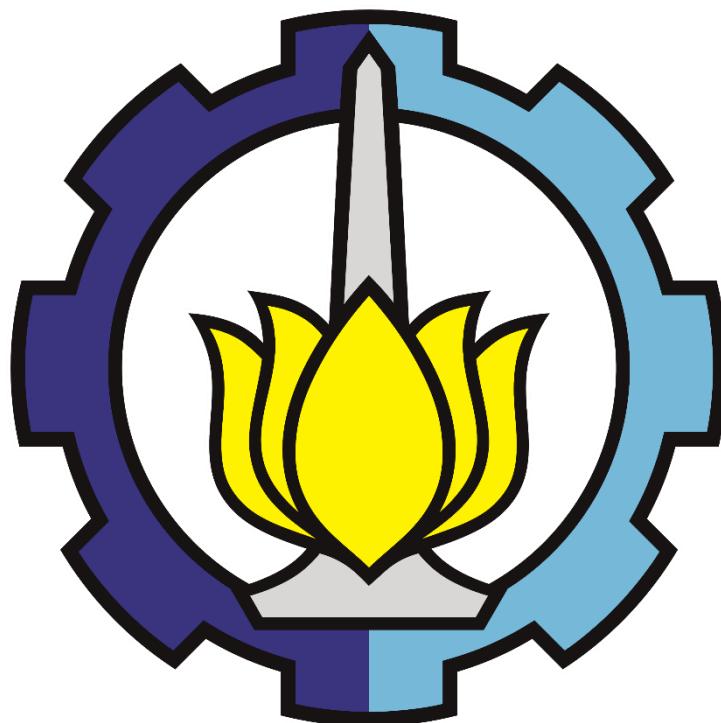


**LAPORAN FINAL PROJECT
DASAR PENGOLAHAN SINYAL**



Disusun oleh:

Ahmad Farhat

5023221060

Dosen Pengampu:

Nada Fitrieyatul Hikmah, S.T., M.T.

**DEPARTEMEN TEKNIK BIOMEDIK
FAKULTAS TEKNOLOGI ELEKTRO DAN INFORMATIKA CERDAS 2024**

KATA PENGANTAR

Segala puji dan syukur kita panjatkan ke hadirat Allah SWT atas limpahan rahmat dan karunia-Nya sehingga saya dapat menyelesaikan laporan final project mata kuliah Dasar Pengolah Sinyal ini.

Dalam penyusunan laporan ini, banyak pihak yang telah memberikan bantuan dan dukungan. Oleh karena itu, pada kesempatan ini, saya ingin menyampaikan rasa terima kasih yang sebesar-besarnya kepada Dosen Pengajar pada Mata Kuliah Dasar Pengolah Sinyal, yang telah memberikan bimbingan, ilmu, dan arahan selama proses perkuliahan berlangsung. Ajaran dan penjelasan yang diberikan mampu membantu saya dalam memahami materi dengan baik dan menyelesaikan proyek ini.

Hormat saya,

Penulis

DAFTAR ISI

LAPORAN FINAL PROJECT.....	1
KATA PENGANTAR.....	2
DAFTAR ISI	3
BAB I	5
1.2 Rumusan Masalah	5
1.3 Tujuan.....	5
BAB II	6
2.1. Elektrokardiogram.....	6
2.2. <i>Discrete Fourier Transform</i>	6
2.3. Transformasi Laplace.....	7
2.4. Filter Digital	8
2.5. Low-Pass Filter (LPF).....	10
2.6. Moving Average (MAV)	10
2.7 Bandpass Filter	11
BAB III	12
3.1. Pembahasan Program (Python)	12
3.1.1 Masukkan Library dan Parameter Awal	12
3.1.2. Memasukkan Data dan Membaca Data	12
3.1.3. Restorasi Baseline Pada Data	14
3.1.4. DFT 1.....	15
3.1.5. Low-Pass Filer (Forward & Backward).....	16
3.1.6. Moving Average (MAV)	20
3.1.7. Segmentasi P, QRS, dan T.....	21
3.1.8. Absolute	27
3.1.9. Moving Average 2 (MAV 2)	28
3.1.10. Threeshold Sinyal	30
3.1.11. Segmentasi T (LPF)	33
3.1.12. Absolut Segmentasi T.....	37
3.1.13. Thresshold Segmentasi T.....	39
3.1.14. Alternatif Segmentasi T	40
3.1.15. Hasil Segmentasi QRS dan T	43
3.2. Study Case Final Project	44

BAB IV.....	47
DAFTAR PUSTAKA.....	48

BAB I

PENDAHULUAN

1.1 Latar Belakang

Pengolahan sinyal adalah bidang studi yang memfokuskan pada teknik dan algoritma untuk menganalisis, memodifikasi, dan menafsirkan berbagai jenis sinyal, termasuk sinyal suara, gambar, dan sinyal biomedis. Salah satu aplikasi penting dari pengolahan sinyal adalah dalam analisis sinyal elektrokardiogram (ECG), yang mencerminkan aktivitas listrik jantung dan sangat penting dalam diagnosis medis. Dengan memanfaatkan deret Fourier dan transformasi Fourier, sinyal dapat diolah menjadi komponen frekuensi, memungkinkan identifikasi dan penghapusan *noise* (gangguan) serta mendapatkan informasi penting dari sinyal tersebut. Proses ini membantu dalam memberikan representasi yang lebih akurat dari data yang dikumpulkan. Selain itu, desain dan implementasi filter digital, seperti filter low-pass, high-pass, bandpass filter, dan bandstop filter digunakan untuk mengurangi noise dan artefak yang tidak diinginkan, sehingga meningkatkan kualitas sinyal yang dianalisis. Mata kuliah Dasar Pengolahan Sinyal memberikan pengetahuan dasar tentang prinsip-prinsip ini, mempersiapkan mahasiswa untuk menerapkan berbagai teknik pengolahan sinyal dalam berbagai aplikasi praktis, dari teknologi komunikasi hingga pemrosesan gambar dan analisis data biomedik.

1.2 Rumusan Masalah

1. Bagaimana cara menerapkan Transformasi Fourier Diskrit (DFT) untuk menganalisis sinyal elektrokardiogram (ECG) sehingga komponen frekuensi dari sinyal tersebut dapat diidentifikasi dengan tepat?
2. Bagaimana mendesain dan mengimplementasikan filter digital untuk mengurangi noise dan artefak dalam sinyal ECG, sehingga memungkinkan penentuan denyut jantung (BPM) yang akurat dari sinyal yang telah difilter?

1.3 Tujuan

1. Mahasiswa mampu menerapkan Transformasi Fourier Diskrit (DFT) pada sinyal ECG untuk menganalisis dan mengidentifikasi komponen frekuensi utama dari sinyal tersebut.
2. Mahasiswa mampu mendesain dan mengimplementasikan filter digital untuk membersihkan sinyal ECG dari noise dan artefak, serta mengukur denyut jantung (BPM) dengan akurat dari sinyal yang telah difilter.

BAB II

DASAR TEORI

2.1. Elektrokardiogram

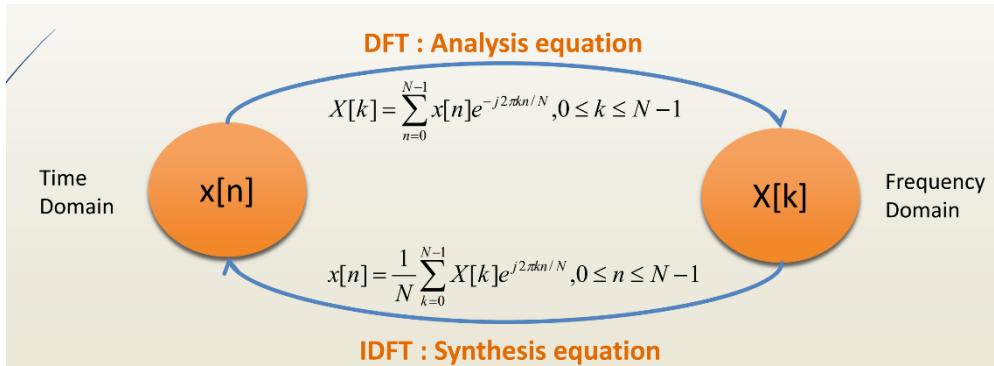
Elektrokardiogram (ECG) adalah alat yang digunakan untuk merekam aktivitas listrik jantung melalui elektroda yang ditempatkan di permukaan kulit. Sinyal yang dihasilkan memberikan informasi tentang berbagai aspek dari fungsi jantung, termasuk detak jantung, ritme, dan kondisi listrik secara keseluruhan. Gelombang ECG terdiri dari beberapa komponen utama: gelombang P, yang mewakili depolarisasi atrium; kompleks QRS, yang menunjukkan depolarisasi ventrikel; dan gelombang T, yang menunjukkan repolarisasi ventrikel (Jevon, 2010). Analisis sinyal ECG sangat penting dalam diagnosis dan pemantauan penyakit jantung seperti aritmia, iskemia, dan infark miokard. Karena sinyal ECG dapat dipengaruhi oleh berbagai jenis gangguan atau noise, pengolahan sinyal yang tepat sangat diperlukan untuk mendapatkan hasil yang akurat dan dapat diandalkan (Lippincott Williams & Wilkins, 2015).

2.2. Discrete Fourier Transform

Transformasi Fourier Diskrit (DFT) adalah alat matematika yang digunakan untuk menganalisis frekuensi komponen dari sinyal digital. DFT mengubah sinyal dari domain waktu ke domain frekuensi, memungkinkan identifikasi dan analisis spektrum frekuensi dari sinyal tersebut (Proakis & Manolakis, 2007). Rumus umum untuk DFT diberikan oleh persamaan :

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N} = \sum_{n=0}^{N-1} x[n]W_N^{kn}, \quad \text{for } k=0,1, \dots, N-1$$

Dimana $X(k)$ adalah nilai DFT pada frekuensi k , $x(n)$ adalah nilai sinyal pada waktu n , N adalah jumlah total data/sampel, dan j adalah unit imajiner (Oppenheim & Schafer, 2010). Implementasi DFT memungkinkan pemrosesan sinyal yang lebih efisien, terutama dalam aplikasi seperti pengolahan sinyal digital, pemrosesan gambar, dan analisis spektrum (Smith, 2003). Dalam desain filter digital, DFT digunakan untuk mengidentifikasi komponen frekuensi yang tidak diinginkan dan menghilangkannya, sehingga meningkatkan kualitas sinyal yang diproses (Proakis & Manolakis, 2007).



Gambar 1. DFT dan IDFT

Selanjutnya yaitu untuk pembahasan gambar yang diberikan yaitu secara konsep dasarnya di mana bertujuan untuk merubah domain seperti yang telah disampaikan di atas, di mana pada domain waktu sinyal digital mempresentasikan amplitudo dari sinyal dengan perbandingan pada *sampling time* nya atau jumlah sampel. Pada domain frekuensi mempresentasikan sinyal digital untuk komponen frekuensinya, atau bisa disebut *Signal Spectrum*. Dimana juga terdapat *Amplitude Spectrum* dan *Power Spectrum* dimana berfungsi untuk mendapatkan jumlah dan titik sampel pada domain waktu atau frekuensi.

Amplitude spectrum :

$$A_k = \frac{1}{N} |X(k)| = \frac{1}{N} \sqrt{(\text{Real}[X(k)])^2 + (\text{Imag}[X(k)])^2},$$

$$k = 0, 1, 2, \dots, N-1$$

Power spectrum :

$$P_k = \frac{1}{N^2} |X(k)|^2 = \frac{1}{N^2} \{(\text{Real}[X(k)])^2 + (\text{Imag}[X(k)])^2\}$$

$$k = 0, 1, 2, \dots, N-1$$

Frequency in k to its corresponding frequency :

$$f = \frac{k f_s}{N}$$

Frequency resolution :

$$\Delta f = \frac{f_s}{N} (\text{Hz})$$

Gambar 2. Rumus Amplitude dan Power Spectrum

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N} = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad \text{for } k=0, 1, \dots, N-1$$

- ▶ Equation can be expanded as :

$$X[k] = x[0]W_N^{k0} + x[1]W_N^{k1} + x[2]W_N^{k2} + \dots + x[N-1]W_N^{k(N-1)}$$

- ▶ W_N ➔ twiddle factor, is defined as :

$$W_N = e^{-j2\pi/N} = \cos\left(\frac{2\pi}{N}\right) - j \sin\left(\frac{2\pi}{N}\right)$$

- ▶ The **inverse DFT** is given by :

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j2\pi kn/N} = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-kn}, \quad \text{for } n=0, 1, \dots, N-1$$

Gambar 3. Rumus DFT

2.3. Transformasi Laplace

Transformasi Laplace adalah alat matematika yang penting dalam analisis sistem dinamis dan teori kendali. Digunakan untuk mengubah persamaan diferensial biasa (ODE) ke dalam domain kompleks. Transformasi Laplace (TF) juga berperan penting pada Dasar Pengolahan Sinyal (DPS) dimana mampu digunakan untuk menganalisis sinyal yang kontinu serta transformasi Laplace memungkinkan analisis sistem yang

kompleks secara lebih mudah dengan mengubah domain waktu menjadi domain frekuensi kompleks. Rumus transformasi Laplace dari fungsi waktu $f(t)$ yaitu :

$$X(s) = L\{x(t)\} = \int_0^{\infty} x(t)e^{-st} dt$$

$$x(t) = L^{-1}\{X(s)\} = \frac{1}{2\pi j} \int_{\gamma-j\infty}^{\gamma+j\infty} X(s)e^{st} ds.$$

Gambar 4. Rumus Transformasi Laplace

Dimana selain rumus di atas, Tranformasi Laplace juga mempunyai tabel domain untuk dalam domain waktu $x(t)$ dan dalam transformasi laplace $X(s) = L(x(t))$, dimana TF juga dapat digunakan untuk fungsi unit step. Lalu, TF sendiri bisa mendeskripsikan sistem linear analog yaitu sebagai berikut :

$$Y(s) = H(s)X(s), \quad \longrightarrow \quad H(s) = \frac{Y(s)}{X(s)}$$

Gambar 5. Rumus Domain Transformasi Laplace

Jika impuls fungsinya sama dengan *input* (masukan) untuk linear sistemnya, $x(t) = \delta(t)$, dimana laplace trasnformnya yaitu $X(s) = 1$ maka :

$$Y(s) = H(s)X(s) = H(s).$$

$$h(t) = L^{-1}\{H(s)\}.$$

Gambar 6. Rumus Impuls Kestabilan

Dimana dari hasil tersebut, bisa didapatkan juga kestabilan sistem dengan cara memplot s-plane dan z-plane dimana nanti bisa menemukan domain $X(z)$ atau domain untuk kestabilan sistem. Transformasi Laplace memungkinkan analisis sistem dalam domain frekuensi kompleks, di mana perilaku sistem dapat dijelaskan lebih detail dan solusi untuk persamaan diferensial dapat dicari dengan pendekatan yang lebih sistematis (Karris, 2007). Dalam aplikasi teknik dan ilmu pengetahuan lainnya, transformasi Laplace digunakan secara luas untuk memodelkan sistem kontrol, analisis sinyal, dan solusi masalah matematika yang melibatkan ODE kompleks (Kuo & Golnaraghi, 2002).

