівнянь для визначення невідомих ϕ_i записується так:

$$\phi_1 = F_1,$$

$$\phi_{i-1} - 10\phi_i + \phi_{i+1} = 8F_i, \ i = \overline{2, N-1},$$

$$\phi_N = F_N.$$

Раніше розглядалася крива, складена з багаточленів, коефіцієнти яких визначались розв язанням системи лінійних алгебраїчних рівнянь. Злегка змінивши умову задачі, можна локально визначити коефіцієнти поліномів, уникаючи тим самим розв'язку системи. Якщо ми будуємо поліноми на відрізках $x \in [x_i, x_{i+1}], \ i = \overline{1, N-1}$, то немає необхідності знаходити ϕ_i з розв'язку системи рівнянь. У цьому випадку отримується локальний сплайн, який записується наступним чином:

$$S(x) = f_{i+1}(x - x_i)/h - f_i(x - x_{i+1})/h + (x - x_i)(x - x_{i+1})(ax + b)$$

$$a = (f_i' + f_{i+1}')/h^2 - 2(f_{i+1} - f_i)/h^3$$

$$a = (f_i' + f_{i+1}')/h^2 - 2(f_{i+1} - f_i)/h^3$$

для $x \in [x_i, x_{i+1}], i = \overline{1, N-1}$.

Локальний варіант кривої використовується в реалізованому алгоритммі для апроксимації вхідних даних ЕКГ. Після цього знаходяться квадрати похідної кривої $(s_i')^2$ у точках τ_i. Використовується формула

$$S'(x) = \frac{f_{i+1}}{h} - \frac{f_i}{h} + (x - x_i)(a_i(2x - x_{i+1}) + b_i) + (x - x_{i+1})(a_i x + b_i)$$

Відбувається пошук імпульсів $(s_i')^2$ які перевищують заданий поріг t, та беруться точки m_j з максимальним значенням $(s_i')^2$ у відповідних імпульсах. Для кожної точки m_j розглядається окіл з заданим радіусом r в якому шукається точка q_j з максимальним значенням аппроксимованої функції. Ці точки й є шуканими вершинами R-зубця QRS-комплексів. Отримавши вершину R-зубця знаходяться вершини Q та S зубців.

7 РЕЗУЛЬТАТИ

На рис.2 - 31 зображені вхідні дані ЕКГ на яких був протестований програмний засіб, отримана в результаті побудови інтерполяційного сплайна апроксимація вхідних даних, квадрат похідної апроксимації значення якого більші за вказаний поріг t вказують на околиці R-зубця. Значення параметрів t та r вказані у таблиці 1.

	1	**2**	**3**	**4**	**5**
t	**10**	**100**	**50**	**50**	**10**
r	**0.025**	**0.025**	**0.015**	**0.015**	**0.005**

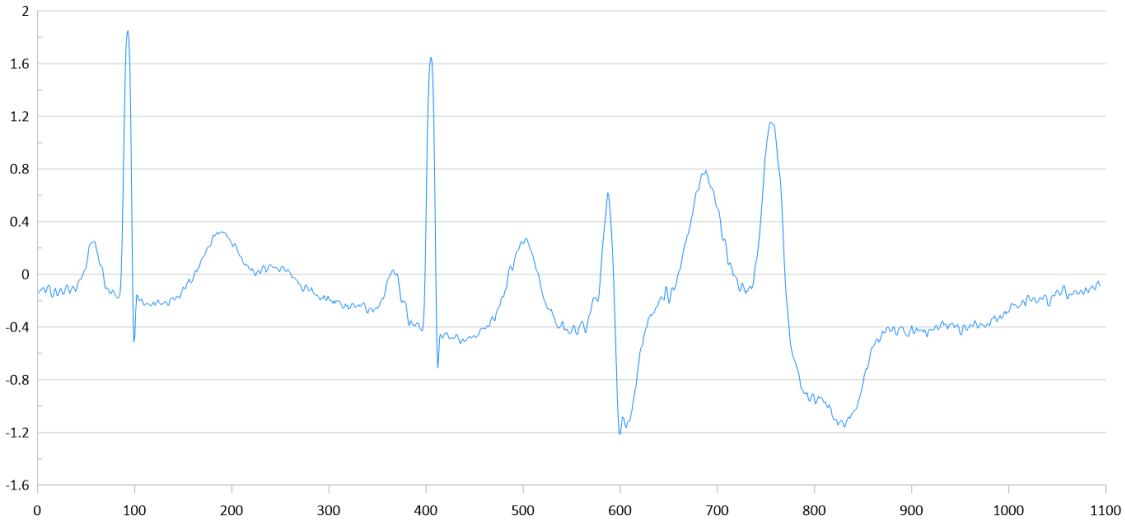

Рисунок 2 – E.1, Вхідні дані

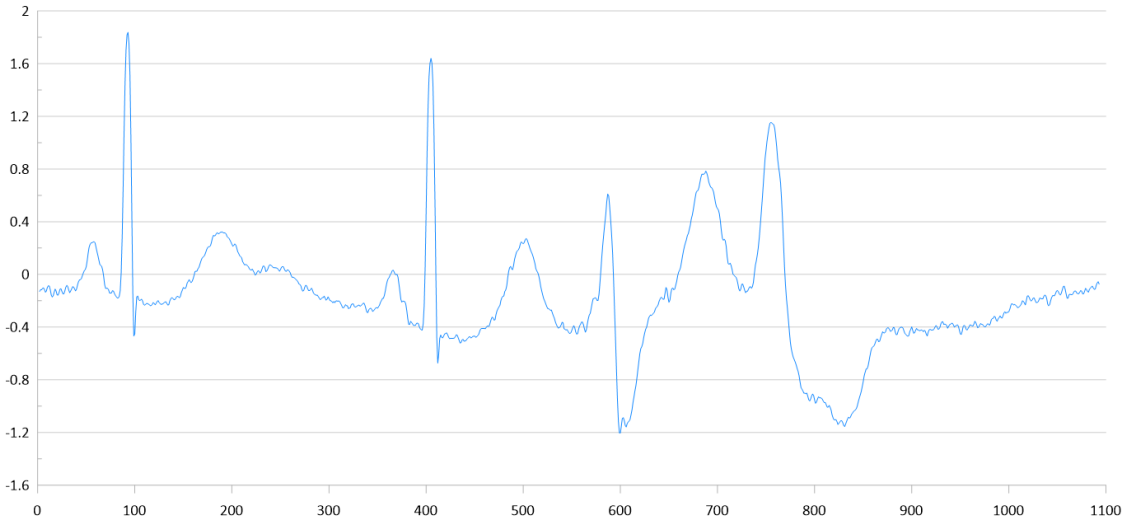

Рисунок 3 – E.1, Апроксимація

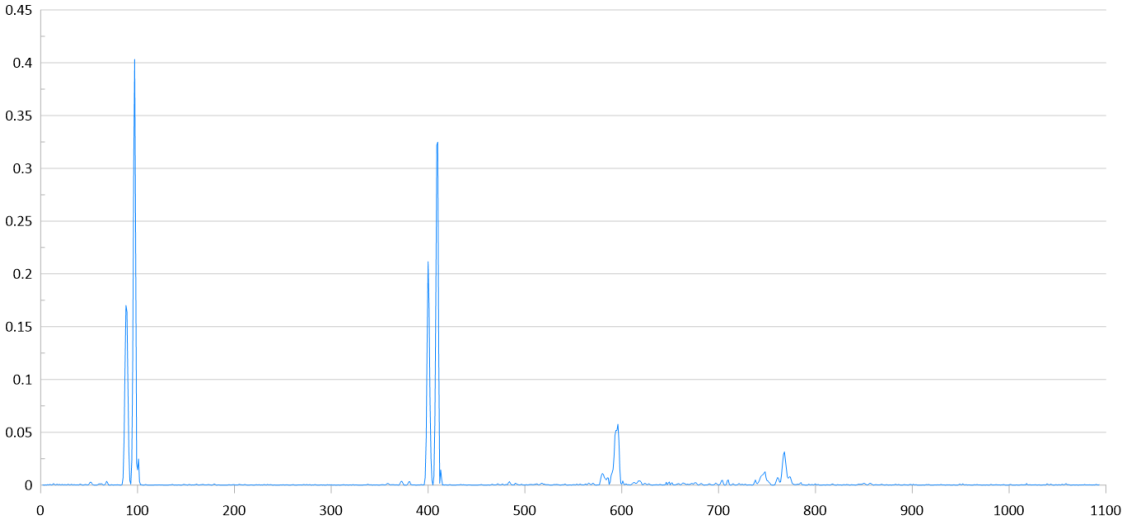

Рисунок 4 – E.1, Квадрат похідної апроксимації

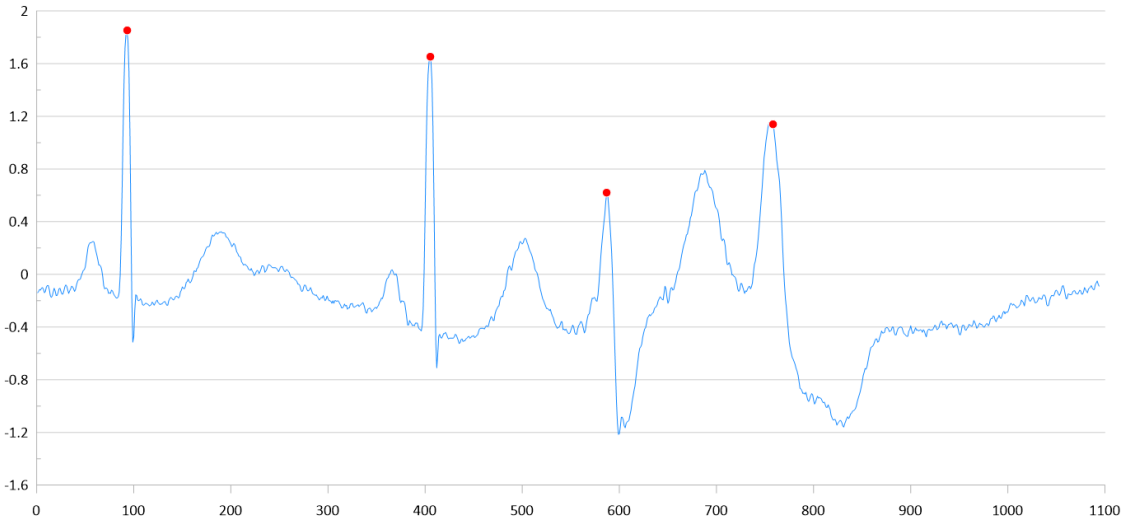

Рисунок 2 – E.1, Вхідні дані з поміченими точками виявлених QRS-комплексів

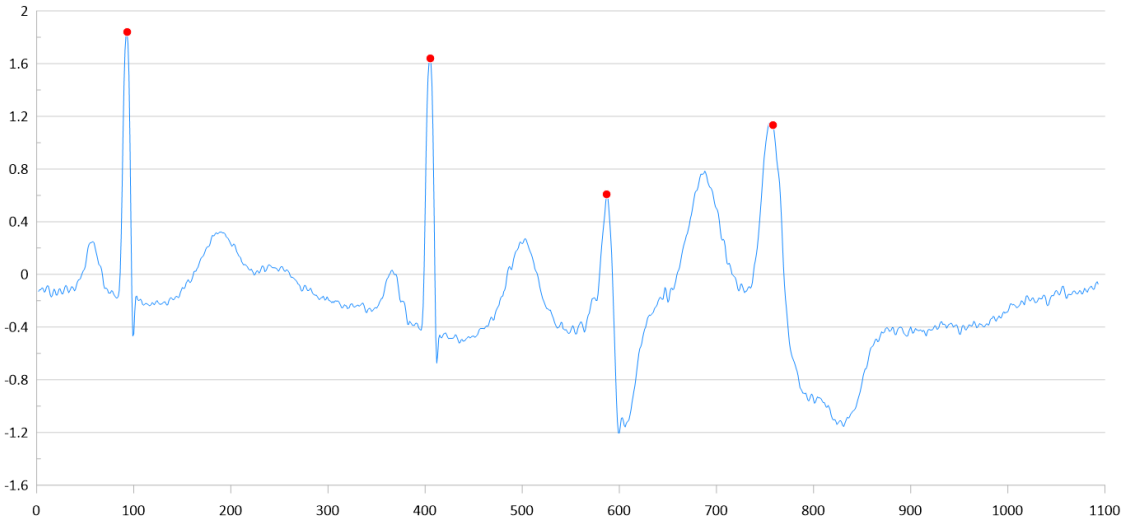

Рисунок 6 – E.1, Апроксимація з поміченими точками виявлених QRS-комплексів

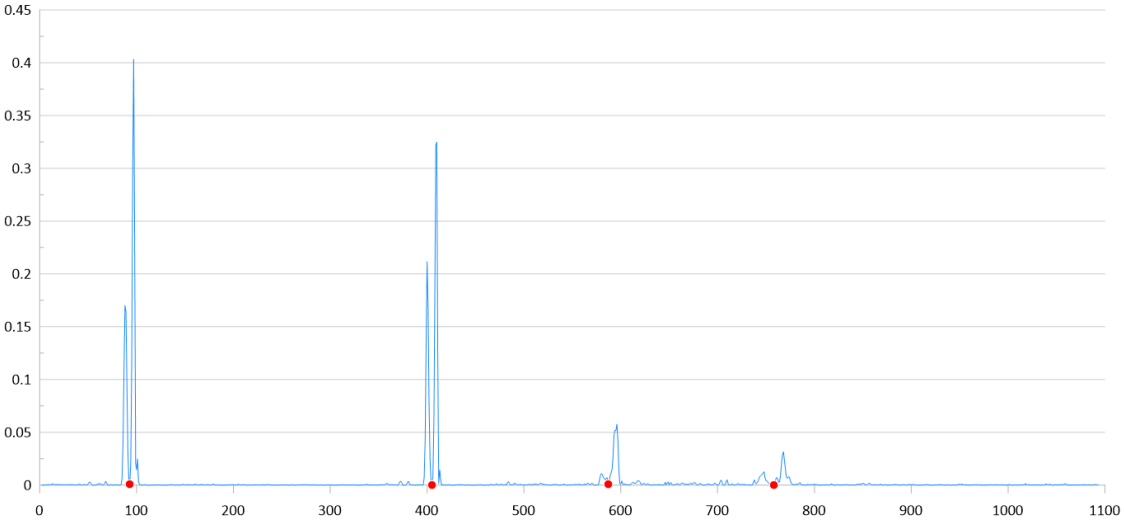

Рисунок 7 – E.1, Квадрат похідної апроксимації з поміченими точками виявлених QRS-комплексів