2.4. Filter Digital

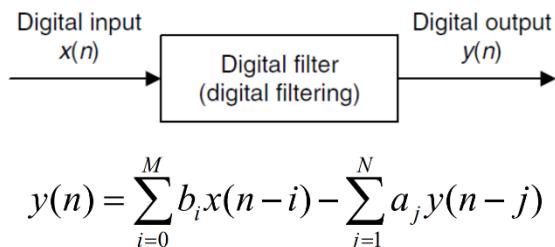
Filter digital adalah alat penting dalam pengolahan sinyal digital yang digunakan untuk memodifikasi karakteristik sinyal dengan cara memilih atau menolak komponen frekuensi tertentu. Filter ini dapat diterapkan dalam berbagai aplikasi, termasuk pemrosesan sinyal audio, pengolahan gambar, dan analisis sinyal biomedis seperti elektrokardiogram (ECG).

Fourier Transform		z Transform
$H(e^{j\Omega}) = \sum_{n=-\infty}^{\infty} h(n) \cdot e^{-j\Omega n}$	\longleftrightarrow	$H(z) = \sum_{n=-\infty}^{\infty} h(n)z^{-n}$

Where : $z = e^{j\Omega}$

Gambar 7. Rumus Transformasi Domain

Salah satu jenis filter digital yang umum digunakan adalah filter FIR (Finite Impulse Response) dan IIR (Infinite Impulse Response), yang masing-masing memiliki karakteristik dan metode desain yang berbeda (Oppenheim & Schafer, 1999). Rumus umum untuk filter FIR adalah:



where b_i , $0 \leq i \leq M$ and a_j , $1 \leq j \leq N$, represent the coefficients of the system and n is the time index.

Gambar 8. Rumus Filter FIR

Dari rumus tersebut di mana $y(n)$ adalah keluaran filter pada waktu n , $x[n-i]$ adalah *input* (masukan) pada waktu *delay* i . Filter FIR memiliki tanggapan impuls yang terbatas, sementara filter IIR memiliki tanggapan impuls yang tak terbatas, yang memungkinkan desain yang lebih fleksibel namun lebih kompleks (Proakis & Manolakis, 2007). Desain filter digital melibatkan penentuan koefisien filter yang tepat untuk mencapai karakteristik penyaringan yang diinginkan, seperti menekan noise frekuensi tinggi atau menyoroti komponen frekuensi tertentu dari sinyal input (Karris, 2007).

2.5. Low-Pass Filter (LPF)

Filter low-pass adalah jenis filter yang memungkinkan frekuensi rendah melewati sementara menolak atau mengurangi frekuensi tinggi. Filter ini digunakan dalam berbagai aplikasi pengolahan sinyal untuk menghilangkan noise frekuensi tinggi dari sinyal, seperti dalam pemrosesan audio, pengolahan gambar, dan analisis sinyal biomedis seperti elektrokardiogram (ECG) (Oppenheim & Schafer, 1999). Rumus umum untuk filter low-pass ideal dalam domain frekuensi adalah :

$$Lowpass \quad h(i) = \begin{cases} \frac{\Omega_c}{\pi} & \Rightarrow i = 0 \\ \frac{\sin(\Omega_c i)}{i\pi} & \Rightarrow \text{for } i \neq 0 \Rightarrow -M \leq i \leq M \end{cases}$$

Gambar 9. Rumus Low Pass Filter (LPF)

Di mana untuk $h(i)$ adalah respon dari filter, dan juga terdapat f_c (*frequency cut-off*) yang berfungsi untuk perpindahan agar bisa terlewati atau menolak frekuensi yang ada/terjadi (Smith, 2003). Dalam implementasi digital, filter low-pass sering kali dirancang menggunakan algoritma seperti Transformasi Fourier Diskrit (DFT) untuk menentukan koefisien filter yang optimal (Proakis & Manolakis, 2007). Filter FIR (Finite Impulse Response) dan IIR (Infinite Impulse Response) dapat digunakan untuk mengimplementasikan filter low-pass, masing-masing dengan keuntungan dan kerugian terkait kestabilan dan linearitas fase (Karris, 2007). Filter low-pass sangat penting dalam memastikan kualitas sinyal yang tinggi dengan menghilangkan komponen frekuensi tinggi yang dapat mengganggu atau mendistorsi sinyal utama.

2.6. Moving Average (MAV)

Moving average adalah metode yang digunakan dalam pengolahan sinyal untuk menghaluskan data atau sinyal dengan menghitung rata-rata nilai data dalam jendela waktu tertentu. Teknik ini sangat berguna untuk mengurangi noise dan fluktuasi jangka pendek, sehingga tren jangka panjang lebih mudah diamati (Smith, 2003). Rumus umum untuk moving average sederhana diberikan oleh:

Moving Average Filter

- Filter equation : $y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i-j]$, $M = \text{order filter}$
- Frequency response :
 - Good smoothing
 - Bad low pass
$$H(f) = \frac{\sin(\pi f M)}{M \sin(\pi f)}$$

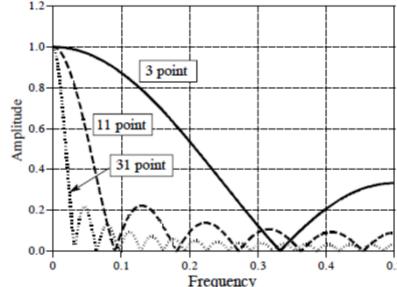
$$f = 0 \dots \frac{f_s}{2}$$

Forward

$$y_1[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i-j]$$

Backward

$$y_2[i] = \frac{1}{M} \sum_{j=0}^{M-1} y_1[i+j]$$



Gambar 10. Rumus Moving Average

Di mana $y[i]$ adalah nilai rata-rata pada waktu I , $y[i-j]$ adalah nilai data pada waktu tersebut, dan M adalah panjang orde. Moving average dapat diterapkan dalam dua cara: forward dan backward. Dalam moving average forward, jendela waktu hanya menggunakan data masa lalu dan nilai saat ini, sedangkan dalam moving average backward, jendela waktu mencakup data masa depan hingga nilai saat ini (Oppenheim & Schafer, 1999). Teknik moving average digunakan dalam berbagai aplikasi seperti analisis keuangan, pemrosesan sinyal audio, dan pengolahan sinyal biomedis untuk menghaluskan data dan mengurangi efek noise (Karris, 2007).

2.7 Bandpass Filter

Bandpass filter adalah jenis filter yang digunakan dalam pengolahan sinyal untuk melewati rentang frekuensi tertentu (bandpass) sementara menolak frekuensi di luar rentang tersebut. Filter ini sangat berguna dalam aplikasi di mana hanya frekuensi tertentu dari sinyal yang ingin diambil atau dianalisis, seperti dalam komunikasi nirkabel, pemrosesan sinyal audio, dan analisis sinyal biomedis seperti elektrokardiogram (ECG) (Oppenheim & Schafer, 1999). Rumus umum untuk filter bandpass dapat diberikan dalam domain frekuensi sebagai berikut :

$$Bandpass \quad h(i) = \begin{cases} \frac{\Omega_H - \Omega_L}{\pi} & \Rightarrow i = 0 \\ \frac{\sin(\Omega_H i)}{i\pi} - \frac{\sin(\Omega_L i)}{i\pi} & \Rightarrow \text{for } i \neq 0 \Rightarrow -M \leq i \leq M \end{cases}$$

Gambar 11. Rumus Bandpass Filter

Dapat dilihat bahwa $h(i)$ merupakan respons dari filter, Pada Bandpass Filter memakai (*Frequency Cut-Off Low* dan *High*) yang rentangnya antara batas atas dan bawah yang juga menjadi penentu rentang batas frekuensi yang dibiarkan untuk melewati filter. Implementasi bandpass filter dapat dilakukan dengan menggunakan teknik forward dan backward, tergantung pada arah sinyal yang diterapkan dalam filter. Bandpass filter digunakan untuk mengekstrak atau menekankan informasi pada rentang frekuensi tertentu dari sinyal, yang sering kali penting dalam aplikasi di mana deteksi atau analisis frekuensi spesifik diperlukan (Karris, 2007).

BAB III

ANALISIS DATA

3.1. Pembahasan Program (Python)

3.1.1 Masukkan Library dan Parameter Awal

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

fs = 125 # Frekuensi Sampling (Hz)
fc = 50 # Frekuensi Cut-Off (Hz)
fcl = 8 # Frekuensi Cut-Off Rendah
fch = 15 # Frekuensi Cut-Off Tinggi
M = 1 # Order
```

Program 1. Input Library dan Parameter Awal

Dimana pada program tersebut saya memasukkan (*import*) untuk fungsi yang dibutuhkan pada program ini sampai dengan selesai. Numpy berfungsi untuk fungsi perhitungan, MatPlotLib untuk plot dan menampilkan grafik, dan Pandas untuk membaca file serta mengolah data. Serta saya menetapkan beberapa parameter awal yang nantinya akan disesuaikan lagi pada setiap program sesuai kebutuhannya, namun untuk yang awal saya tetapkan sebagai berikut yang sesuai dengan gambar di atas.

3.1.2. Memasukkan Data dan Membaca Data

```
# Masukkan Data dan Baca

# Masukan ECG data
df = pd.read_csv('Data ECG Davis.txt', delimiter='\t')
df.columns = ['waktu', 'amplitudo']

# Buat time vector berdasarkan jumlah sampel dan Frekuensi Sampling
waktu_sinyal = np.arange(len(df['amplitudo'])) / fs
sekuens_sinyal = df['waktu']
#signal_sinyal = df['amplitudo']

# Tentukan ukuran plot nya
plt.figure(figsize=(15, 6)) # Adjust width and height as needed
# Plot Data-nya
plt.plot(waktu_sinyal, df['amplitudo'])
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('ECG Signal in Time Domain')
# Atur batas x-axis pada time vector
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
# Tunjukkan garis dari grid
plt.grid(True)
# Tunjukkan plotnya
```

```

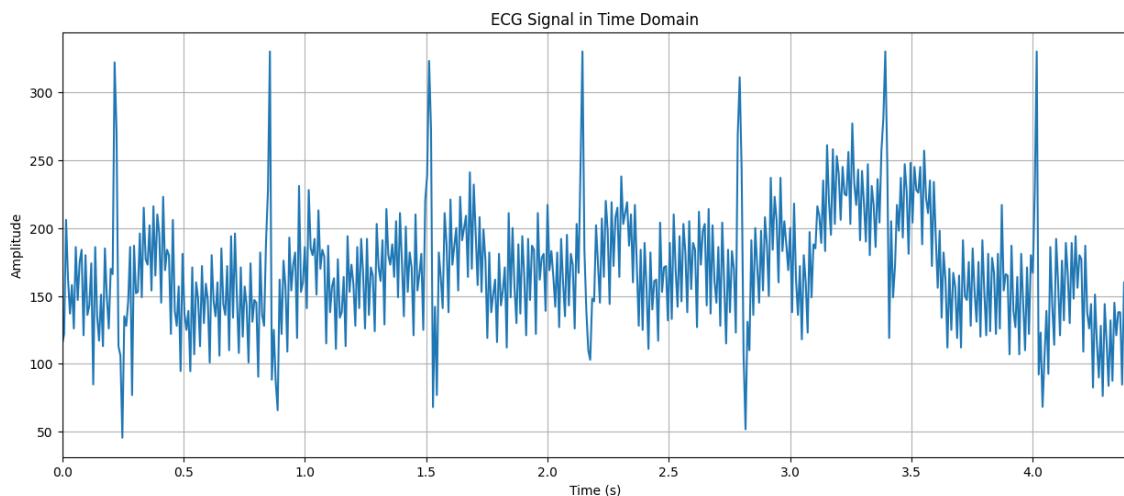
plt.show()

plt.figure(figsize=(15,6))
plt.plot(df['waktu'], df['amplitudo'])
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('ECG Signal in Time Domain')
plt.grid()
plt.show()

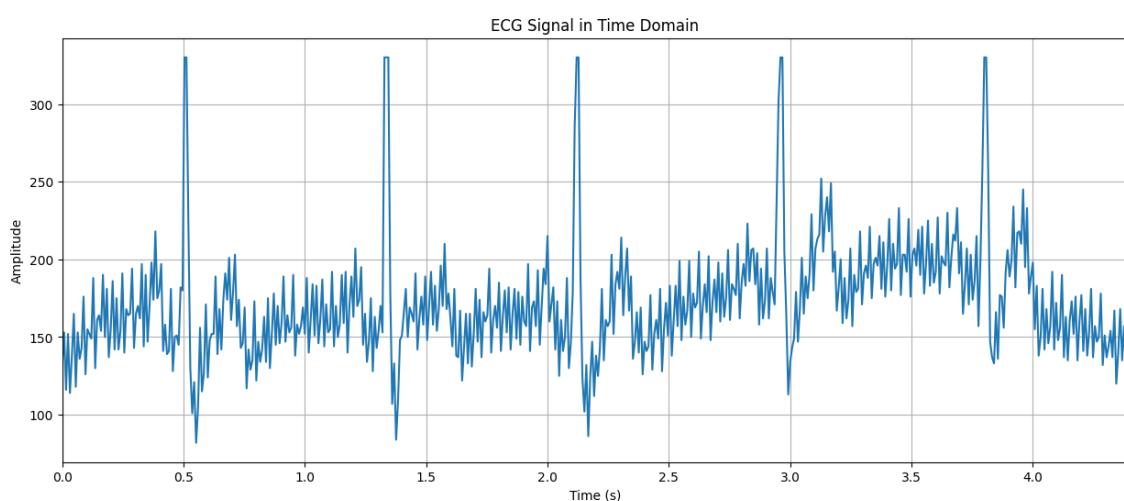
```

Program 2. Memasukkan dan Membaca Data

Pada program selanjutnya, saya membuat program untuk memasukkan dan membaca data ECG yang direkam agar selanjutnya bisa dipakai untuk pengolahan serta saya juga menentukan untuk domain nya dan men-*define* parameter yang diperlukan. Untuk hasil dari data didapatkan dalam bentuk grafik sebagai berikut :



Gambar 12. Hasil Plot Data ECG Asli Levy



Gambar 13. Hasil Plot Data ECG Asli Davis

Pada data tersebut dapat dilihat bahwa karena saat perekaman data ECG baik Levy/Davis masih terdapat kesalahan dalam garis dasarnya (*baseline*) karena tidak berada pada nol (0) sehingga harus di turunkan dulu sinyalnya sehingga kembali ke nol (0).

3.1.3. Restorasi Baseline Pada Data

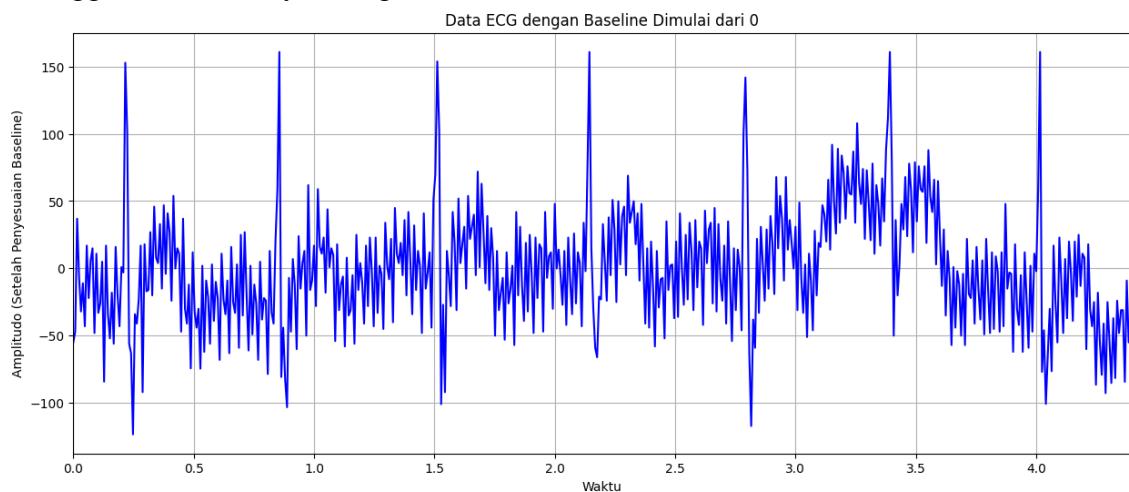
```
# Baseline Restoration