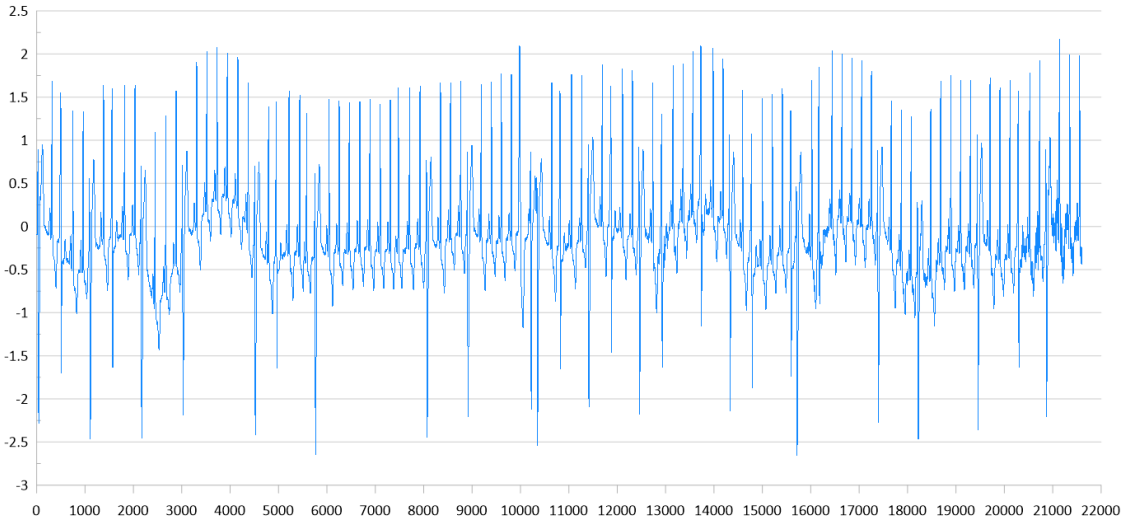

Рисунок 8 – E.2, Вхідні дані

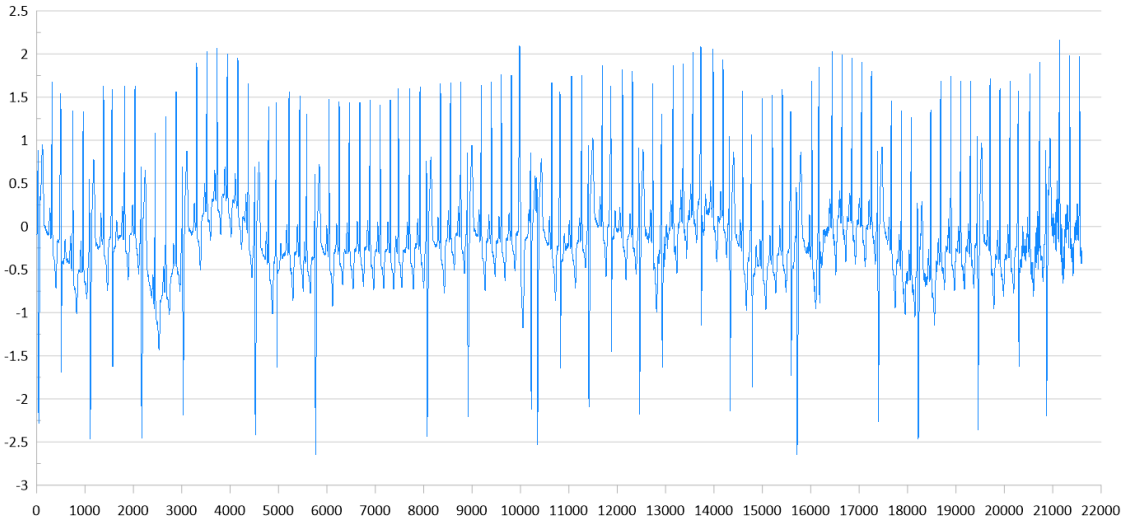

Рисунок 9 – E.2, Апроксимація

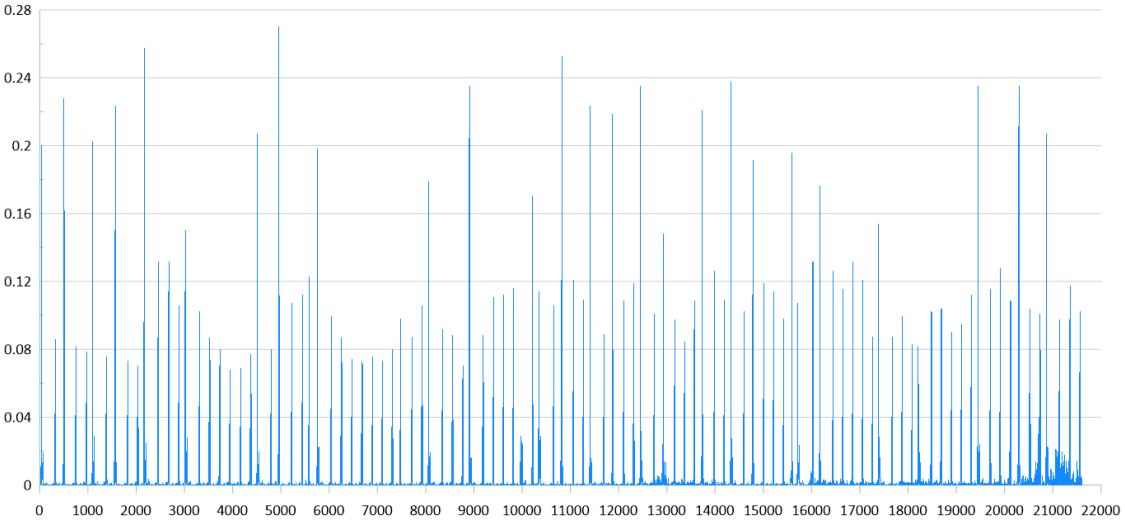

Рисунок 10 – E.2, Квадрат похідної апроксимації

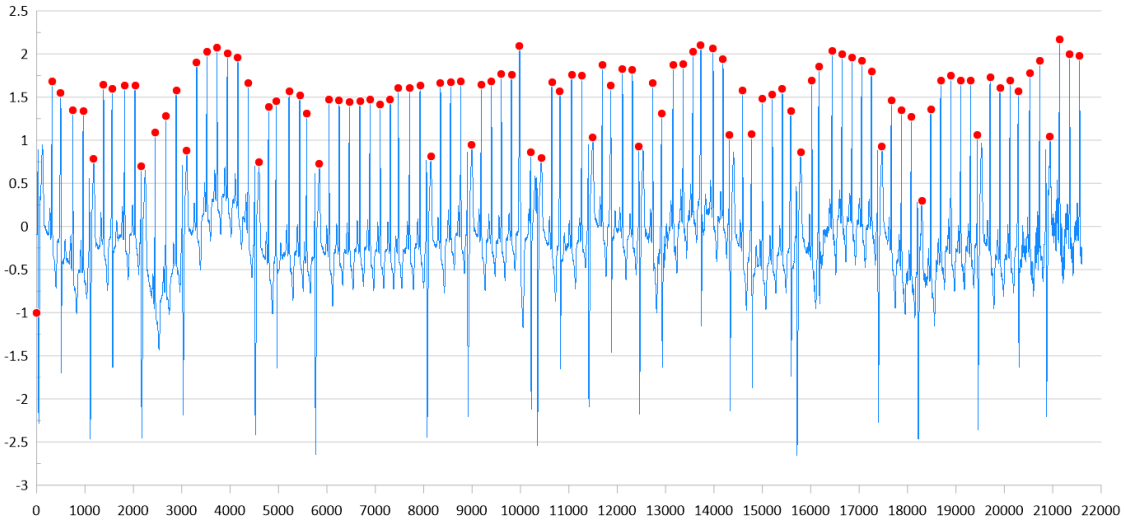

Рисунок 11 – E.2, Вхідні дані з поміченими точками виявлених QRS-комплексів

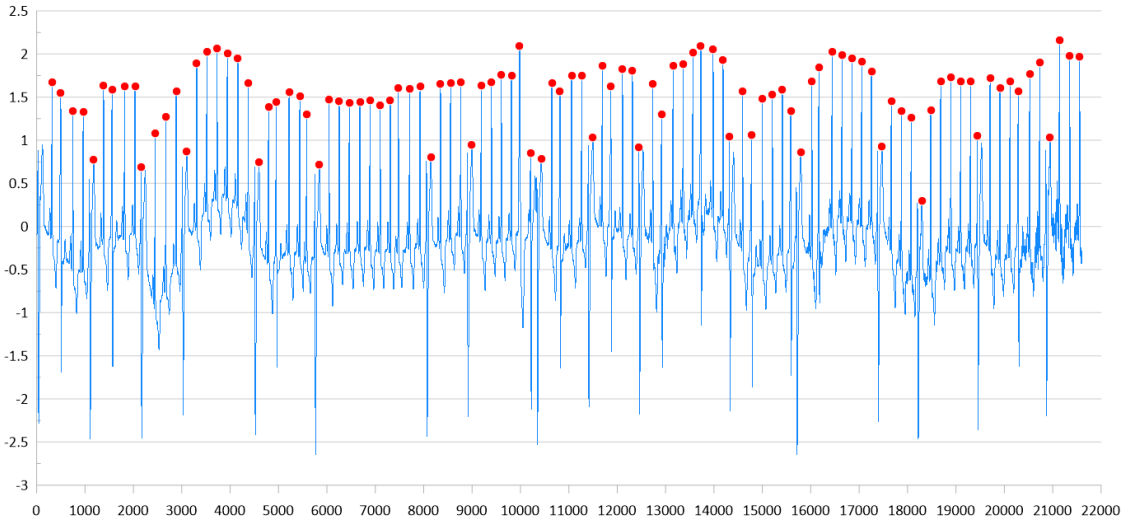

Рисунок 12 – E.2, Апроксимація з поміченими точками виявлених QRS-комплексів

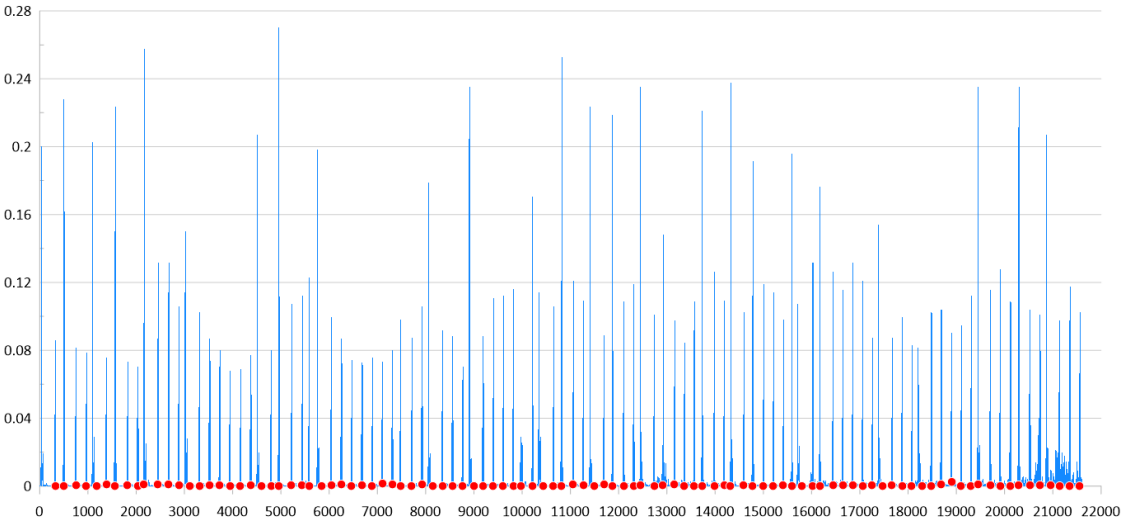

Рисунок 13 – E.2, Квадрат похідної апроксимації з поміченими точками виявлених QRS-комплексів

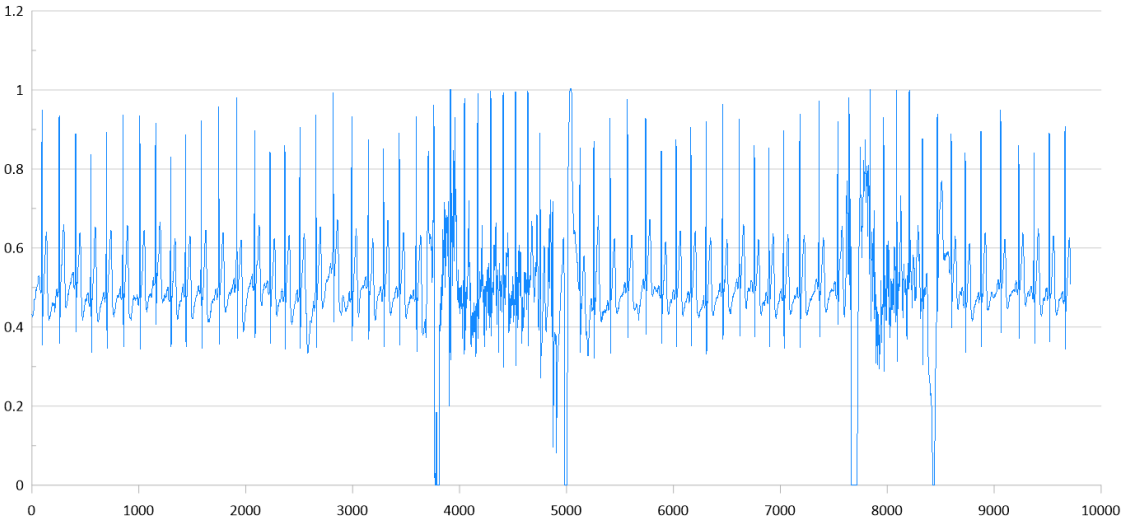

Рисунок 14 – E.3, Вхідні дані

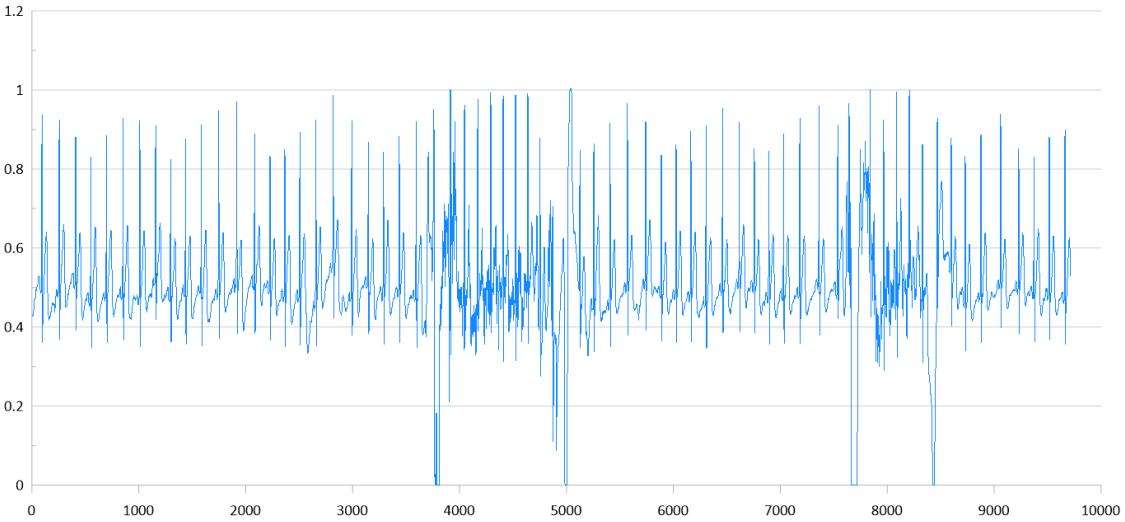

Рисунок 15 – E.3, Апроксимація

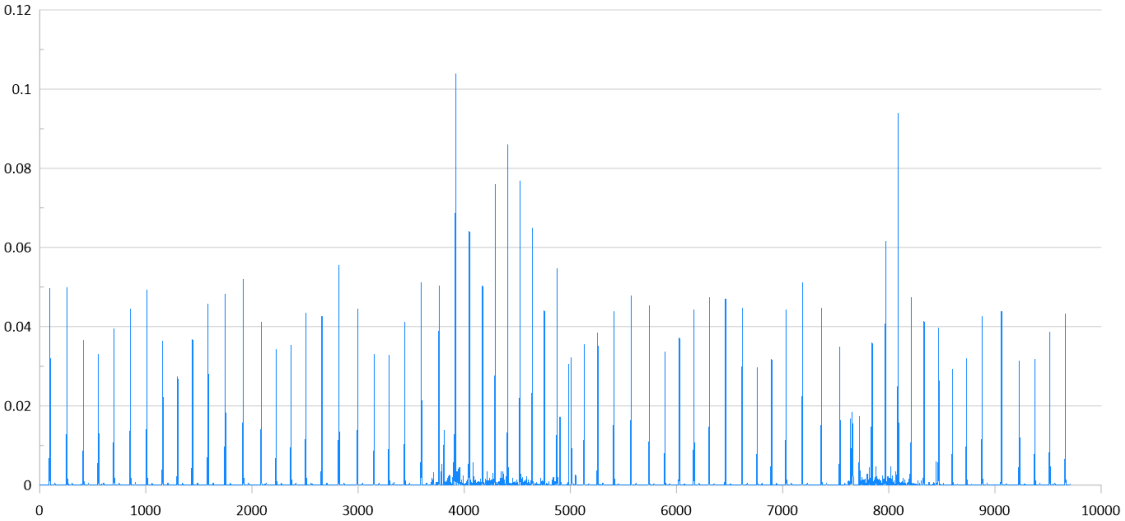

Рисунок 16 – E.3, Квадрат похідної апроксимації

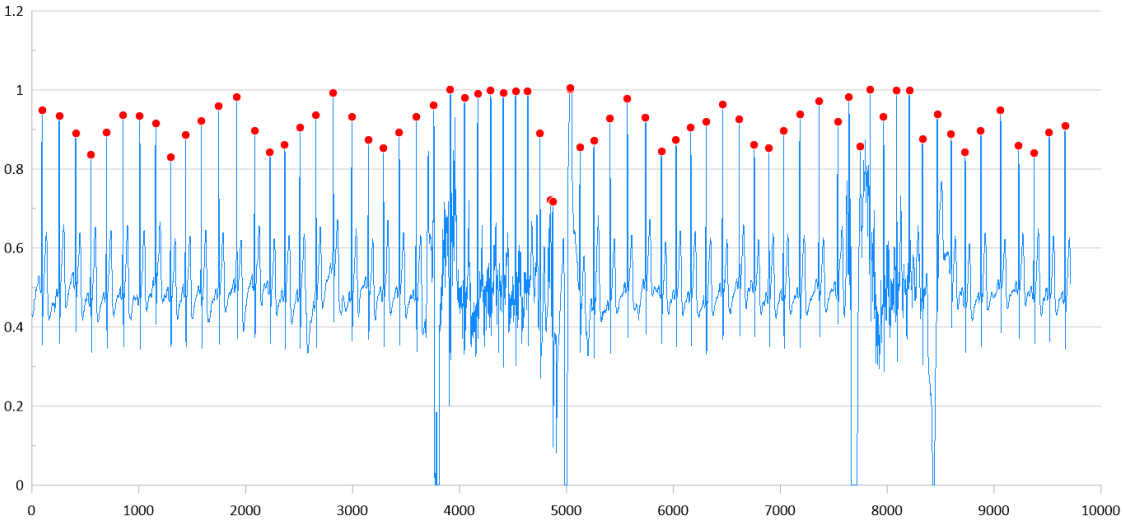

Рисунок 17 – Е.3, Вхідні дані з поміченими точками виявлених QRS-комплексів

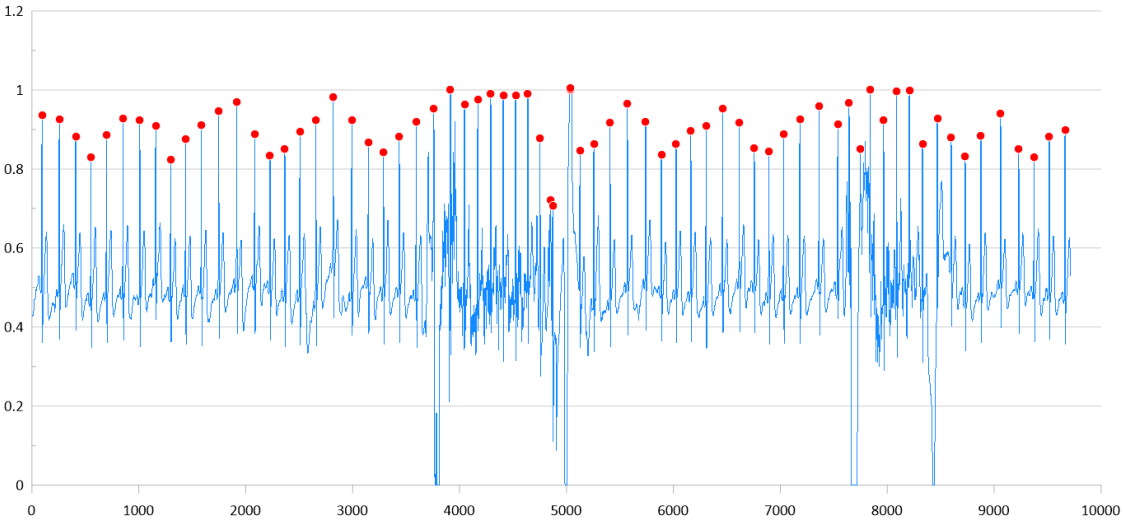

Рисунок 18 – Е.3, Апроксимація з поміченими точками виявлених QRS-комплексів

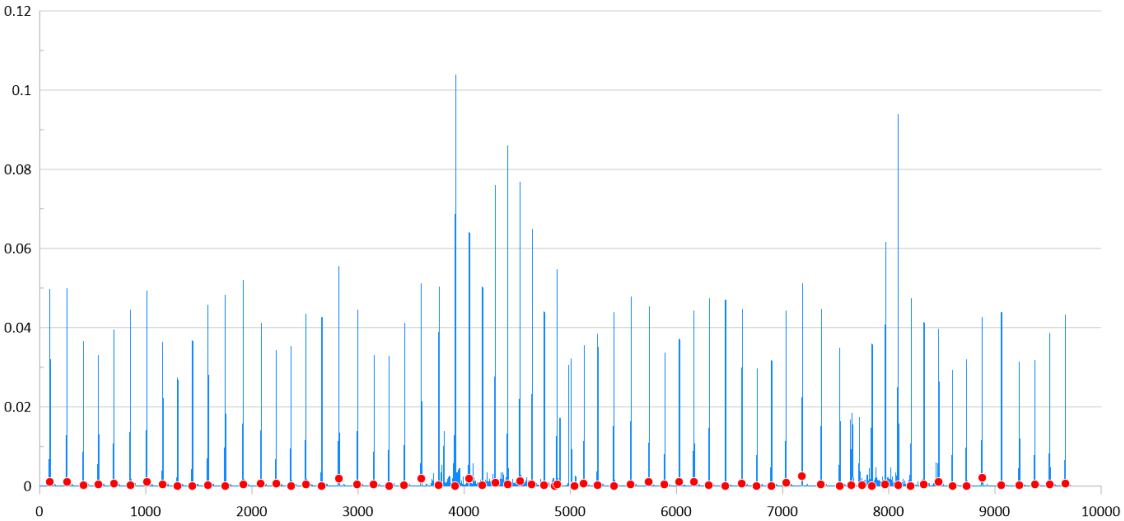

Рисунок 19 – Е.3, Квадрат похідної апроксимації з поміченими точками виявлених QRS-комплексів