# Mendapatkan rata-rata amplitudo
baseline = np.mean(df['amplitudo'])
# Mengurangkan setiap nilai amplitudo dengan rata-rata amplitudo
restored_signal = df['amplitudo'] - baseline

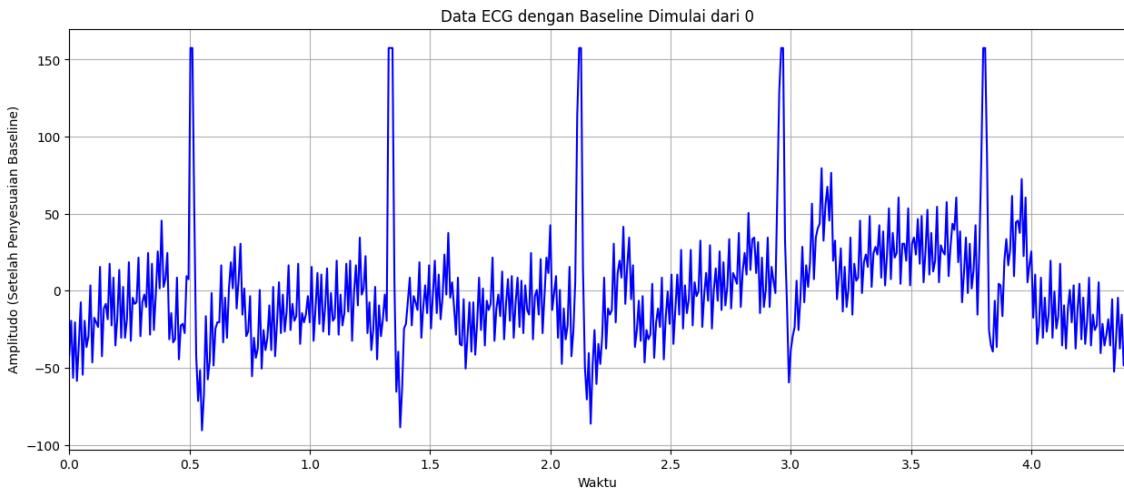
# Plot data setelah penyesuaian baseline
plt.figure(figsize=(15, 6))
plt.plot(waktu_sinyal, restored_signal, color='blue')
plt.xlabel('Waktu')
plt.ylabel('Amplitudo (Setelah Penyesuaian Baseline)')
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.title('Data ECG dengan Baseline Dimulai dari 0')
plt.grid(True)
plt.show()
```

Program 3. Restorasi Baseline

Pada program *Baseline Restoration* ini, saya memakai mean (nilai tengah) dari data lalu dari amplitudo yang awal saya kurangi dengan hasil mean untuk baselinennya, sehingga untuk hasilnya sebagai berikut :



Gambar 14. Hasil Restorasi *Baseline* ECG Levy



Gambar 15. Hasil Restorasi *Baseline* ECG Davis

Dari hasil tersebut dapat kita lihat bahwa dengan *baseline restoration*, kita dapat mengembalikan sinyal kembali ke nol (0) sehingga lebih mudah untuk dianalisis suntuk tahap selanjutnya.

3.1.4. DFT 1

```
# DFT Sinyal Asli

# Fungsi dari Discrete Fourier Transform (DFT)
def DFT(x):
    N = len(x)
    X_real = np.zeros(N)
    X_imag = np.zeros(N)

    for k in range(N):
        for n in range(N):
            X_real[k] += x[n] * np.cos(2 * np.pi * k * n / N)
            X_imag[k] -= x[n] * np.sin(2 * np.pi * k * n / N)

    mag = np.sqrt(X_real**2 + X_imag**2)

    return mag

# Hitung DFT dari sinyal yang telah direstorasi
magDFT = DFT(restored_signal)

# Plot magnitude spectrum
freq = np.arange(len(magDFT)) * fs / len(magDFT)
freq_half = freq[:len(magDFT)//2] # Half of the frequency range

plt.figure(figsize=(15, 5))
plt.stem(freq_half, magDFT[:len(magDFT)//2])
```

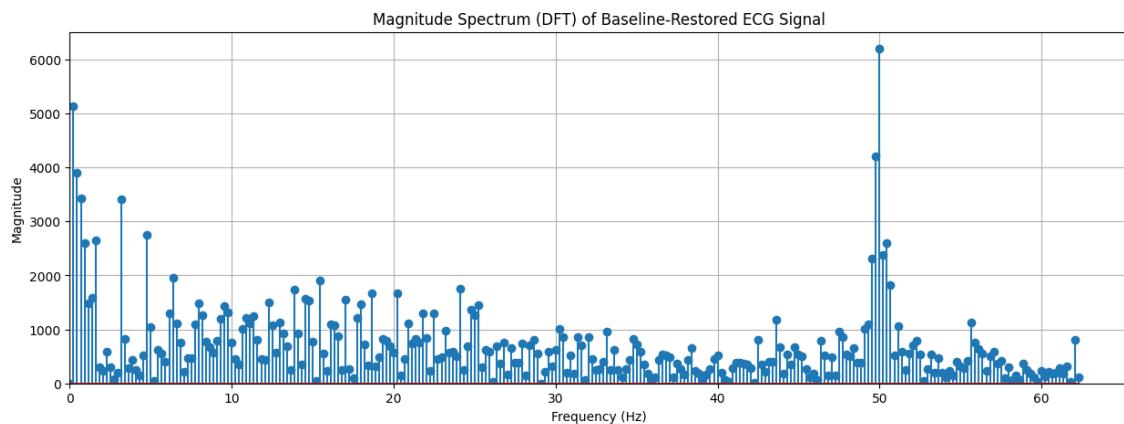
```

plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude')
plt.xlim(waktu_sinyal.min())
plt.ylim(magDFT[:len(magDFT)//2].min())
plt.title('Magnitude Spectrum (DFT) of Baseline-Restored ECG
Signal')
plt.grid(True)
plt.show()

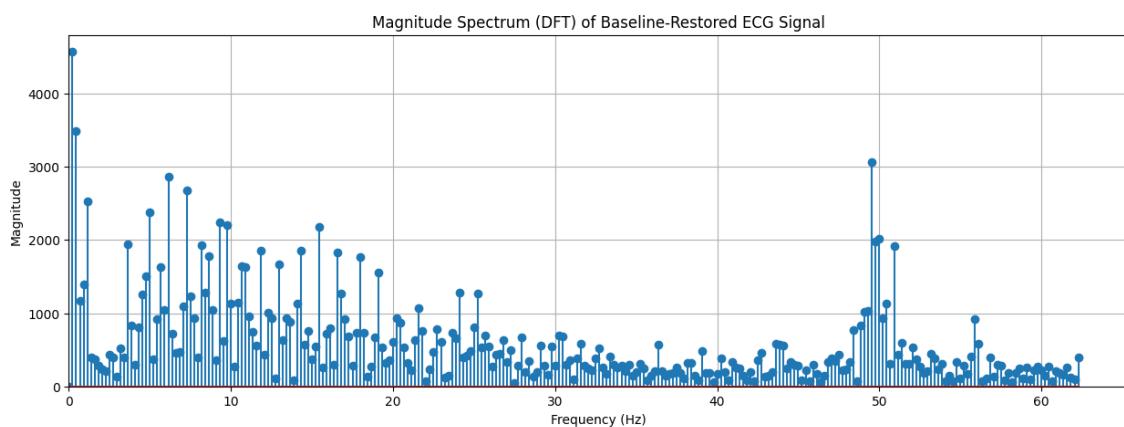
```

Program 4. DFT Sinyal

Pada program di atas yaitu *Discrete Fourier Transform* yang berguna untuk melihat data dalam bentuk diskrit sehingga bisa kita temukan untuk noise nya berada pada berapa Hz, sehingga bisa untuk program selanjutnya di mana pada titik noise tersebut akan kita potong sehingga sinyal bisa dianalisis lebih baik. Adapun rumusnya sesuai dengan dasar teori di atas dan ditampilkan dalam bentuk plot dengan hasilnya sebagai berikut :



Gambar 16. Hasil DFT Sinyal ECG Levy



Gambar 17. Hasil DFT Sinyal ECG Davis

Dapat dilihat bahwa pada Levy dan Davis mempunyai noise yang paling tinggi ketika pada frekuensi 50 Hz, sehingga harus kita hilangkan/kurangi agar mempermudah proses pengolahan data.

3.1.5. Low-Pass Filer (Forward & Backward)

```
# Low Pass Filter (LPF)
```

```

# Hitung ohm_c cut-off frekuensinya
ohm_c = 2 * np.pi * fc / fs

fs = 125 # Frekuensi Sampling (Hz)
M_LPF1 = 5 # Order

# Hitung koefisien filter untuk low-pass filter dari order 3 (2M + 1, M = 1) # Atau masukkan Order yang diinginkan di atas
arr_M = np.arange(-M_LPF1, M_LPF1 + 1, 1)
h = np.zeros(2 * M_LPF1 + 1)
for n in range(-M_LPF1, M_LPF1 + 1, 1):
    if n == 0:
        h[n + M_LPF1] = ohm_c / np.pi
    else:
        h[n + M_LPF1] = np.sin(n * ohm_c) / (n * np.pi)

# Jumlah Point Data
N_LPF = len(restored_signal)

# low-pass filter untuk (Forward Filtering)
x_filtered_forward_LPF = np.zeros(N_LPF)

# Forward Filtering
for i in range(N_LPF):
    for j in range(-M_LPF1, M_LPF1 + 1, 1):
        if 0 <= i - j < N_LPF: # Pastikan index berada pada batasnya
            x_filtered_forward_LPF[i] += h[j + M_LPF1] *
restored_signal[i - j]

# low-pass filter untuk (Backward Filtering)
x_filtered_backward_LPF = np.zeros(N_LPF)

# Backward Filtering
for i in range(N_LPF-1, -1, -1):
    for j in range(-M_LPF1, M_LPF1 + 1, 1):
        if 0 <= i + j < N_LPF: # Ensure index is within bounds
            x_filtered_backward_LPF[i] += h[j + M_LPF1] *
x_filtered_forward_LPF[i + j]

# Plot forward and backward filtered ECG signal pada 1 plot yang sama
plt.figure(figsize=(15, 5))

plt.plot(waktu_sinyal, restored_signal, label='Restored Signal')
plt.xlabel('Time (s)')

```

```

plt.ylabel('Amplitude')
plt.title('Restored ECG Signal')
plt.legend()
plt.grid(True)

plt.plot(waktu_sinyal, x_filtered_forward_LPF, label='Forward
Filtered Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Forward Filtered ECG Signal')
plt.legend()
plt.grid(True)

plt.plot(waktu_sinyal, x_filtered_backward_LPF, label='Forward-
Backward Filtered Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Combined Filtered ECG Signal')
plt.legend()
plt.grid(True)

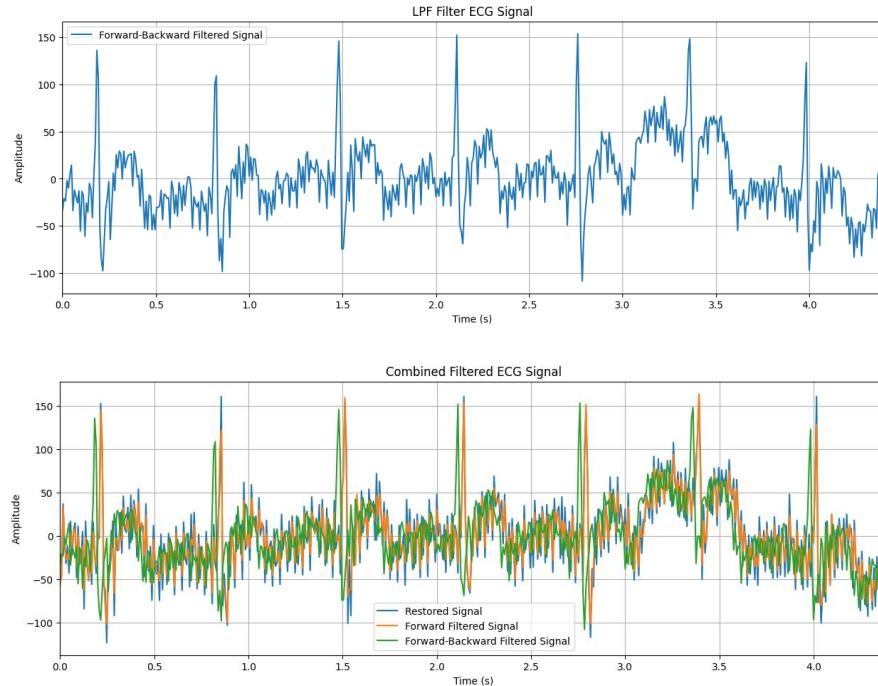
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.show()

# Plot forward and backward filtered ECG signal nya
plt.figure(figsize=(15, 5))
plt.plot(waktu_sinyal, x_filtered_backward_LPF, label='Forward-
Backward Filtered Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('LPF Filter ECG Signal')
plt.legend()
plt.grid(True)
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.show()

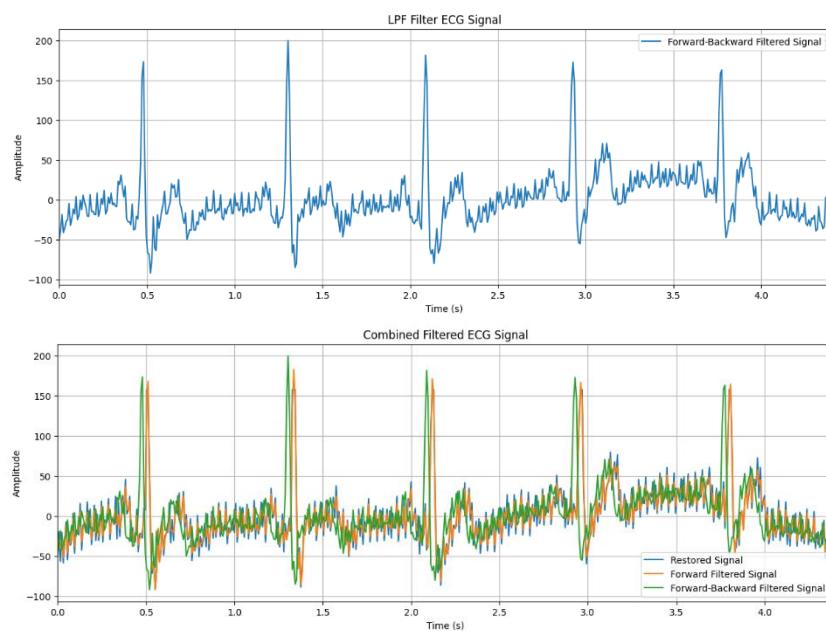
```

Program 5. Low-Pass Filtering

Pada program di atas yang bertujuan sebagai *Low-Pass Filter* yang berguna untuk memotong noise pada 50 Hz yang ditemukan pada sinyal ketika analisis DFT tadi, sehingga untuk memotong frekuensi tinggi, kita membutuhkan pemotongan untuk frekuensi yang tinggi dan membiarkan frekuensi rendah untuk lewat. Adapun hasil *Low-Pass Filter* bisa dilihat sebagai berikut :



Gambar 18. Hasil LPF Sinyal ECG Levy



Gambar 19. Hasil LPF Sinyal LPF Davis

Dari hasil yang didapatkan setelah LPF dan juga menggunakan Rumus Maju dan Mundur, yaitu sinyal menjadi lebih jelas walaupun tidak terlalu bersih tapi mengurangi noise nya sehingga bisa dilanjutkan ke tahap selanjutnya.

3.1.6. Moving Average (MAV)