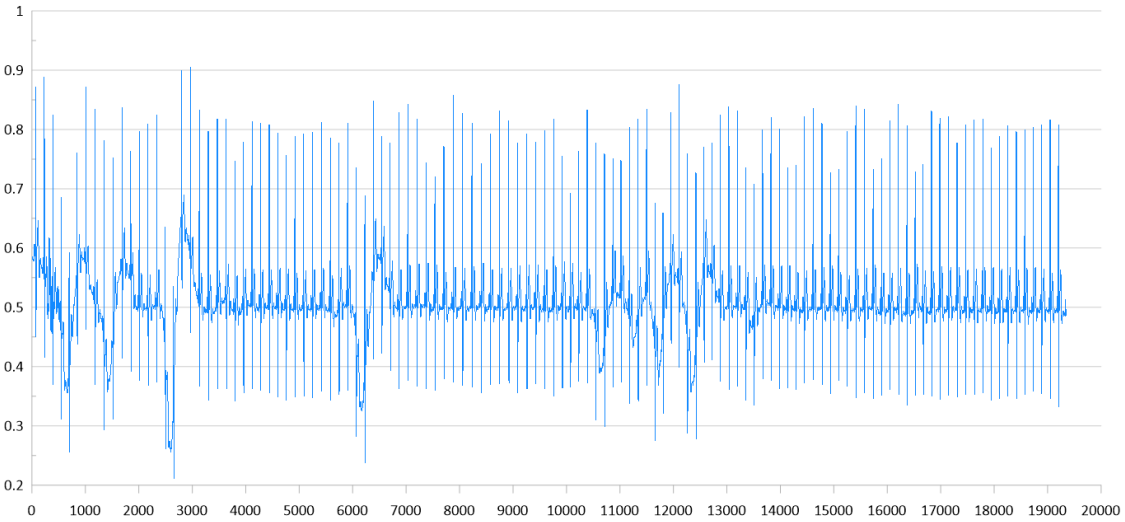

Рисунок 20 – E.4, Вхідні дані

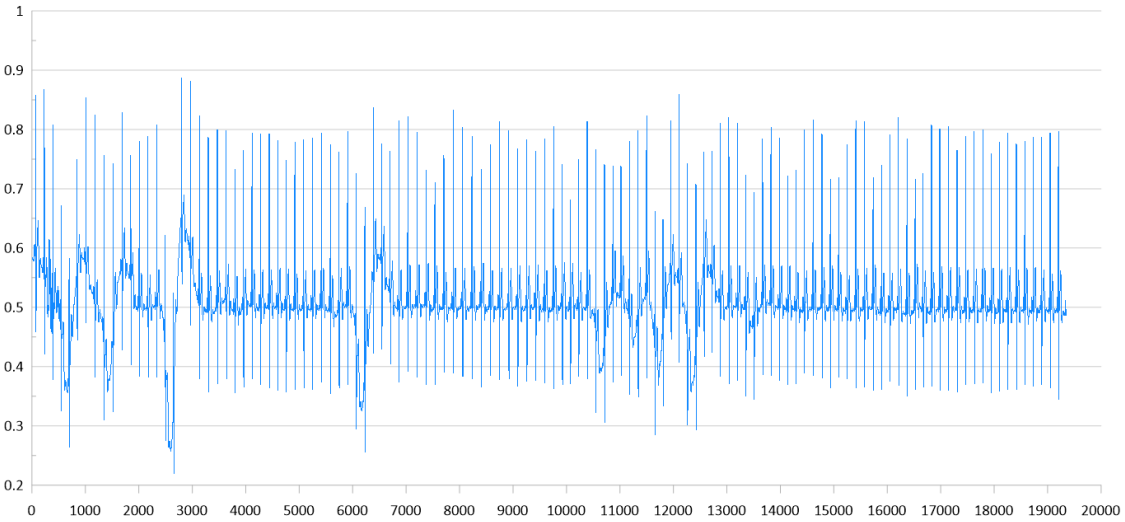

Рисунок 21 – E.4, Апроксимація

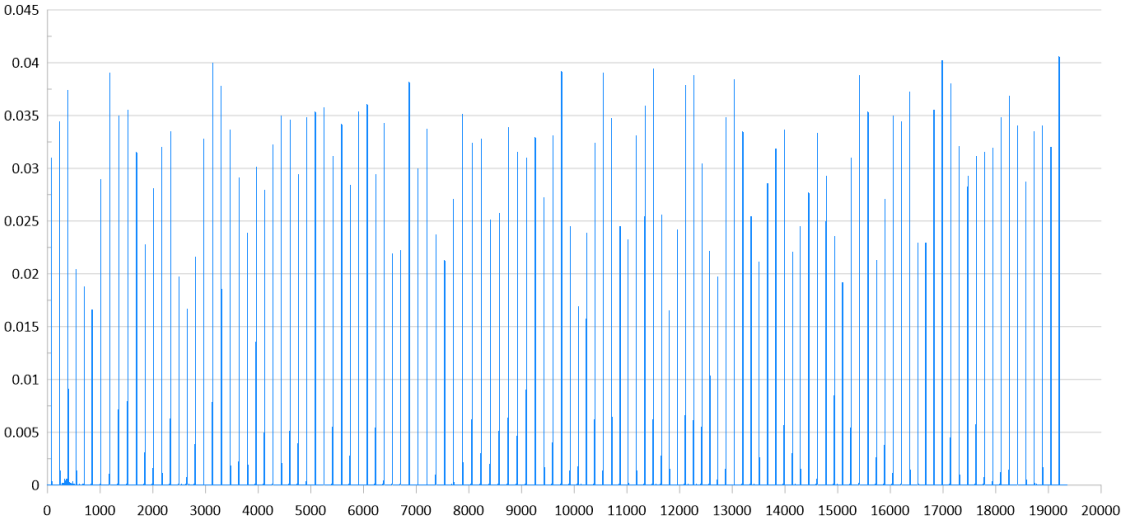

Рисунок 22 – E.4, Квадрат похідної апроксимації

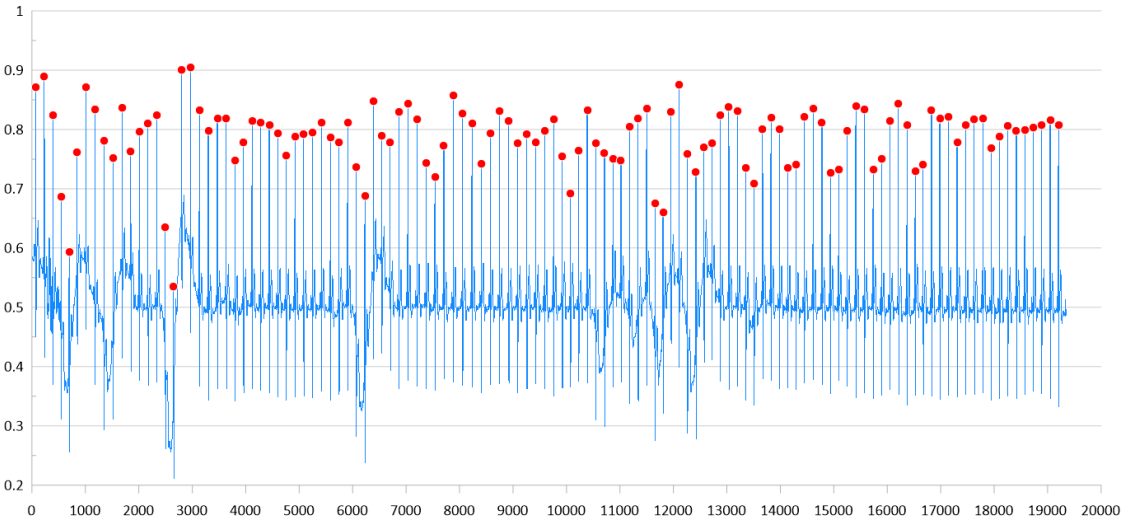

Рисунок 23 – E.4, Вхідні дані з поміченими точками виявлених QRS-комплексів

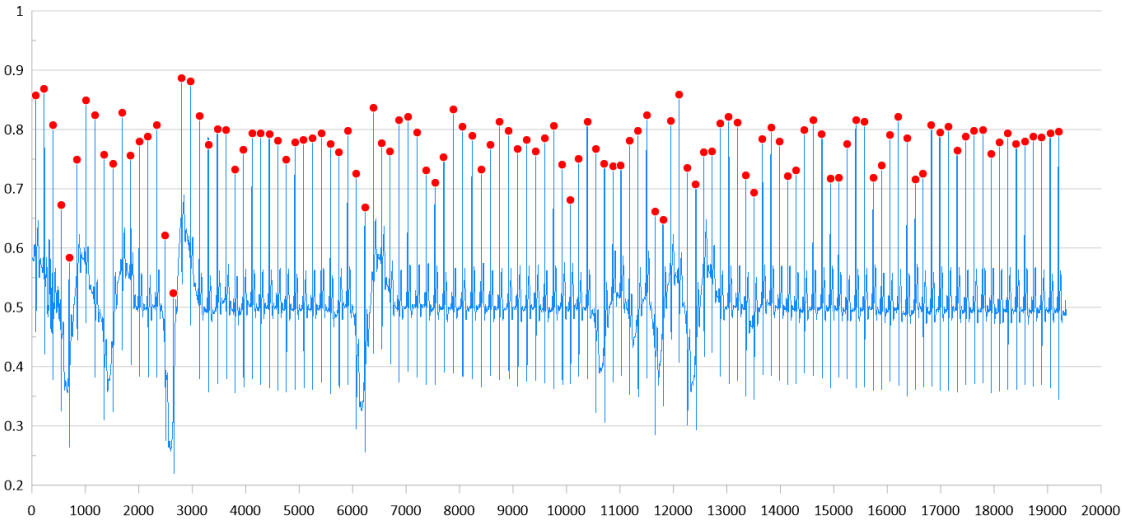

Рисунок 24 – E.4, Апроксимація з поміченими точками виявлених QRS-комплексів

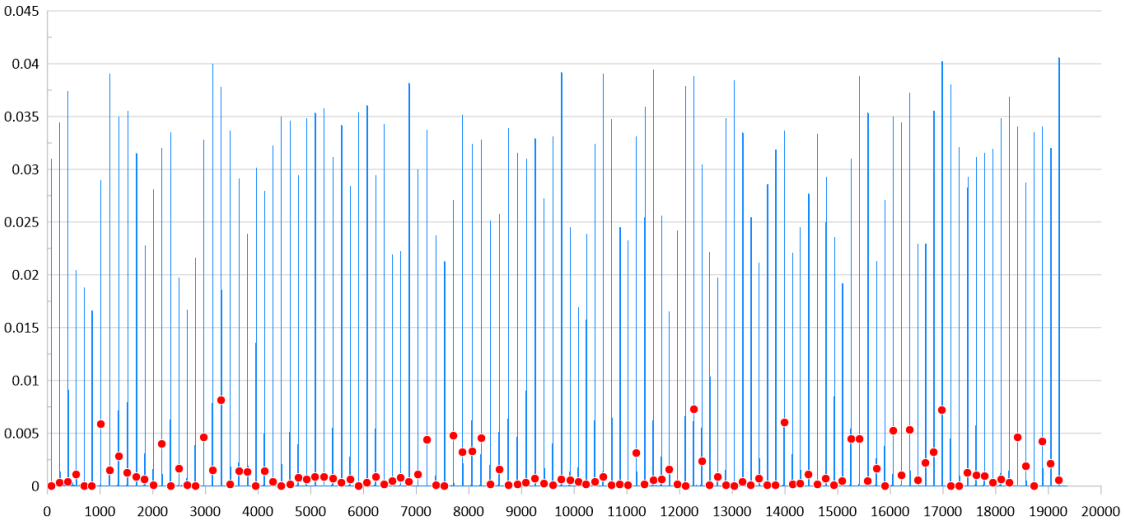

Рисунок 25 – E.4, Квадрат похідної апроксимації з поміченими точками виявлених QRS-комплексів

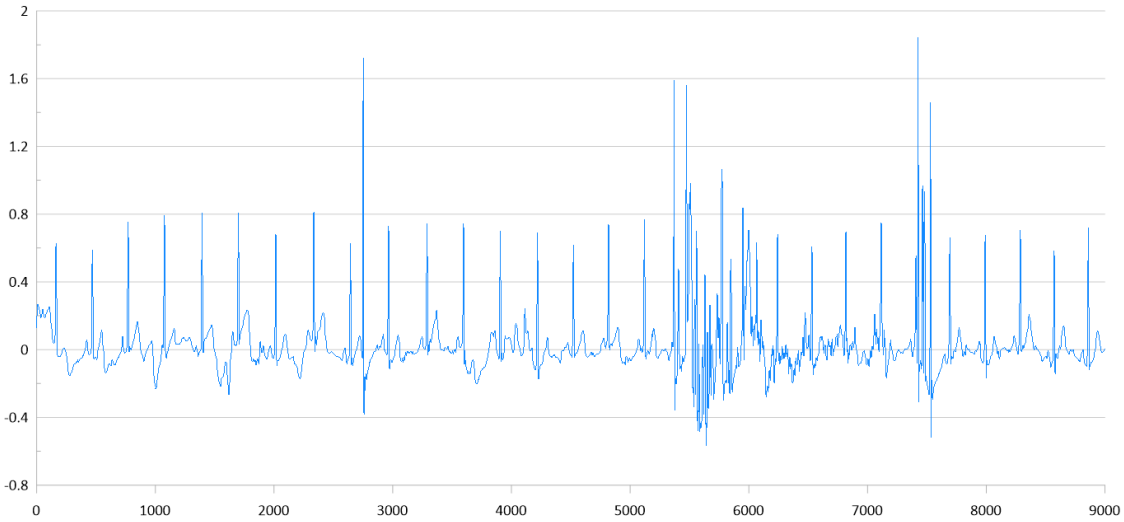

Рисунок 26 – E.5, Вхідні дані

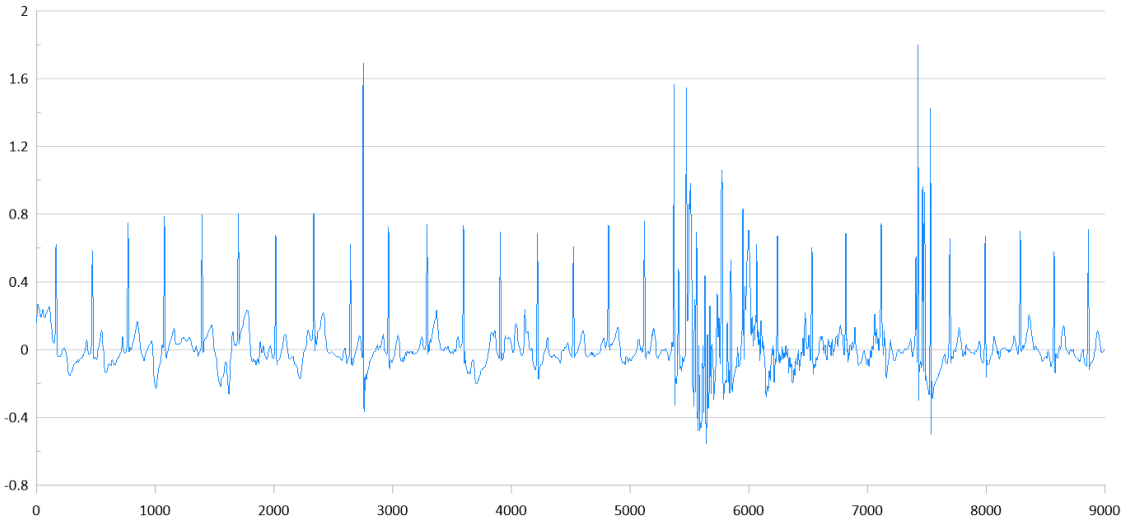

Рисунок 27 – E.5, Апроксимація

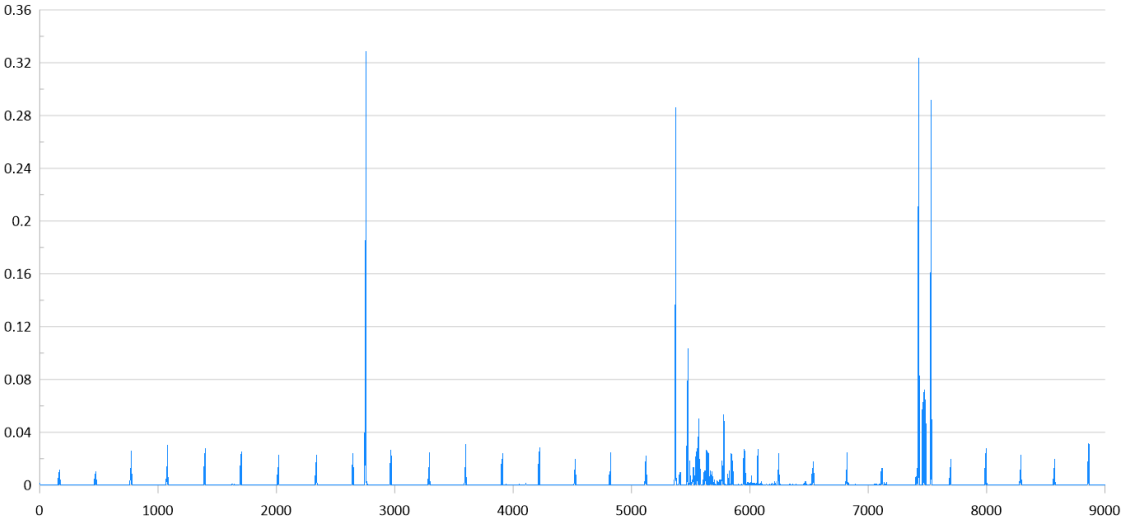

Рисунок 28 – E.5, Квадрат похідної апроксимації

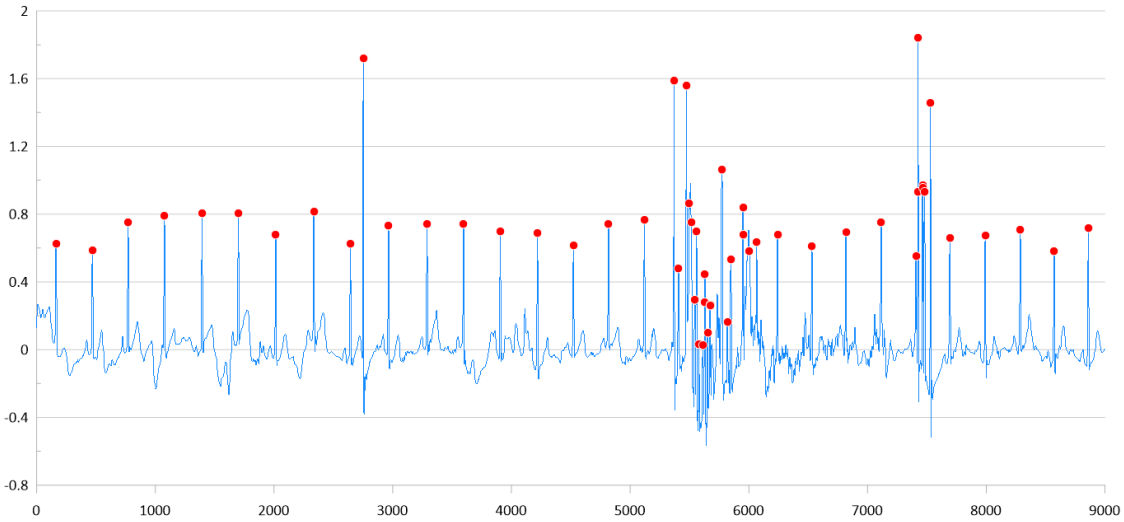

Рисунок 29 – E.5, Вхідні дані з поміченими точками виявлених QRS-комплексів

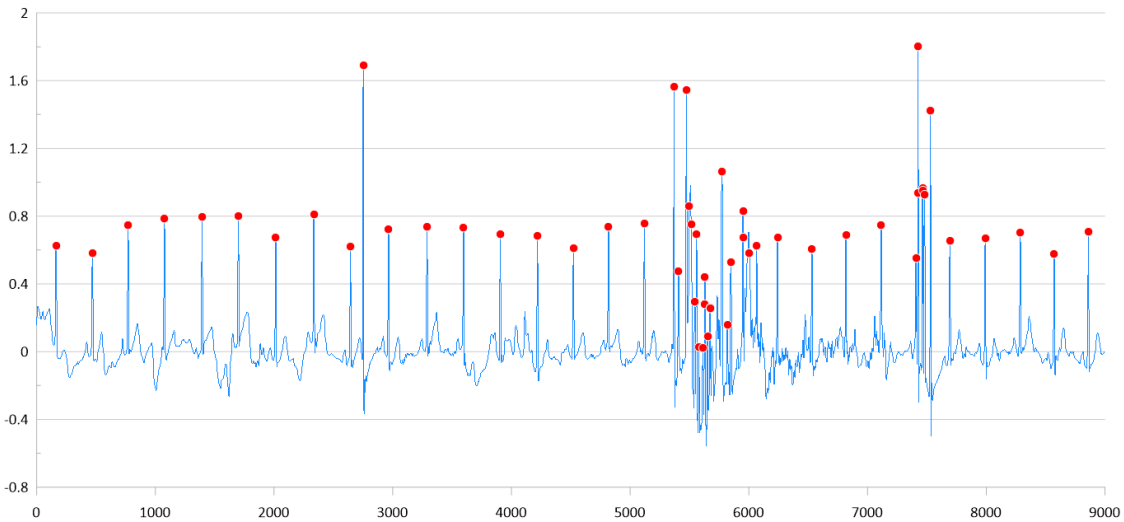

Рисунок 30 – E.5, Апроксимація з поміченими точками виявлених QRS-комплексів

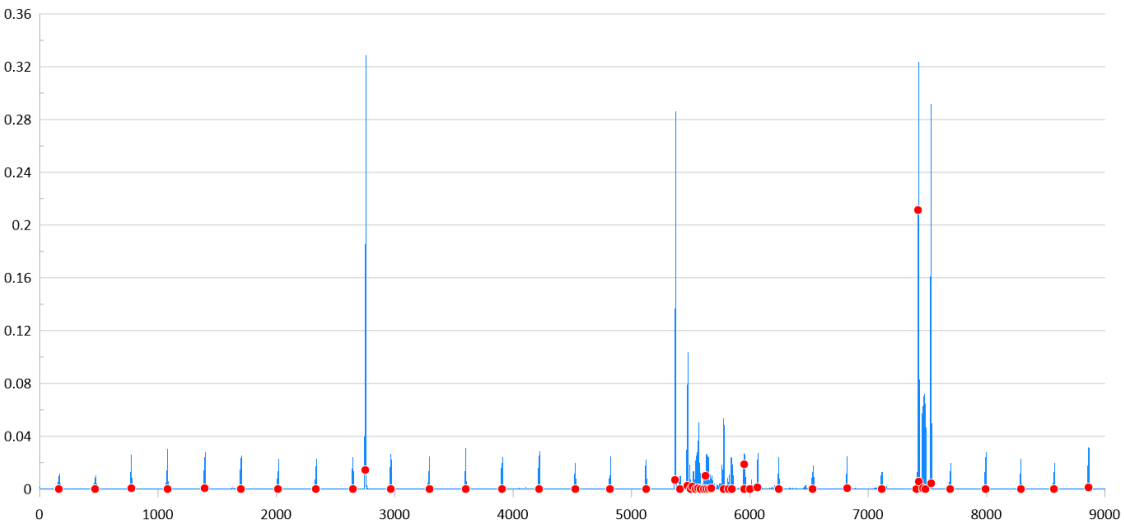

Рисунок 31 – E.5, Квадрат похідної апроксимації з поміченими точками виявлених QRS-комплексів

8 ВИСНОВКИ

Побудована реалізація алгоритму виявлення QRS-комплексів використовуючого локальну кусково-заданну поліноміальну криву змінного порядку та проведене тестування. Програмна реалізація може використовуватись у мікроконтроллерах автоматизованих систем моніторингу пацієнтів задля отримання частоти сердцевого ритму з метою його подальшого аналізу.