```
# Moving Average (MAV)

# Parameter
M_MAV = 3
N = len(x_filtered_backward_LPF)

# Initialize the moving average result arrays
df_MAV_forward = np.zeros(N)

# Moving Average Filter Forward
for idx in range(N):
    for shift in range(M_MAV):
        if idx - shift >= 0:
            df_MAV_forward[idx] += x_filtered_backward_LPF[idx - shift]
    df_MAV_forward[idx] /= M_MAV

# Initialize the moving average result arrays
N_2 = len(df_MAV_forward)
df_MAV_backward = np.zeros(N_2)
#N = len(df_MAV_forward)

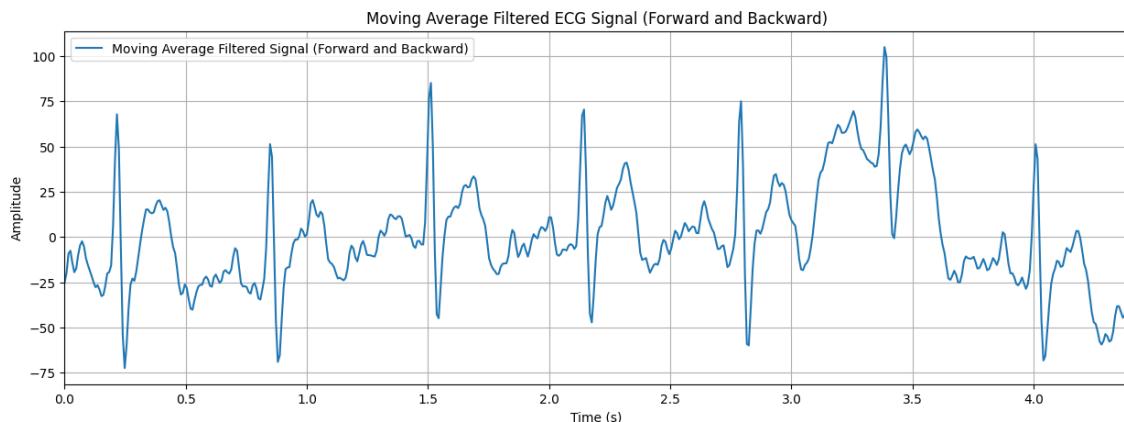
# Moving Average Filter Backward
for idx in range(N_2):
    for shift in range(M_MAV):
        if idx + shift < N_2:
            df_MAV_backward[idx] += df_MAV_forward[idx + shift]
    df_MAV_backward[idx] /= M_MAV

df['Hasil MAV'] = df_MAV_backward

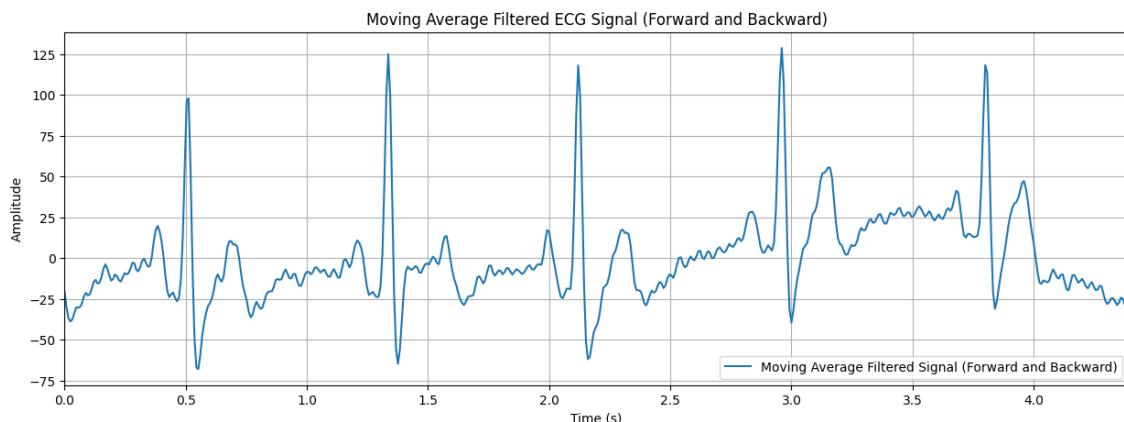
# Plotting the results
plt.figure(figsize=(15, 5))
plt.plot(waktu_sinyal, df_MAV_backward, label='Moving Average Filtered Signal (Forward and Backward)')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Moving Average Filtered ECG Signal (Forward and Backward)')
plt.legend()
plt.grid(True)
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.show()
```

Program 6. Moving Average Filtering

Pada program di atas merupakan sebuah filtering yaitu *moving average* (MAV) yang mempunyai tujuan untuk memfilter sinyal dari LPF sehingga hasil yang ditampilkan bisa lebih mulus, pada MAV kali ini juga sudah seperti rumus *backward* sehingga bisa kembali menjadi seperti sinyal aslinya namun lebih mulus, untuk menentukan kemulusannya bisa mengatur ordenya dimana semakin orde tinggi, maka semakin mulus hasil sinyal MAV nya, namun jika berlebih juga bisa membuat sinyal terlalu mulus sehingga menghilangkan bentuk aslinya. Oleh karena itu hasil dari MAV bisa dilihat sebagai berikut :



Gambar 20. Hasil *Moving Average* Sinyal Davis



Gambar 21. Hasil *Moving Average* Sinyal Davis

Dari hasil sinyal di atas dapat dilihat bahwa baik sinyal Levy atau Davis setelah melalui proses MAV akhirnya sinyal menjadi lebih mulus sehingga bisa dianalisis lebih baik di tahap selanjutnya.

3.1.7. Segmentasi P, QRS, dan T

```
# Segmentasi the P, QRS, and T waves
x = np.arange(len(df['Hasil MAV']))
y_mav = df['Hasil MAV']

# Batas Awal dan Akhir Segmentasi Levy
# P = 0.2 - 0.3
# QRS = 0.8 - 1.0
```

```

# T = 1.2 - 1.4

# Batas Awal dan Akhir Segmentasi Davis
# P =
# QRS =
# T =

# Definisikan segmentasi berdasarkan hasil interval waktu
mulai_p, akhir_p = 1.97, 2.00
mulai_qrs, akhir_qrs = 2.05, 2.17
mulai_t, akhir_t = 2.20, 2.32

# Buat bagian untuk setiap bagian segmentasi
bagian_p = (waktu_sinyal >= mulai_p) & (waktu_sinyal <= akhir_p)
bagian_qrs = (waktu_sinyal >= mulai_qrs) & (waktu_sinyal <=
akhir_qrs)
bagian_t = (waktu_sinyal >= mulai_t) & (waktu_sinyal <= akhir_t)

# Buat bagian untuk setiap bagian segmentasi
y_P = np.where(bagian_p, y_mav, 0)
y_QRS = np.where(bagian_qrs, y_mav, 0)
y_T = np.where(bagian_t, y_mav, 0)

plt.figure(figsize=(15, 5))
plt.plot(waktu_sinyal, df['Hasil MAV'])
plt.plot(waktu_sinyal, y_P, label='P Segment', color='red')
plt.plot(waktu_sinyal, y_QRS, label='QRS Segment', color='green')
plt.plot(waktu_sinyal, y_T, label='T Segment', color='blue')
plt.xlabel("Time (s)")
plt.ylabel("Amplitude (mV)")
plt.title("Segment P, QRS, dan T")
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.legend()
plt.grid()
plt.show()

P_segment = np.zeros(len(df['Hasil MAV']))
for i in range(len(df['Hasil MAV'])):
    if mulai_p <= waktu_sinyal[i] <= akhir_p:
        P_segment[i] = df['Hasil MAV'][i]
N_P = len(P_segment)

QRS_segment = np.zeros(len(df['Hasil MAV']))
for i in range(len(df['Hasil MAV'])):
    if mulai_qrs <= waktu_sinyal[i] <= akhir_qrs:
        QRS_segment[i] = df['Hasil MAV'][i]
N_QRS = len(QRS_segment)

```

```

T_segment = np.zeros(len(df['Hasil MAV']))
for i in range(len(df['Hasil MAV'])):
    if mulai_t <= waktu_sinyal[i] <= akhir_t:
        T_segment[i] = df['Hasil MAV'][i]
N_T = len(T_segment)

# Fungsi Perhitungan DFT
def compute_dft(df):
    N = len(df)
    X_real = np.zeros(N)
    X_imaj = np.zeros(N)
    MagDFT = np.zeros(N)

    for k in range(N):
        for n in range(N):
            X_real[k] += df[n] * np.cos(2 * np.pi * k * n / N)
            X_imaj[k] -= df[n] * np.sin(2 * np.pi * k * n / N)
        MagDFT[k] = np.sqrt(X_real[k]**2 + X_imaj[k]**2)

    return MagDFT

# Hitung DFT Segmentasi
MagDFT_P = compute_dft(P_segment)
MagDFT_QRS = compute_dft(QRS_segment)
MagDFT_T = compute_dft(T_segment)

fs = 125 # Sampling frequency

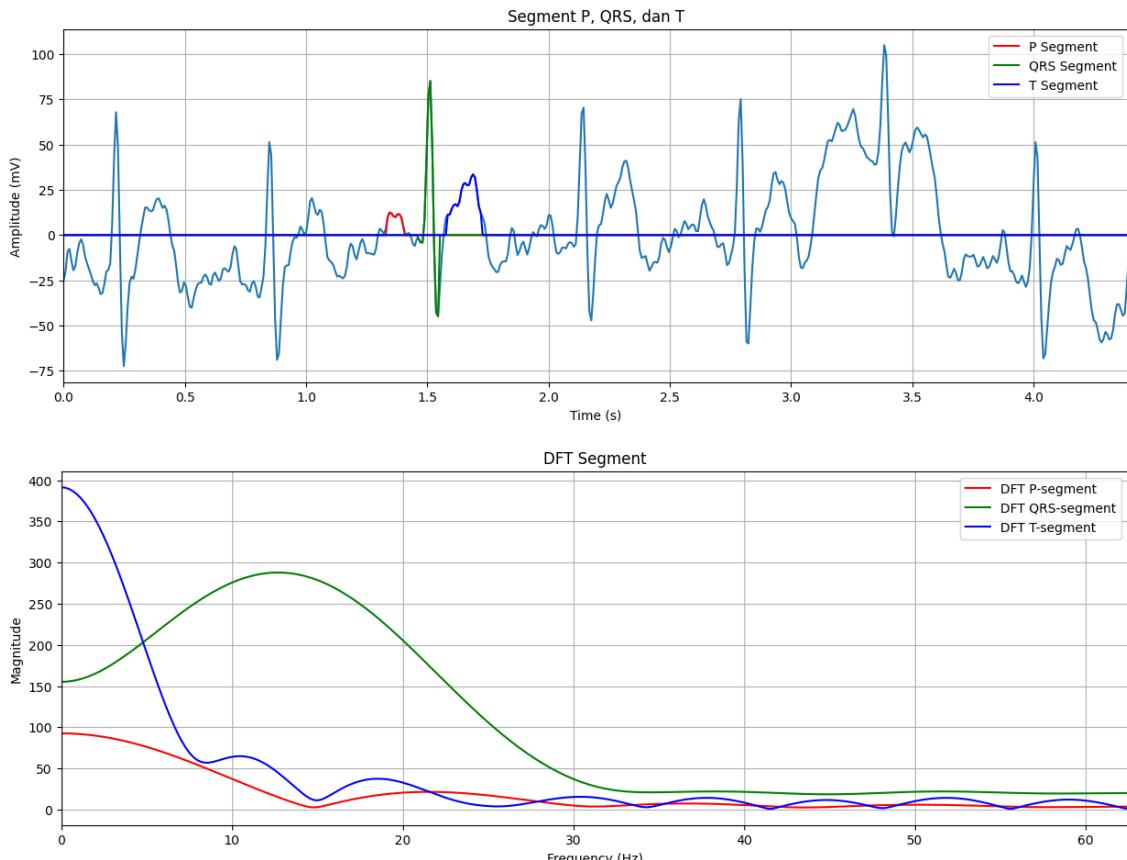
# Buat array untuk frekuensi nya
freq_P = np.arange(0, len(P_segment)) * fs / len(P_segment)
freq_QRS = np.arange(0, len(QRS_segment)) * fs / len(QRS_segment)
freq_T = np.arange(0, len(T_segment)) * fs / len(T_segment)

# Plot DFT dari setiap segmentasi
plt.figure(figsize=(15, 5))
plt.plot(freq_P, MagDFT_P, label='DFT P-segment', color='red')
plt.plot(freq_QRS, MagDFT_QRS, label='DFT QRS-segment',
         color='green')
plt.plot(freq_T, MagDFT_T, label='DFT T-segment', color='blue')
plt.title("DFT Segment")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Magnitude")
plt.xlim(0, fs / 2)
plt.grid()
plt.legend()
plt.show()

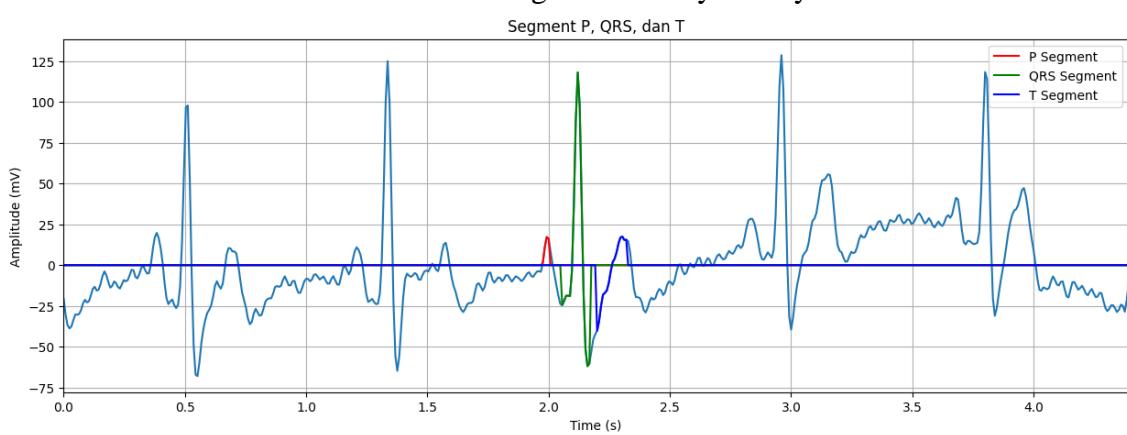
```

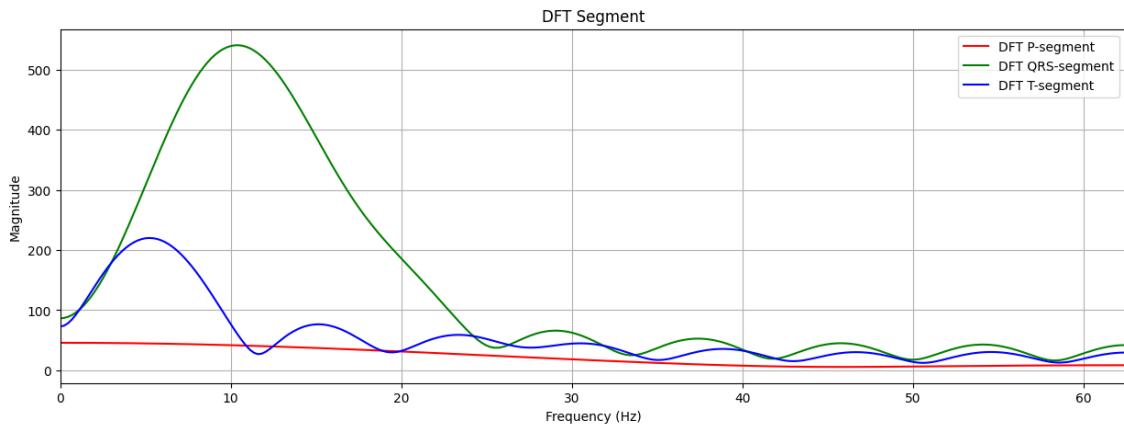
Program 7. Segmentasi Bagian P, QRS, dan T

Pada program di atas yaitu saya menggunakan cara manual untuk mendapatkan segmentasi yaitu dengan cara mengukur berdasarkan waktunya, sehingga saya bisa mendapatkan hasil segmentasi p, qrs, dan t sebagai berikut :



Gambar 22. Segmentasi Sinyal Levy





Gambar 23. Segmentasi Sinyal Davis

Dari data segmentasi di atas dapat kita lihat bahwa P itu bernilai lebih rendah dibanding keduanya, sedangkan QRS membentuk sebuah bukit, dan T bernilai bisa jadi lebih tinggi dari QRS atau lebih rendah. Dari sini kita bisa melihat untuk DFT sinyal segmentasinya.

3.1.7. Bandpass Filter (Forward and Backward)

```
# Parameter
fs = 125 # Frekuensi Sampling (Hz)
fcl = 8 # Lower cutoff frequency (Hz)
fch = 15 # Higher cutoff frequency (Hz)
M_BP = 25 # Order dari Bandpass

# Hitung dan Definisikan koefisien filter untuk Bandpass Filter
def bandpass_filter_coeff(fcl, fch, fs, M_BP):
    h_bandpass = np.zeros(2 * M_BP + 1)
    for n in range(-M_BP, M_BP + 1):
        if n == 0:
            h_bandpass[n + M_BP] = (2 * np.pi * fch / fs) / np.pi - (2 * np.pi * fcl / fs) / np.pi
        else:
            h_bandpass[n + M_BP] = (np.sin(2 * np.pi * fch * n / fs) / (np.pi * n)) - (np.sin(2 * np.pi * fcl * n / fs) / (np.pi * n))
    return h_bandpass

# Hitung koefisien filter untuk Bandpass Filter
h_bandpass = bandpass_filter_coeff(fcl, fch, fs, M_BP)

# Forward Filtering
N = len(df_MAV_backward)
x_filtered_forward = np.zeros(N)
for i in range(N):
    for j in range(-M_BP, M_BP + 1):
```

```

if 0 <= i - j < N:
    x_filtered_forward[i] += h_bandpass[j + M_BP] *
df_MAV_backward[i - j]

# Backward Filtering
x_filtered_backward_BPF = np.zeros(N)
for i in range(N-1, -1, -1):
    for j in range(-M_BP, M_BP + 1):
        if 0 <= i + j < N:
            x_filtered_backward_BPF[i] += h_bandpass[j + M_BP] *
x_filtered_forward[i + j]

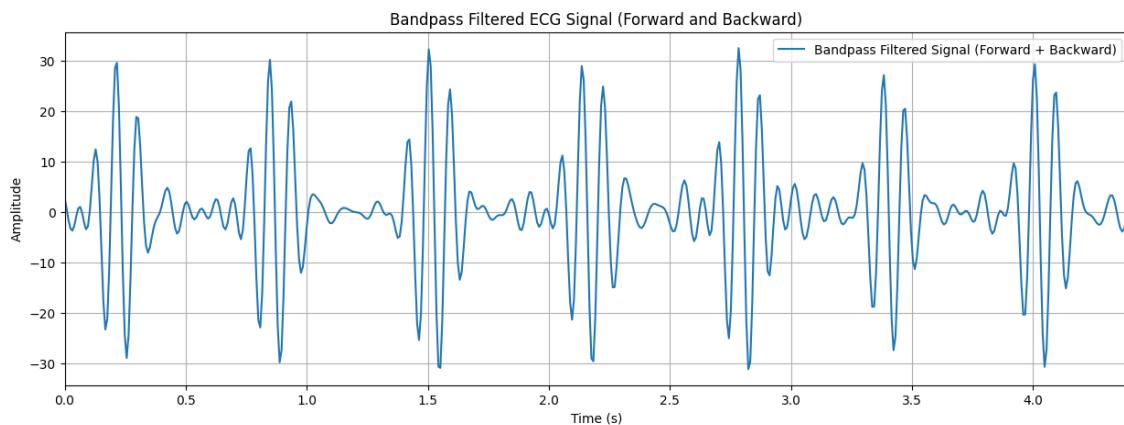
filtered_bandpass = x_filtered_backward_BPF

# Plot bandpass filtered signal
plt.figure(figsize=(15, 5))
plt.plot(waktu_sinyal, x_filtered_backward_BPF, label='Bandpass
Filtered Signal (Forward + Backward)')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Bandpass Filtered ECG Signal (Forward and Backward)')
plt.legend()
plt.grid(True)
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.show()

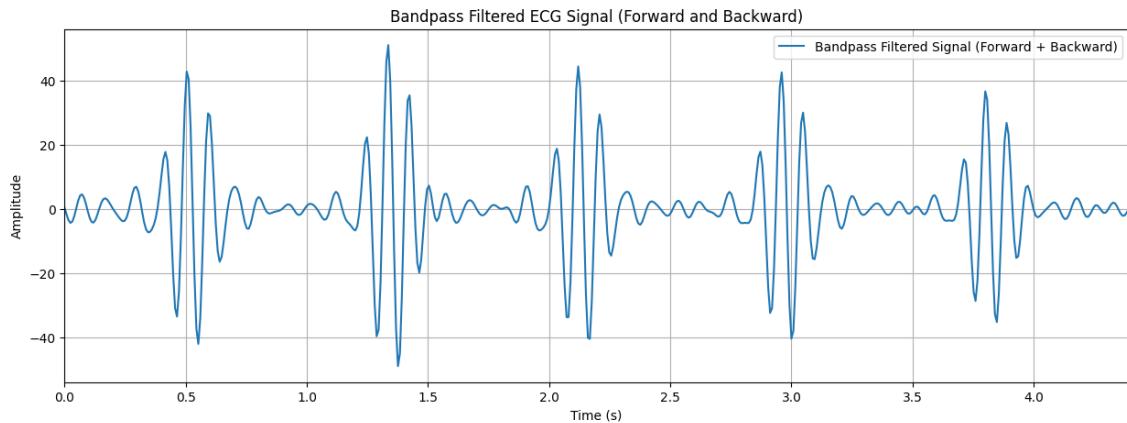
```

Program 7. Bandpass Filtering

Pada program di atas yang merupakan *bandpass filter* (BPF) dimana bertujuan untuk menemukan segmen qrs sehingga bisa dilanjutkan untuk pencarian tahap selanjutnya, dimana untuk BPF menggunakan *frequency cut-off low and high* yang bertujuan untuk memotong dan membiarkan frekuensi tertentu untuk QRS, adapun hasilnya sebagai berikut :



Gambar 24. Bandpass Filtering Sinyal Levy



Gambar 25. Bandpass Filtering Sinyal Davis

Dari data di atas dapat kita lihat dengan orde nya 20, dapat ditemukan untuk hasilnya yaitu sinyal QRS yang lebih terolah sehingga kita tinggal melanjutkan ke tahap selanjutnya untuk analisis selanjutnya.

3.1.8. Absolute

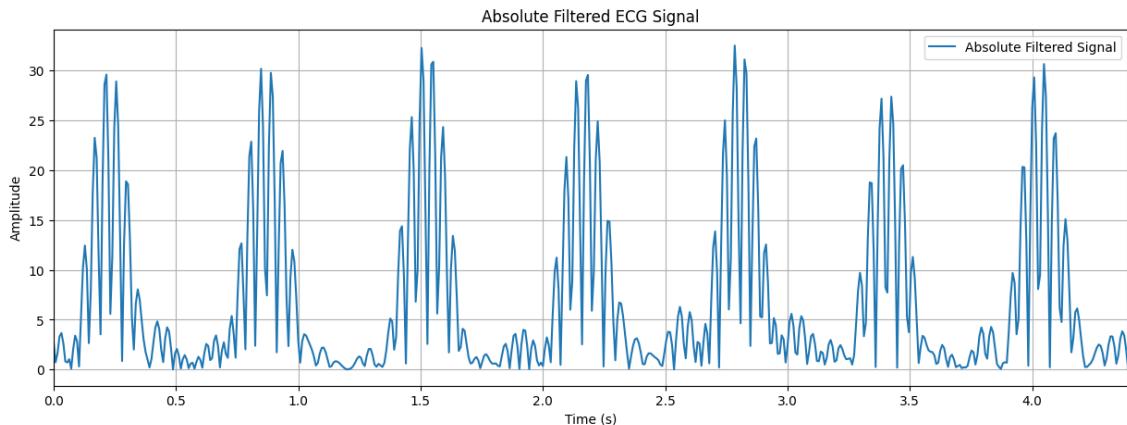
```
# Absolute

filtered_bandpass_ABS = np.abs(filtered_bandpass)

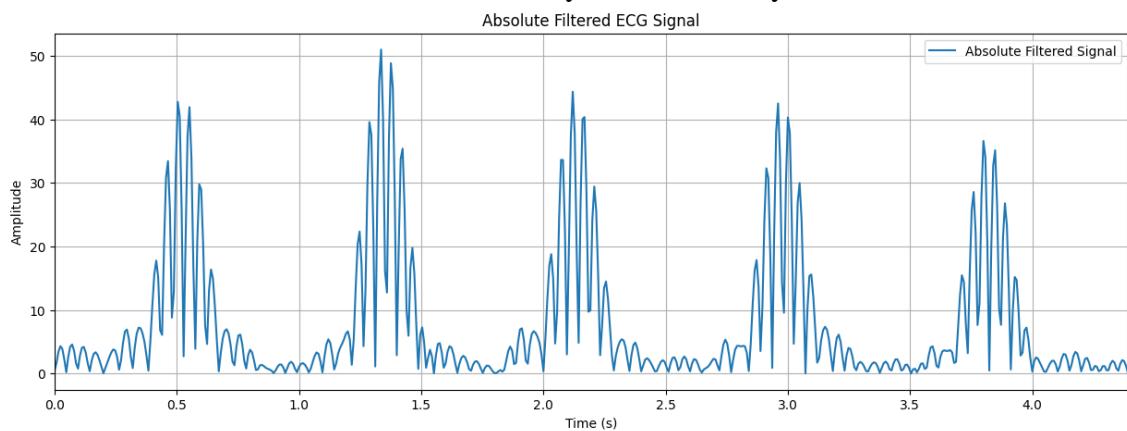
plt.figure(figsize=(15, 5))
plt.plot(waktu_sinyal, filtered_bandpass_ABS, label='Absolute
Filtered Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Absolute Filtered ECG Signal')
plt.legend()
plt.grid(True)
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.show()
```

Program 7. Absoulte Calculation

Pada program ini merupakan program yang berfungsi untuk meng absolut kan nilai dari BPF sehingga dapat dilihat dan dianalisis sinyalnya untuk tahap selanjutnya, dimana absolut ini merupakan salah satu fungsi dari numpy sehingga bisa langsung dieksekusi dengan hasil program sebagai berikut :



Gambar 26. Sinyal Absolut Levy



Gambar 27. Sinyal Absolut Davis

Dari hasil data di atas kita dapat menemukan nilai absolut sehingga bisa dianalisis dan diolah untuk tahap berikutnya.

3.1.9. Moving Average 2 (MAV 2)

```
# Moving Average (MAV) Parameters
```

```
M_MAV = 20
N = len(filtered_bandpass_ABS) # Jumlah data points
df_MAVERIK_abs = np.zeros(N) # Inisiasi array untuk menyimpan MAV
filtered signal

# Moving Average Filtering
for idx in range(N):
    for shift in range(M_MAV):
        if idx - shift >= 0:
            df_MAVERIK_abs[idx] += filtered_bandpass_ABS[idx - shift]
    df_MAVERIK_abs[idx] /= M_MAV

# Inisiasi hasil array moving average
N = len(df_MAVERIK_abs)
df_MAVERIK_backward_MAVERIK = np.zeros(N)
```

```

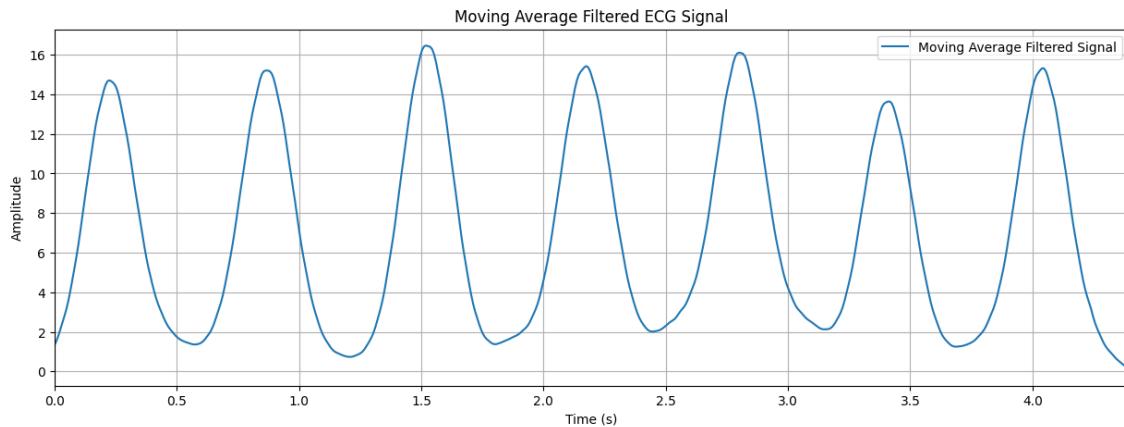
# Moving Average Filter Backward
for idx in range(N):
    for shift in range(M_MAV):
        if idx + shift < N:
            df_MAV_backward_MAV[idx] += df_MAV_abs [idx + shift]
df_MAV_backward_MAV[idx] /= M_MAV

# Plot Moving Average filtered signal
plt.figure(figsize=(15, 5))
plt.plot(waktu_sinyal, df_MAV_backward_MAV, label='Moving Average
Filtered Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Moving Average Filtered ECG Signal')
plt.legend()
plt.grid(True)
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.show()