9 ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. L. S¨ornmo and P. Laguna, Bioelectrical Signal Processing in Cardiac and Neurological Applications. Elsevier Academic Press, June 2005.

2. Moody GB, Mark RG, Zoccola A, Mantero S. Derivation of Respiratory Signals from Multi-lead ECGs Computers in Cardiology 1985; 12:113-116.

3. Kohler BU, Henning C, Orglmeister R. The principles of software QRS detection IEEE Engineering in Medicine and Biology 2002;1:42-57.

4. Lu, Xuanyu & Pan, Maolin & Yu, Yang. (2018). QRS Detection Based on Improved Adaptive Threshold. Journal of Healthcare Engineering. 2018. 1-8. 10.1155/2018/5694595.

5. O. Pahlm and L. S¨ornmo, “Software qrs detection in ambulatory monitoring — a review,” Medical and Biological Engineering and Computing, vol. 22, no. 4, pp. 289–297, 1984.

6. G. Friesen, T. Jannett, M. Jadallah, S. Yates, S. Quint, and H. Nagle, “A comparison of the noise sensitivity of nine qrs detection algorithms,” Biomedical Engineering, IEEE Transactions on, vol. 37, no. 1, pp. 85–98, 1990.

7. N. Arzeno, Z.-D. Deng, and C.-S. Poon, “Analysis of first-derivative based qrs detection algorithms,” Biomedical Engineering, IEEE Transactions on, vol. 55, no. 2, pp. 478–484, 2008.

8. V. Afonso, W. J. Tompkins, T. Nguyen, and S. Luo, “Ecg beat detection using filter banks,” Biomedical Engineering, IEEE Transactions on, vol. 46, no. 2, pp. 192–202, 1999.

9. M. Okada, “A digital filter for the qrs complex detection,” Biomedical Engineering, IEEE Transactions on, vol. BME-26, no. 12, pp. 700–703, 1979.

10. J. Pan and W. J. Tompkins, “A real-time qrs detection algorithm,” Biomedical Engineering, IEEE Transactions on, vol. BME-32, 1985.

11. P. S. Hamilton and W. J. Tompkins, “Quantitative investigation of qrs detection rules using the mit/bih arrhythmia database,” Biomedical Engineering, IEEE Transactions on, vol. BME-33, no. 12, pp. 1157–1165, 1986.

12. J. Martinez, R. Almeida, S. Olmos, A. Rocha, and P. Laguna, “A wavelet-based ecg delineator: evaluation on standard databases,” Biomedical Engineering, IEEE Transactions on, vol. 51, no. 4, pp. 570–581, 2004.

13. C. Li, C. Zheng, and C. Tai, “Detection of ecg characteristic points using wavelet transforms,” Biomedical Engineering, IEEE Transactions on, vol. 42, no. 1, pp. 21–28, 1995.

14. B. Abibullaev and H. Seo, “A new qrs detection method using wavelets and artificial neural networks,” Journal of Medical Systems, vol. 35, no. 4, pp. 683–691, 2011.

15. Q. Xue, Y. Hu, and W. J. Tompkins, “Neural-network-based adaptive matched filtering for qrs detection,” Biomedical Engineering, IEEE Transactions on, vol. 39, no. 4, pp. 317–329, 1992.

16. D. Coast, R. Stern, G. Cano, and S. Briller, “An approach to cardiac arrhythmia analysis using hidden markov models,” Biomedical Engineering, IEEE Transactions on, vol. 37, no. 9, pp. 826–836, 1990.

17. R. Poli, S. Cagnoni, and G. Valli, “Genetic design of optimum linear and nonlinear qrs detectors,” Biomedical Engineering, IEEE Transactions on, vol. 42, no. 11, pp. 1137–1141, 1995.

18. Oppenheim AV, Willsky AS. Signals and Systems. Prentice-Hall, Englewood Cliffs, NJ, 1983.

19. Castiglioni, Paolo & Piccini, Luca & Di Rienzo, Marco. (2003). Interpolation technique for extracting features from ECG signals sampled at low sampling rates. Computers in Cardiology. 30. 481 - 484. 10.1109/CIC.2003.1291197.

20. de Boor, C.: A Practical Guide to Splines (Revised Edition), Springer-Verlag (2001).

21. Zavyalov, Yu. S., Kvasov, B.I., Miroshnichenko, V.L.: Methods of spline functions. Nauka, Moscow (1980) (In Russian).

22. Ahlberg, J. H., Nilson, E. N., Walsh, J. L.: The theory of splines and their applications, New York, Academic Press (1967).

23. Akima, H.: A new method of interpolation and smooth curve fitting based on local procedures. Boulder Journal of the ACM (JACM) 17(4): 589-602 (1970).

24. Catmull, E., Rom, R.: A Class of local interpolation splines, Computer Aided Geometric Design. Barnhill R.E. and R.F. Riesenfled (eds.), Academic Press, 317-326 (1974).

25. Dtrose, T. D., Barsky, B.A.: Geometric continuity, shape parameters, and geometric constructions for Catmull-Rom splines. ACM Transactions on Graphics 7(1): 1-41 (1988).

26. Yuksel, C., Schaefer, S., Keyser, J.: Parameterization and applications of Catmull-Rom curves. Journal Computer-Aided Design 43(7): 747-755 (2011).

27. Li, J., Chen, S.: The Cubic α-Catmull-Rom spline. Mathematical and Computational Applications 21(3): 21-33 (2016).

28. Stelia, O., Potapenko, L., Sirenko, I.: Application of piecewise-cubic functions for constructing a Bezier type curve of smoothness. Eastern European Journal of Enterprise Technologies 2(4-92): 46-52 (2018).

29. Stelia O., Krak I., Potapenko L. (2020) Controlled Spline of Third Degree: Approximation Properties and Practical Application. In: Lytvynenko V., Babichev S., Wójcik W., Vynokurova O., Vyshemyrskaya S., Radetskaya S. (eds) Lecture Notes in Computational Intelligence and Decision Making. ISDMCI 2019. Advances in Intelligent Systems and Computing, vol 1020. Springer, Cham. https://doi.org/10.1007/978-3-030-26474-1_16,