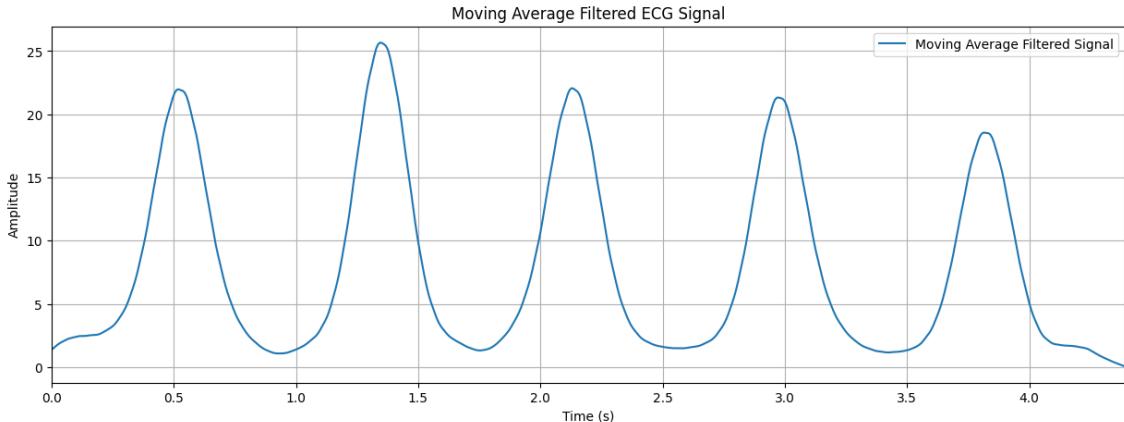
```

Program 8. Moving Average 2 Filtering

Pada program MAV 2 ini bertujuan untuk memfilter dari bandpass yang telah diabsolut sebelumnya sehingga bisa dilihat bentuknya lebih mudah untuk dianalisis dan mempermudah perhitungan tahap selanjutnya yaitu *thresholding*, adapun hasil sinyal MAV 2 ini sebagai berikut :



Gambar 28. Hasil MAV 2 Sinyal Levy



Gambar 29. Hasil MAV 2 Sinyal Davis

Pada hasil di atas dapat kita lihat bahwa bentukan dari MAV ini lebih jelas dan bagus sehingga bisa dianalisis untuk tahapan thresshold selanjutnya.

3.1.10. Threeshold Sinyal

```
# Thresshold
```

```
# Temukan threshold untuk MAV
max_value = df_MAV_backward_MA.max()
threshold = max_value * 0.5 # Disini saya memakai : 50% dari nilai maksimal
print(f"Max value is: {max_value}")

# Buat bentuk sinyal binary dari hasil threshold
sinyal_kotak = np.where(df_MAV_backward_MA > threshold, 1, 0)

# Plot binary signal yang ditemukan
plt.figure(figsize=(16, 5))
plt.plot(waktu_sinyal, df_MAV_backward_MA, label='Moving Average Filtered Signal')
plt.plot(waktu_sinyal, sinyal_kotak * max_value, label='Binary Signal', color='red', linewidth=1.5)
plt.axhline(y=threshold, color='green', label='Threshold')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude / Binary')
plt.title('Thresholded Binary Signal')
plt.legend()
plt.grid(True, which='both', linewidth=0.5)
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.tight_layout()
plt.show()

plt.figure(figsize=(16, 5))
plt.plot(waktu_sinyal, df_MAV_backward, label='Moving Average Filtered Signal')
```

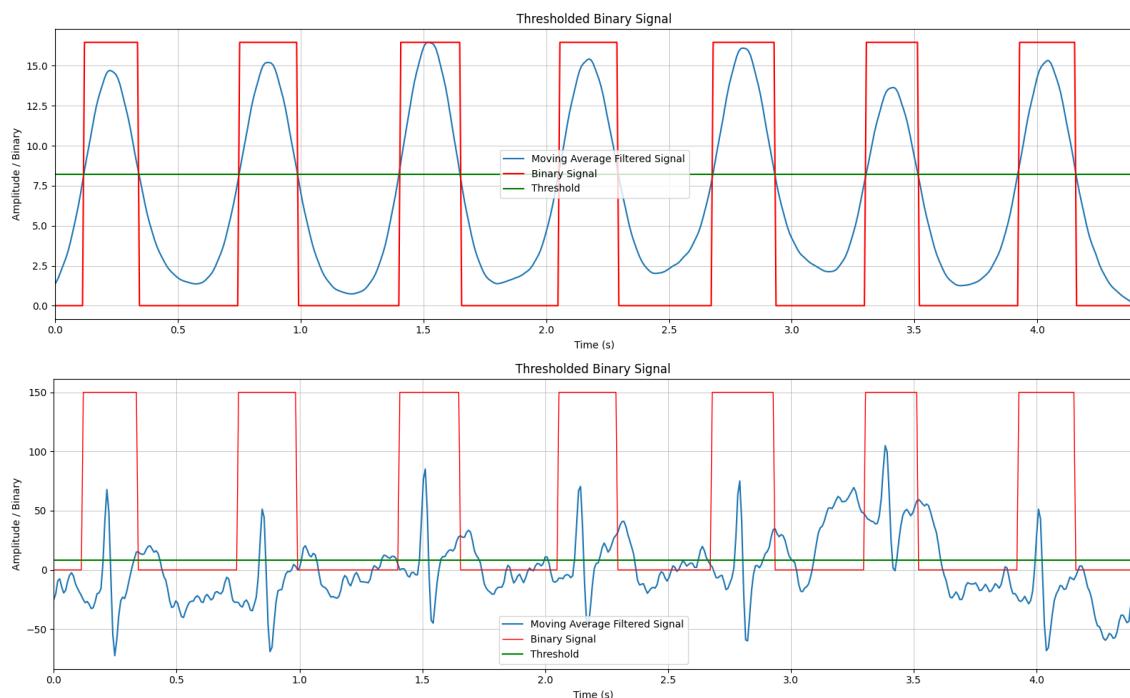
```

plt.plot(waktu_sinyal, sinyal_kotak * 150, label='Binary Signal',
color='red', linewidth=1.0)
plt.axhline(y=threshold, color='green', label='Threshold')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude / Binary')
plt.title('Thresholded Binary Signal')
plt.legend()
plt.grid(True, which='both', linewidth=0.5)
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.tight_layout()
plt.show()

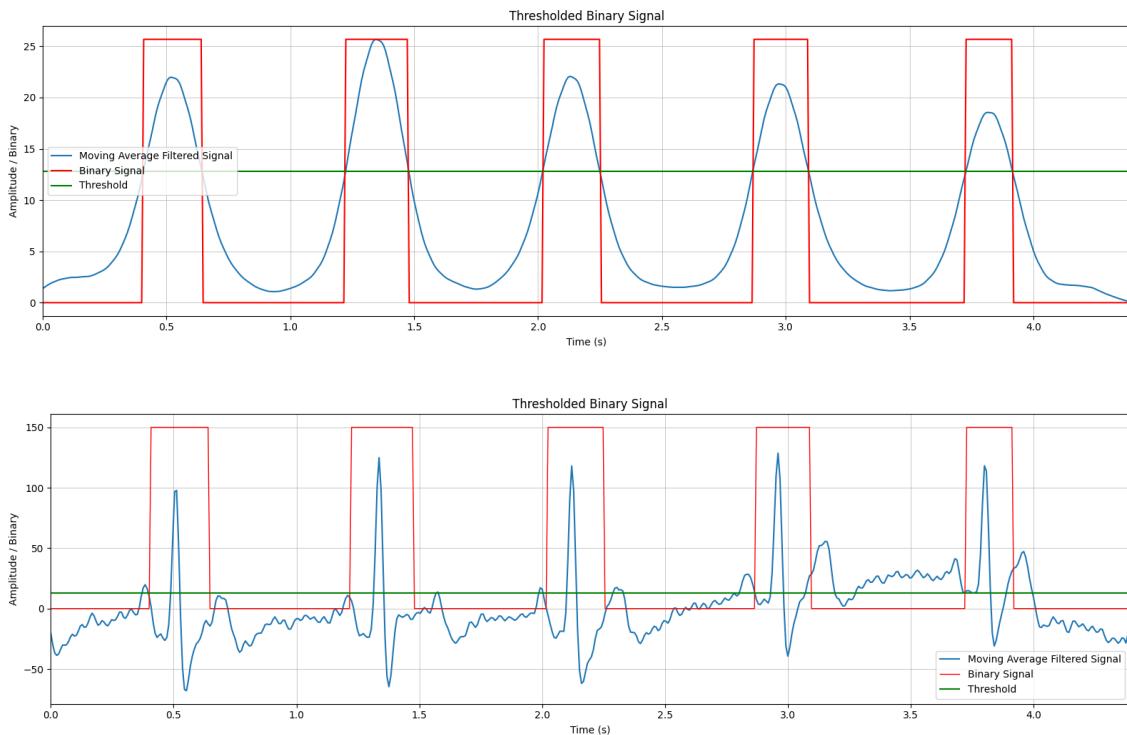
```

Program 9. Thresholding

Pada program di atas berfungsi untuk men *threshold* atau membatasi sinyal MAV2 sehingga merubah dari bentuk sinyal yang masih analog menjadi sinyal digital yang dibentuk dalam bentuk sinyal kotak, dimana bisa dirubah menjadi anatara bernilai 1 atau 0 sehingga bisa dianalisis secara komputasi dan mendapatkan data yang lebih akurat untuk analisa terutama dalam menemukan *Heart Rate* dan *BPM*, adapun hasilnya sebagai berikut :



Gambar 30. Thressholding Sinyal Levy



Gambar 31. Threshholding Sinyal Davis

Pada sinyal yang didapatkan di atas yang merupakan threshholding sehingga bisa ditemukan peak R dari tiap QRS dan selanjutnya bisa di analisis untuk menjadi BPM nya dan menemukan BPM dari masing-masing subjek ECG.

3.1.11. Interval dan BPM

INTERVAL

```
# Buat Kerangka Data dari sinyal_kotak biar lebih mudah
df = pd.DataFrame({'sinyal_kotak': sinyal_kotak})

# Identifikasi Puncak R dari sinyal_kotak
df_flags = []
df_RR = []

# Pastikan index ketika - 1 tidak keluar dari batas
for index in range(1, len(df['sinyal_kotak'])):
    if df['sinyal_kotak'][index] == 1 and df['sinyal_kotak'][index - 1] == 0:
        df_flags.append(index / fs)

# Membuat parameter dimana minimal ada 2 puncak sehingga bisa
# menghitung RR Intervalnya
if len(df_flags) > 1:
    for index in range(1, len(df_flags)):
        df_RR.append(df_flags[index] - df_flags[index - 1])
else:
```

```

        print(" R peaks yang terdeteksi tidak cukup untuk melakukan
perhitungan RR interval")

# Print RR intervals
print("RR Intervals:", df_RR)

# Hitung Mean Heart Rate nya jika terdapat RR Interval
if df_RR:
    mean_HR = 60 / (sum(df_RR) / len(df_RR))
    print(f"The average heart rate is: {mean_HR:.2f} BPM")
else:
    print("Tidak bisa menghitung BPM karena RR interval yang tidak
cukup")

```

Program 10. Interval Finding

Pada program di atas bertujuan untuk menemukan BPM dengan cara $BPM = 60 / R - R$ Peak, dimana untuk menemukannya kita menggunakan fungsi append yang berguna untuk memasukkan data baru ke array dan memperbaruiinya, sehingga dari sana ditemukan tiap Peak R yang jika terdapat lebih dari 1 Peak R, maka dapat dilakukan perhitungan interval antar Peak-Peak dari threshold sebelumnya, dimana hasilnya sebagai berikut :

```
RR Intervals: [0.632, 0.6559999999999999, 0.6480000000000001, 0.6240000000000001, 0.6239999999999997, 0.6240000000000001]
The average heart rate is: 94.54 BPM
```

Gambar 32. Interval BPM Sinyal Levy

```
RR Intervals: [0.8160000000000001, 0.8, 0.8479999999999999, 0.8560000000000003]
The average heart rate is: 72.29 BPM
```

Gambar 33. Interval BPM Sinyal Davis

Pada data tersebut dapat kita peroleh untuk hasil masing-masing subjek yaitu,

Levy :

Davis :

Di mana ditemukan untuk BPM Levy lebih tinggi daripada Davis, hal tersebut bisa dikarenakan beberapa aspek dan faktor namun yang bisa dilihat yaitu bahwa Levy lebih sehat dibandingkan Davis. Lalu, dari hasil tersebut bisa dilakukan analisis lannya dan dilanjutkan ke tahap-tahap yang lebih jauh kedepannya.

3.1.11. Segmentasi T (LPF)

```

# Parameter
fs = 125 # Frekuensi Sampling (Hz)
M = 25 # Order
fc = 5 # Cutoff frekuensi (Hz) untuk low-pass filter

# Hitung ohm_c cut-off frekuensinya

```

```

ohm_c = 2 * np.pi * fc / fs

# Hitung koefisien filter untuk low-pass filter
arr_M = np.arange(-M, M + 1, 1)
h = np.zeros(2 * M + 1)
for n in range(-M, M + 1, 1):
    if n == 0:
        h[n + M] = ohm_c / np.pi
    else:
        h[n + M] = np.sin(n * ohm_c) / (n * np.pi)

# Jumlah Point Data
N = len(df_MAV_backward)

# low-pass filter untuk (Forward Filtering)
x_filtered_forward_t = np.zeros(N)

# Forward Filtering
for i in range(N):
    for j in range(-M, M + 1, 1):
        if 0 <= i - j < N: # Pastikan index berada pada batasnya
            x_filtered_forward_t[i] += h[j + M] * df_MAV_backward[i - j]

# low-pass filter untuk (Backward Filtering)
x_filtered_backward_t = np.zeros(N)

# Backward Filtering
for i in range(N - 1, -1, -1):
    for j in range(-M, M + 1, 1):
        if 0 <= i + j < N: # Pastikan index berada pada batasnya
            x_filtered_backward_t[i] += h[j + M] *
x_filtered_forward_t[i + j]

# Plot forward and backward filtered ECG signal pada 1 plot yang sama
plt.figure(figsize=(15, 5))

plt.plot(waktu_sinyal, df_MAV_backward, label='Restored Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Restored ECG Signal')
plt.legend()
plt.grid(True)

plt.plot(waktu_sinyal, x_filtered_forward_t, label='Forward Filtered T Signal')

```

```

plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Forward Filtered ECG Signal')
plt.legend()
plt.grid(True)

plt.plot(waktu_sinyal, x_filtered_backward_t, label='Forward-
Backward Filtered T Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Combined Filtered ECG Signal')
plt.legend()
plt.grid(True)

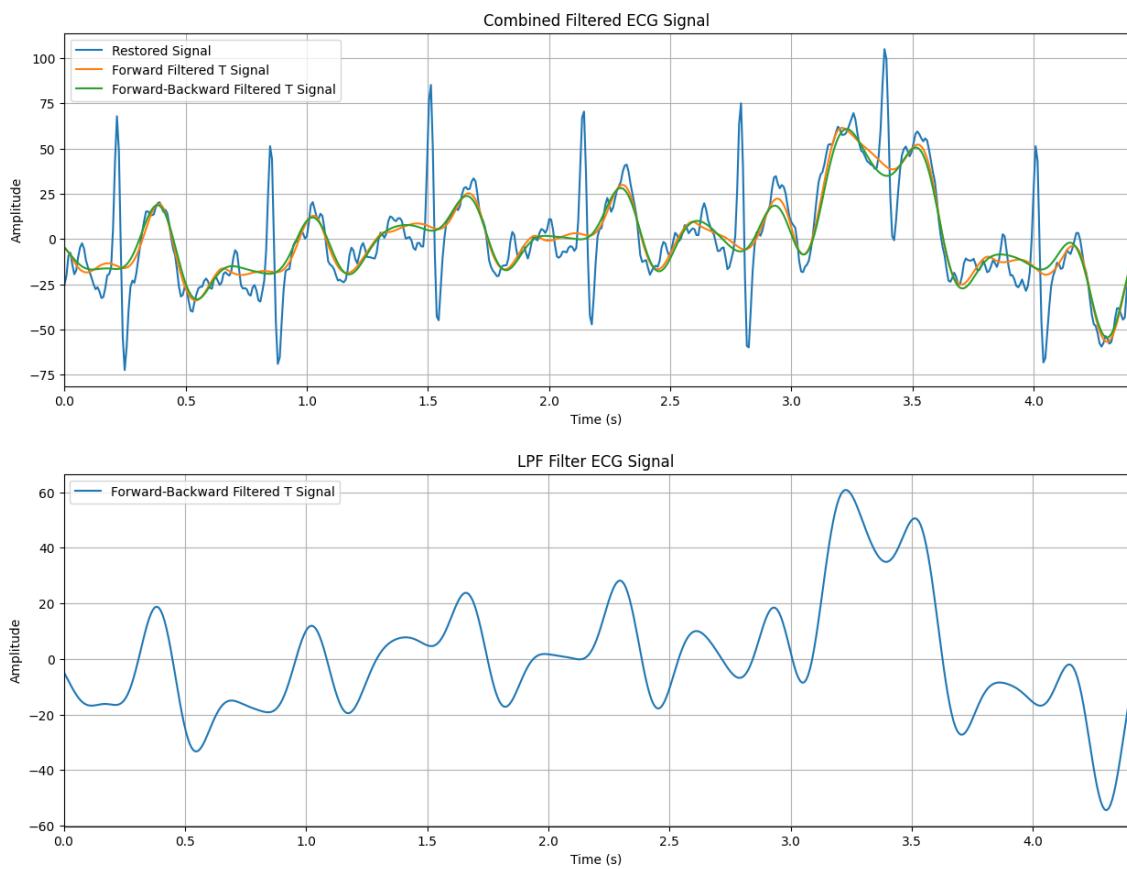
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.show()

# Plot forward and backward filtered ECG signal nya
plt.figure(figsize=(15, 5))
plt.plot(waktu_sinyal, x_filtered_backward_t, label='Forward-
Backward Filtered T Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('LPF Filter ECG Signal')
plt.legend()
plt.grid(True)
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.show()

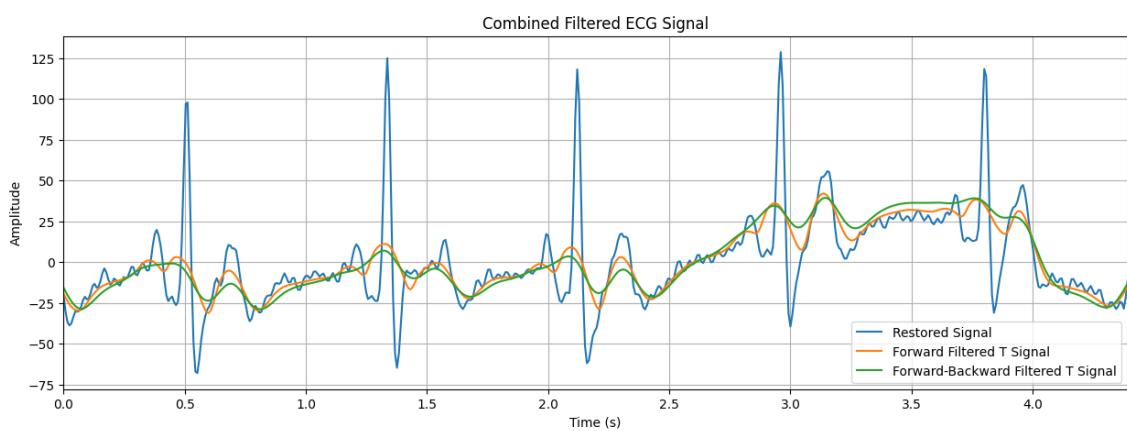
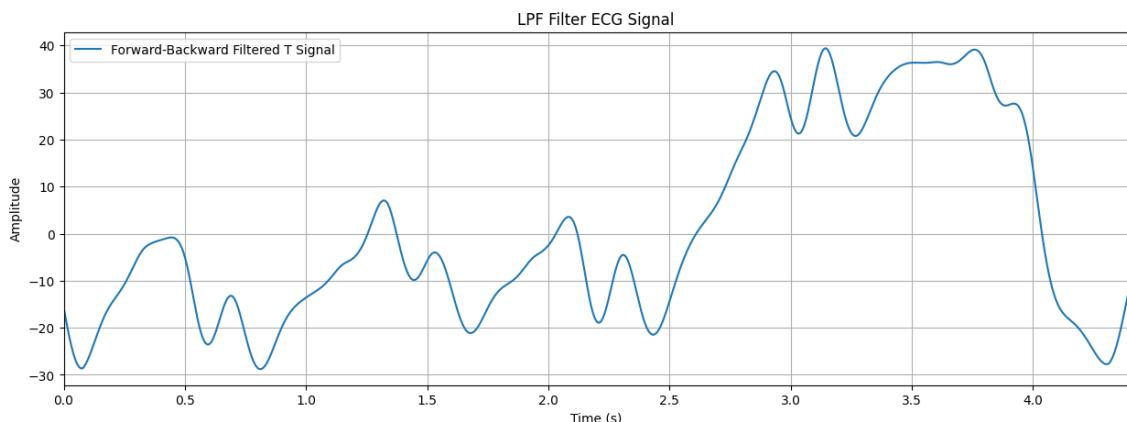
```

Program 11. LPF Segmentasi T

Pada program di atas yaitu LPF untuk segmentasi T dimana ketika dicari maka kita harus tau dimana untuk memotong frekuensi lainnya sehingga harapannya, hanya sinyal T yang nanti akan terlewat untuk bisa di filtering dan menemukan peak dari tiap-tiap puncak T. Di mana pada kasus ini ditemukan untuk menggunakan orde untuk levy sepanjang 25 dan davis sepanjang 10, untuk frekuensi cut-off yaitu di 5 Hz sehingga pada frekuensi tersebut bisa membiarkan sinyal T untuk lewat, adapun hasil outputnya sebagai berikut :



Gambar 34. LPF Segmentasi T Sinyal Levy



Gambar 35. LPF Segmentasi T Sinyal Davis

Pada hasil di atas dapat kita lihat setelah di LPF, Peak T dari masing-masing sinyal mulai menonjol dan lebih jelas dibandingkan P dan QRS nya, namun untuk davis pada program saya untuk sinyalnya masih belum jelas.

3.1.12. Absolut Segmentasi T

```
# Absolute

filtered_lpf2_ABS = np.abs(x_filtered_backward_t)

plt.figure(figsize=(15, 5))
plt.plot(waktu_sinyal, filtered_lpf2_ABS, label='Absolute Filtered
Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Absolute Filtered ECG Signal')
plt.legend()
plt.grid(True)
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.show()

#Hanya Memunculkan Sinyal T

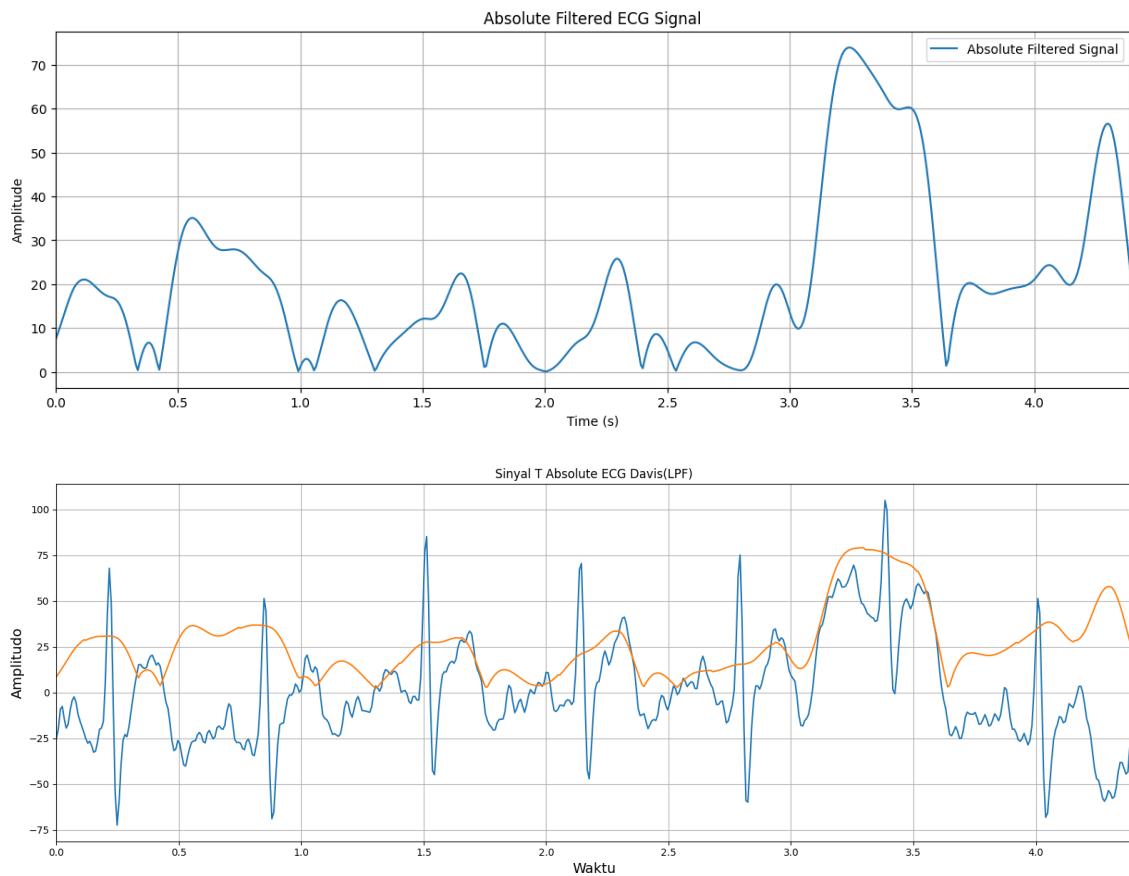
abs_lowpass_T = filtered_lpf2_ABS - (sinyal_kotak -
df_MAV_backward_MAV)

# Plot the filtered signal
plt.figure(figsize=(16, 6))

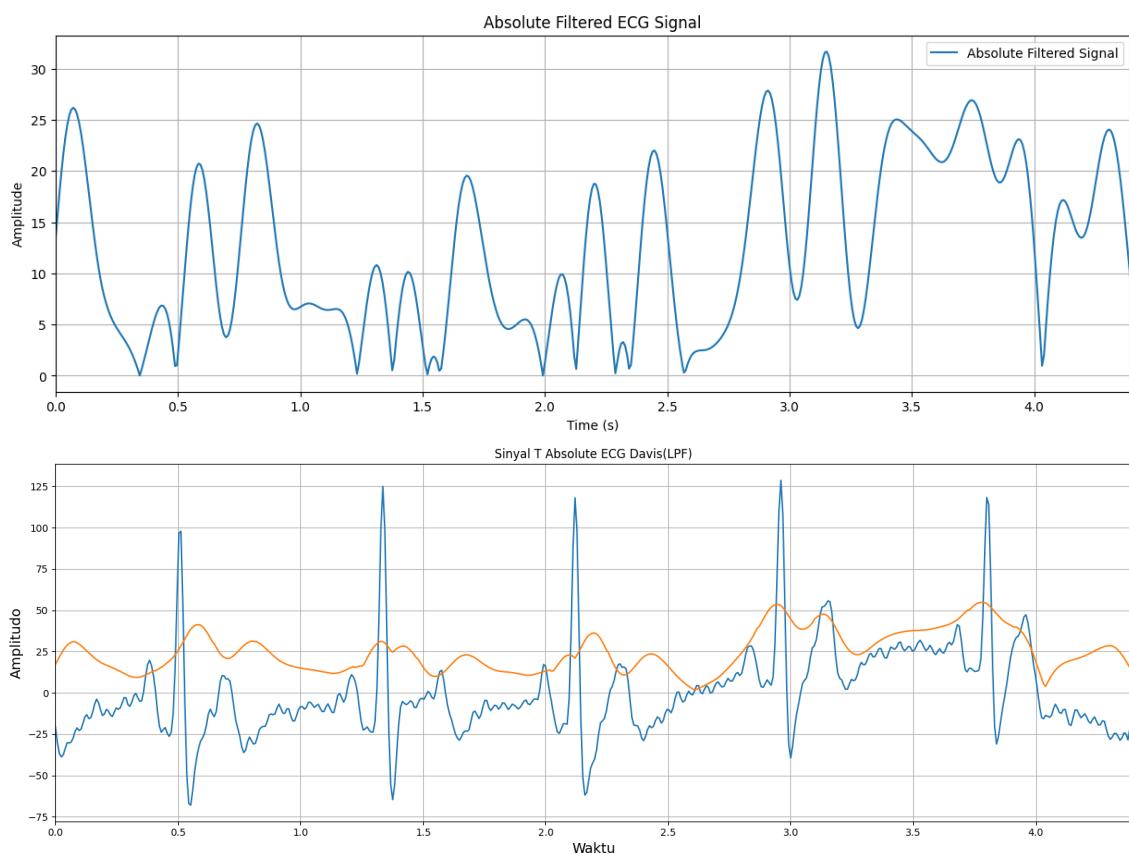
plt.plot(waktu_sinyal, df_MAV_backward)
plt.plot(waktu_sinyal,abs_lowpass_T)
plt.xlabel('Waktu', fontsize=14)
plt.ylabel('Amplitudo', fontsize=14)
plt.title('Sinyal T Absolute ECG Davis(LPF)')
plt.grid()
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.tight_layout()
plt.show()
```

Program 12. Absolut Segemen T

Pada program di atas memiliki tujuan untuk men absolut nilai dari LPF sebelumnya, dengan hasil sebagai berikut :



Gambar 36. Absolut Segmentasi T Sinyal Davis



Gambar 37. Absolut Segementasi T Sinyal Davis

Pada hasil di atas dapat dilihat bahwa dari sinyal absolut tersebut sudah terlihat untuk peak dari nilai yang dicari, namun pada program ini saya sendiri masih terdapat kesalahan sehingga untuk peak yang muncul setelah absolut tidak hanya segmen T saja namun juga terdapat yang lain.

3.1.13. Threshold Segmentasi T

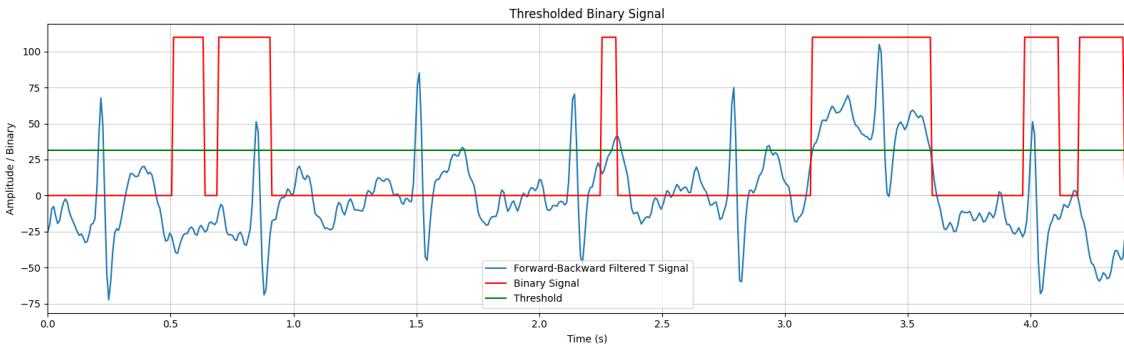
```
# Calculate the maximum value of the filtered T signal
max_value_t = abs_lowpass_T.max()
threshold = max_value_t * 0.4
print(f"Max value is: {max_value_t}")