10 ДОДАТОК А

10.1 Програмний код

```
#include <algorithm>
#include <cmath>
#include <fstream>
#include <iostream>
#include <string>
#include <tuple>
#include <utility>
#include <vector>

std::vector<std::pair<long double, long double>> read_data(
    std::string file_name) {
  std::vector<std::pair<long double, long double>> data;
  std::pair<long double, long double> point;
  std::ifstream file_stream;
  std::string line;
  std::string delim = " ";
  long double x;
  long double y;
  size_t start_pos;
  size_t fin_pos;

  file_stream.open(file_name);
  while (std::getline(file_stream, line)) {
    start_pos = line.find_first_not_of(delim);
    fin_pos = line.find_first_of(delim, start_pos);
    x = std::stold(line.substr(start_pos, fin_pos - start_pos));
```

```
    start_pos = line.find_first_not_of(delim, fin_pos);
    fin_pos = line.find_first_of(delim, start_pos);
    if (fin_pos == std::string::npos) {
      fin_pos = line.size();
    }
    y = std::stold(line.substr(start_pos, fin_pos - start_pos));
    point = std::pair<long double, long double>(x, y);
    data.push_back(point);
  }
  file_stream.close();

  return data;
}

std::vector<long double> read_beta(std::string file_name) {
  std::vector<long double> beta;
  std::ifstream file_stream;
  std::string line;
  std::string delim;
  long double beta_val;
  size_t start_pos;
  size_t fin_pos;
  delim = " ";

  file_stream.open(file_name);
  while (std::getline(file_stream, line)) {
    start_pos = line.find_first_not_of(delim);
    fin_pos = line.find_first_of(delim, start_pos);
    if (fin_pos == std::string::npos) {
      fin_pos = line.size();
```

```
    }
    beta_val = std::stold(line.substr(start_pos, fin_pos - start_pos));
    beta.push_back(beta_val);
  }
  file_stream.close();

  return beta;
}

long double read_threshold(std::string file_name) {
  std::ifstream file_stream;
  std::string line;
  std::string delim;
  long double threshold;
  size_t start_pos;
  size_t fin_pos;
  delim = " ";

  file_stream.open(file_name);
  std::getline(file_stream, line);
  start_pos = line.find_first_not_of(delim);
  fin_pos = line.find_first_of(delim, start_pos);
  if (fin_pos == std::string::npos) {
    fin_pos = line.size();
  }
  threshold = std::stold(line.substr(start_pos, fin_pos - start_pos));
  file_stream.close();

  return threshold;
}
```

```
long double read_radius(std::string file_name) {
  std::ifstream file_stream;
  std::string line;
  std::string delim;
  long double radius;
  size_t start_pos;
  size_t fin_pos;
  delim = " ";

  file_stream.open(file_name);
  std::getline(file_stream, line);
  start_pos = line.find_first_not_of(delim);
  fin_pos = line.find_first_of(delim, start_pos);
  if (fin_pos == std::string::npos) {
    fin_pos = line.size();
  }
  radius = std::stoi(line.substr(start_pos, fin_pos - start_pos));
  file_stream.close();

  return radius;
}

std::tuple<std::vector<std::pair<long double, long double>>,
           std::vector<long double>, std::vector<long double>>
get_coefs(std::vector<std::pair<long double, long double>> data,
          std::vector<long double> beta) {
  std::tuple<std::vector<std::pair<long double, long double>>,
             std::vector<long double>, std::vector<long double>>
      coefs;
```

```
std::vector<std::pair<long double, long double>> grid;
std::pair<long double, long double> point;
std::vector<long double> a;
std::vector<long double> b;
long double a_val;
long double b_val;
std::string::size_type i;

for (i = 0; i < data.size() - 1; ++i) {
  point = std::pair<long double, long double>(
      data.at(i + 1).first * beta.at(i) + data.at(i).first * (1 - beta.at(i)),
      data.at(i + 1).second * beta.at(i) +
          data.at(i).second * (1 - beta.at(i)));
  grid.push_back(point);
  if (i >= 1) {
    a_val = (((data.at(i).second - data.at(i - 1).second) /
              (data.at(i).first - data.at(i - 1).first)) +
             ((data.at(i + 1).second - data.at(i).second) /
              (data.at(i + 1).first - data.at(i).first))) /
                std::pow((data.at(i).first - data.at(i - 1).first) *
                             (1 - beta.at(i - 1)) +
                         (data.at(i + 1).first - data.at(i).first) *
                             beta.at(i),
                         2) -
            2 * (grid.at(i).second - grid.at(i - 1).second) /
                std::pow((data.at(i).first - data.at(i - 1).first) *
                             (1 - beta.at(i - 1)) +
                         (data.at(i + 1).first - data.at(i).first) *
                             beta.at(i),
                         3);
```

```
    a.push_back(a_val);
    b_val =
        (((data.at(i + 1).second - data.at(i).second) /
          (data.at(i + 1).first - data.at(i).first)) -
         (((data.at(i).second - data.at(i - 1).second) /
           (data.at(i).first - data.at(i - 1).first)))) /
          (2 * ((data.at(i).first - data.at(i - 1).first) *
                  (1 - beta.at(i - 1)) +
              (data.at(i + 1).first - data.at(i).first) * beta.at(i))) -
        (((data.at(i).second - data.at(i - 1).second) /
          (data.at(i).first - data.at(i - 1).first)) +
         ((data.at(i + 1).second - data.at(i).second) /
          (data.at(i + 1).first - data.at(i).first))) *
          (grid.at(i - 1).first + grid.at(i).first) /
          (2 * std::pow((data.at(i).first - data.at(i - 1).first) *
                          (1 - beta.at(i - 1)) +
                      (data.at(i + 1).first - data.at(i).first) *
                          beta.at(i),
                    2)) +
        (grid.at(i).second - grid.at(i - 1).second) *
          (grid.at(i - 1).first + grid.at(i).first) /
          std::pow(
              (data.at(i).first - data.at(i - 1).first) *
                  (1 - beta.at(i - 1)) +
                (data.at(i + 1).first - data.at(i).first) * beta.at(i),
              3);
    b.push_back(b_val);
  }
}
coefs = std::tuple<std::vector<std::pair<long double, long double>>,
```

```
                std::vector<long double>, std::vector<long double>>(grid,
                                                                    a, b);

  return coefs;
}

long double get_point_approximation(
    std::tuple<std::vector<std::pair<long double, long double>>,
               std::vector<long double>, std::vector<long double>>
        coefs,
    long double x, std::string::size_type i) {
  long double point_approximation;
  point_approximation =
      std::get<0>(coefs).at(i + 1).second *
          (x - std::get<0>(coefs).at(i).first) /
          (std::get<0>(coefs).at(i + 1).first -
           std::get<0>(coefs).at(i).first) +
      std::get<0>(coefs).at(i).second *
          (x - std::get<0>(coefs).at(i + 1).first) /
          (std::get<0>(coefs).at(i).first -
           std::get<0>(coefs).at(i + 1).first) +
      (x - std::get<0>(coefs).at(i).first) *
          (x - std::get<0>(coefs).at(i + 1).first) *
          (std::get<1>(coefs).at(i) * x + std::get<2>(coefs).at(i));

  return point_approximation;
}

std::vector<std::pair<long double, long double>> get_approximation(
    std::vector<std::pair<long double, long double>> data_points,
```

```
    std::tuple<std::vector<std::pair<long double, long double>>,
               std::vector<long double>, std::vector<long double>>
        coefs) {
  std::vector<std::pair<long double, long double>> approximation;
  std::pair<long double, long double> point;
  std::string::size_type i;
  long double x;
  long double y;

  for (i = 0; i < data_points.size() - 2; ++i) {
    x = data_points.at(i + 1).first;
    y = get_point_approximation(coefs, x, i);
    point = std::pair<long double, long double>(x, y);
    approximation.push_back(point);
  }

  return approximation;
}

void save_approximation(
    std::string file_name,
    std::vector<std::pair<long double, long double>> approximation) {
  std::ofstream file_stream;
  std::string::size_type i;

  file_stream.open(file_name);
  for (i = 0; i < approximation.size(); ++i) {
    file_stream << approximation.at(i).first << " "
                << approximation.at(i).second << std::endl;
  }
```

```
  file_stream.close();
}

long double get_point_derivative_square_approximation(
    std::tuple<std::vector<std::pair<long double, long double>>,
               std::vector<long double>, std::vector<long double>>
        coefs,
    long double x, std::string::size_type i) {
  long double point_derivative_square_approximation;
  point_derivative_square_approximation = std::pow(
      std::get<0>(coefs).at(i + 1).second /
          (std::get<0>(coefs).at(i + 1).first -
           std::get<0>(coefs).at(i).first) +
          std::get<0>(coefs).at(i).second /
          (std::get<0>(coefs).at(i).first -
           std::get<0>(coefs).at(i + 1).first) +
          (x - std::get<0>(coefs).at(i).first) *
          (std::get<1>(coefs).at(i) *
               (2 * x - std::get<0>(coefs).at(i + 1).first) +
           std::get<2>(coefs).at(i)) +
          (x - std::get<0>(coefs).at(i + 1).first) *
          (std::get<1>(coefs).at(i) * x + std::get<2>(coefs).at(i)),
      2);

  return point_derivative_square_approximation;
}

std::vector<std::pair<long double, long double>>
get_derivative_square_approximation(
    std::vector<std::pair<long double, long double>> data_points,
```

```
  std::tuple<std::vector<std::pair<long double, long double>>,
             std::vector<long double>, std::vector<long double>>
      coefs) {
  std::vector<std::pair<long double, long double>>
    derivative_square_approximation;
  std::pair<long double, long double> point;
  std::string::size_type i;
  long double x;
  long double y;

  for (i = 0; i < data_points.size() - 2; ++i) {
    x = data_points.at(i + 1).first;
    y = get_point_derivative_square_approximation(coefs, x, i);
    point = std::pair<long double, long double>(x, y);
    derivative_square_approximation.push_back(point);
  }

  return derivative_square_approximation;
}

void save_derivative_square_approximation(
    std::string file_name, std::vector<std::pair<long double, long double>>
                           derivative_square_approximation) {
  std::ofstream file_stream;
  std::string::size_type i;

  file_stream.open(file_name);
  for (i = 0; i < derivative_square_approximation.size(); ++i) {
    file_stream << derivative_square_approximation.at(i).first << " "
                << derivative_square_approximation.at(i).second << std::endl;
```

```
  }
  file_stream.close();
}

std::vector<std::pair<long double, long double>> detect_qrs(
    std::vector<std::pair<long double, long double>> data,
    std::vector<std::pair<long double, long double>>
        derivative_square_approximation,
    long double threshold, int radius) {
  std::vector<std::pair<long double, long double>> qrs;
  std::pair<long double, long double> point;
  std::string::size_type i;
  std::string::size_type j;
  std::string::size_type j_max;
  long double max_derivative_square_approximation;
  long double max_derivative_square_approximation_i;
  long double max_data;
  long double max_data_j;
  long double x;
  long double y;

  max_derivative_square_approximation = -1;
  for (i = 0; i < derivative_square_approximation.size(); ++i) {
    if (derivative_square_approximation.at(i).second > threshold) {
      if (derivative_square_approximation.at(i).second >
          max_derivative_square_approximation) {
        max_derivative_square_approximation =
            derivative_square_approximation.at(i).second;
        max_derivative_square_approximation_i = i;
      }
```

```
    } else if (max_derivative_square_approximation != -1) {
      max_data = -1;
      j = max_derivative_square_approximation_i - radius + 1;
      if (j < 0) {
        j = 0;
      }
      j_max = max_derivative_square_approximation_i + radius + 1;
      if (j_max >= data.size()) {
        j_max = data.size() - 1;
      }
      for (; j <= j_max; ++j) {
        if (data.at(j).second > max_data) {
          max_data = data.at(j).second;
          max_data_j = j;
        }
      }
      x = data.at(max_data_j).first;
      y = max_data;
      point = std::pair<long double, long double>(x, y);
      qrs.push_back(point);
      max_derivative_square_approximation = -1;
    }
  }
  qrs.erase(std::unique(qrs.begin(), qrs.end()), qrs.end());

  return qrs;
}

void save_qrs(std::string file_name,
              std::vector<std::pair<long double, long double>> qrs) {
```

```
  std::ofstream file_stream;
  std::string::size_type i;

  file_stream.open(file_name);
  for (i = 0; i < qrs.size(); ++i) {
    file_stream << qrs.at(i).first << " " << qrs.at(i).second << std::endl;
  }
  file_stream.close();
}

int main() {
  std::tuple<std::vector<std::pair<long double, long double>>,
             std::vector<long double>, std::vector<long double>>
      coefs;
  std::vector<std::pair<long double, long double>> data;
  std::vector<std::pair<long double, long double>> approximation;
  std::vector<std::pair<long double, long double>>
      derivative_square_approximation;
  std::vector<long double> beta;
  std::vector<std::pair<long double, long double>> qrs;
  std::string data_file_name;
  std::string beta_file_name;
  std::string threshold_file_name;
  std::string radius_file_name;
  std::string approximation_file_name;
  std::string derivative_square_approximation_file_name;
  std::string qrs_file_name;
  long double threshold;
  long double radius;
```

```
  data_file_name = "data/TAY_FI.DAT";
  beta_file_name = "data/TAY_FI.DAT.BETA";
  threshold_file_name = "data/TAY_FI.DAT.THRS";
  radius_file_name = "data/TAY_FI.DAT.RAD";
  approximation_file_name = "data/TAY_FI.DAT.APRX";
  derivative_square_approximation_file_name = "data/TAY_FI.DAT.DRV";
  qrs_file_name = "data/TAY_FI.DAT.QRS";
  data = read_data(data_file_name);
  beta = read_beta(beta_file_name);
  coefs = get_coefs(data, beta);
  approximation = get_approximation(data, coefs);
  save_approximation(approximation_file_name, approximation);
  derivative_square_approximation =
     get_derivative_square_approximation(data, coefs);
  save_derivative_square_approximation(
     derivative_square_approximation_file_name,
     derivative_square_approximation);
  threshold = read_threshold(threshold_file_name);
  radius = read_threshold(radius_file_name);
  qrs = detect_qrs(data, derivative_square_approximation, threshold, radius);
  save_qrs(qrs_file_name, qrs);

  return 0;
}
```