# Buat Binary nya secara manual berdasarkan threshold
sinyal_kotak_t = np.zeros_like(abs_lowpass_T)
for i in range(len(abs_lowpass_T)):
    if abs_lowpass_T[i] > threshold:
        sinyal_kotak_t[i] = 1
    elif abs_lowpass_T[i] < threshold:
        sinyal_kotak_t[i] = 0

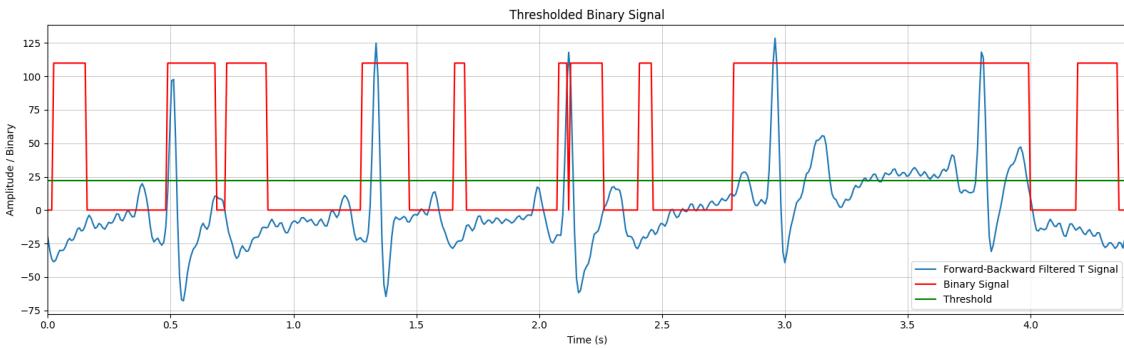
# Plot binary signal
plt.figure(figsize=(16, 5))
plt.plot(waktu_sinyal, df_MAV_backward, label='Forward-Backward
Filtered T Signal')
plt.plot(waktu_sinyal, sinyal_kotak_t * 110, label='Binary Signal',
color='red', linewidth=1.5)
plt.axhline(y=threshold, color='green', label='Threshold')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude / Binary')
plt.title('Thresholded Binary Signal')
plt.legend()
plt.grid(True, which='both', linewidth=0.5)
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.tight_layout()
plt.show()
```

Program 13. Thresholding T

Pada bagian ini sinyal yang di absolut kemudian di threshold sehingga bisa ditemukan peak nilai T nya dan dijadikan sinyal biner, adapun punya saya masih belum benar dan masih terdapat kesalahan sehingga hasilnya sebagai berikut :



Gambar 38. Hasil Thressholding T Sinyal Levy



Gambar 39. Hasil Thressholding T Sinyal Davis

Pada hasil tersebut dapat dilihat bahwa masih belum benar dan dapat dicari lagi untuk segmen T nya yang benar, hal tersebut bisa terjadi karena pada thressholding tidak hanya sinyal T yang muncul peak nya sehingga sinyal lain ikut terdeteksi dan masuk menjadi thresshold T, untuk perbaikannya bisa dengan mencoba programnya lebih jauh dan lebih lama lagi sehingga bisa menampilkan sinyal T yang benar.

3.1.14. Alternatif Segmentasi T

```
# Segmentasi Sinyal T (Alternatif)
```

```
xt = waktu_sinyal

segmenT_sinyal = []

segmenT_sinyal = np.zeros(len(waktu_sinyal))

#LEVY
for i in range(len(waktu_sinyal)):
    if xt[i]>=0.28 and xt[i]<=0.5:
        segmenT_sinyal[i]=df_MAV_backward[i]
    elif xt[i]>=0.9 and xt[i]<=1.1:
        segmenT_sinyal[i]=df_MAV_backward[i]
    elif xt[i]>=1.58 and xt[i]<=1.75:
        segmenT_sinyal[i]=df_MAV_backward[i]
    elif xt[i]>=2.21 and xt[i]<=2.37:
        segmenT_sinyal[i]=df_MAV_backward[i]
```

```

    elif xt[i]>=2.87 and xt[i]<=3:
        segmenT_sinyal[i]=df_MAV_backward[i]
    elif xt[i]>=3.45 and xt[i]<=3.6:
        segmenT_sinyal[i]=df_MAV_backward[i]
    elif xt[i]>=4.05 and xt[i]<=4.26:
        segmenT_sinyal[i]=df_MAV_backward[i]
    else:
        segmenT_sinyal[i]=0

#DAVIS
for i in range(len(waktu_sinyal)):
    if xt[i]>=0.6 and xt[i]<=0.76:
        segmenT_sinyal[i]=df_MAV_backward[i]
    elif xt[i]>=1.54 and xt[i]<=1.60:
        segmenT_sinyal[i]=df_MAV_backward[i]
    elif xt[i]>=2.21 and xt[i]<=2.37:
        segmenT_sinyal[i]=df_MAV_backward[i]
    elif xt[i]>=3.10 and xt[i]<=3.18:
        segmenT_sinyal[i]=df_MAV_backward[i]
    elif xt[i]>=3.87 and xt[i]<=4.0:
        segmenT_sinyal[i]=df_MAV_backward[i]
    else:
        segmenT_sinyal[i]=0

plt.figure(figsize=(20, 5))
plt.plot(waktu_sinyal, df_MAV_backward)
plt.plot(waktu_sinyal, segmenT_sinyal)
plt.title('Plot Sinyal T')
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.grid(True)
plt.show()

# Absolut Sinyal T
absolutT_sinyal = np.zeros(len(segmenT_sinyal))
for i in range(len(absolutT_sinyal)):
    absolutT_sinyal[i] = segmenT_sinyal[i] * segmenT_sinyal[i]

plt.figure(figsize=(20, 5))
plt.plot(waktu_sinyal, absolutT_sinyal)
plt.title('Absolute Sinyal T')
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.grid(True)
plt.show()

# Thresshold Sinyal T nya
thresholdT_sinyal = np.zeros(np.size(absolutT_sinyal))

```

```

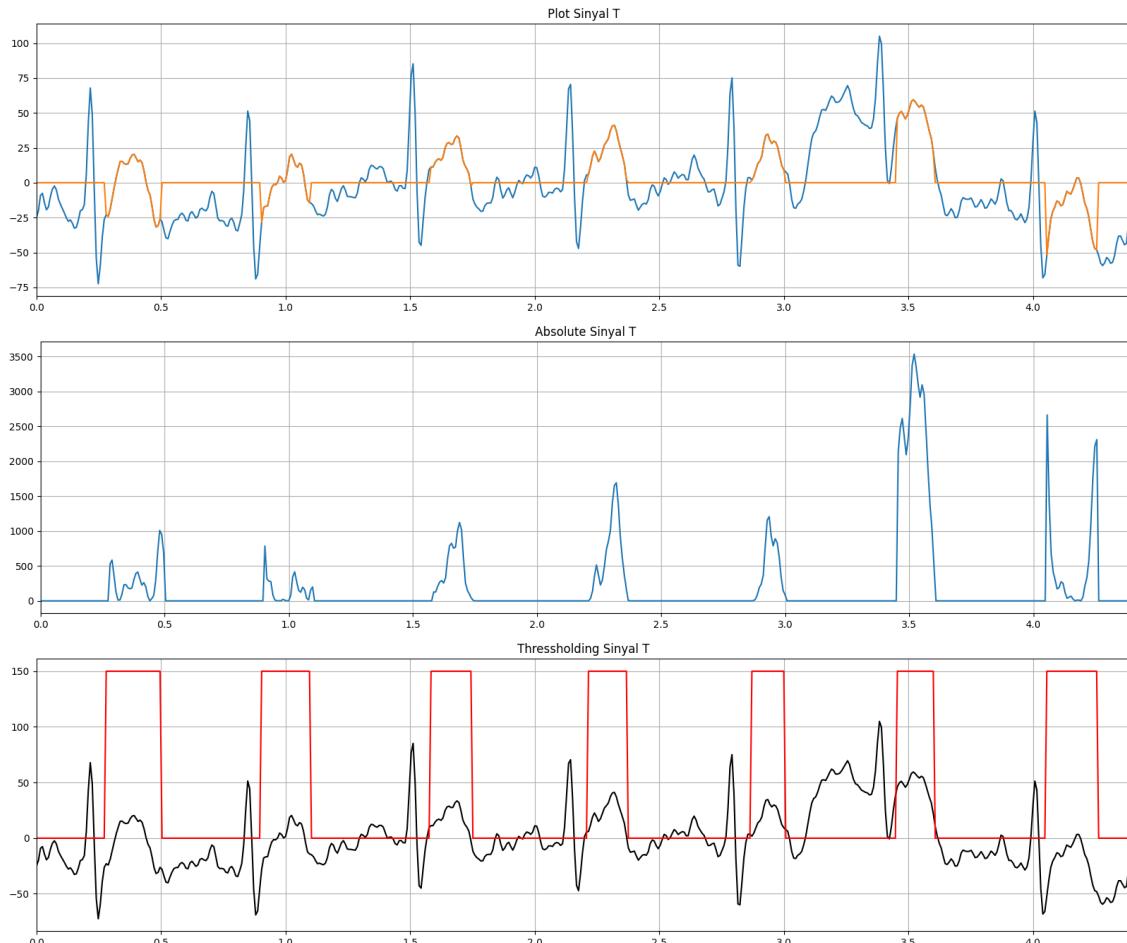
for i in range(len(thresholdT_sinyal)):
    if absolutT_sinyal[i] > 0:
        thresholdT_sinyal[i] = 150
    else:
        thresholdT_sinyal[i] = 0

# Plot Sinyal T
plt.figure(figsize=(20,5))
plt.plot(waktu_sinyal, df_MAV_backward, color="black")
plt.plot(waktu_sinyal, thresholdT_sinyal, color="red")
plt.title('Thressholding Sinyal T')
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.grid(True)
plt.show()

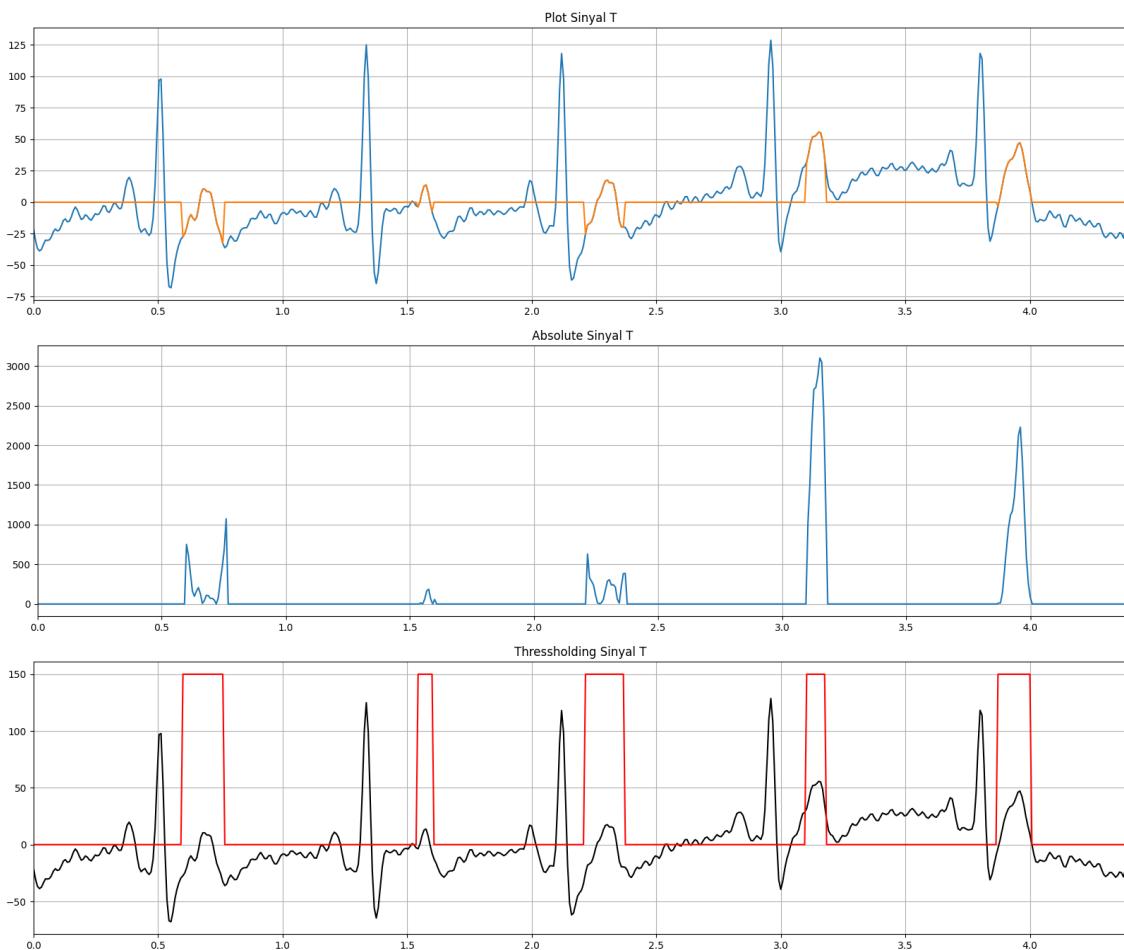
```

Program 14. Alternatif Segmentasi T

Pada program di atas yang di mana saya mempelajarinya dari beberapa teman saya untuk solusi alternatifnya, yaitu dengan cara manual sehingga untuk sinyal T pada data ini dimana tidak terlalu banyak data, masih bisa kita plot secara manual, sedangkan cara ini tidak efektif untuk jumlah data yang banyak. Adapun hasilnya sebagai berikut :



Gambar 40. Hasil Segmentasi T Alternatif Sinyal Levy

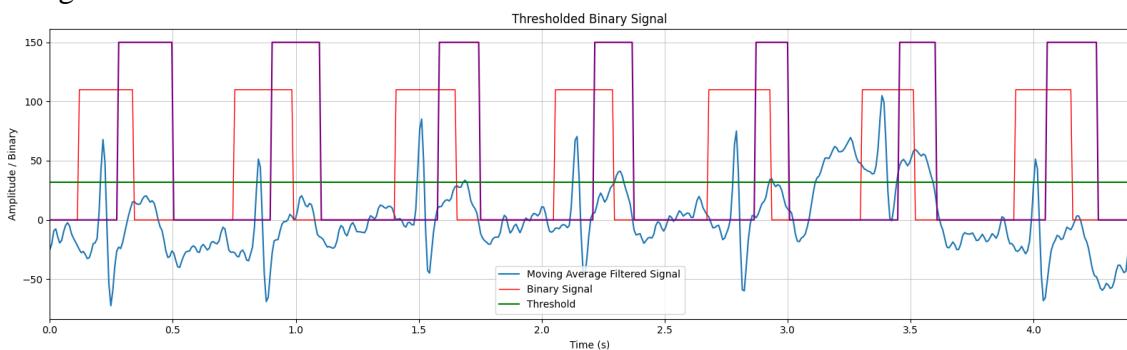


Gambar 41. Hasil Segmentasi T Sinyal Davis

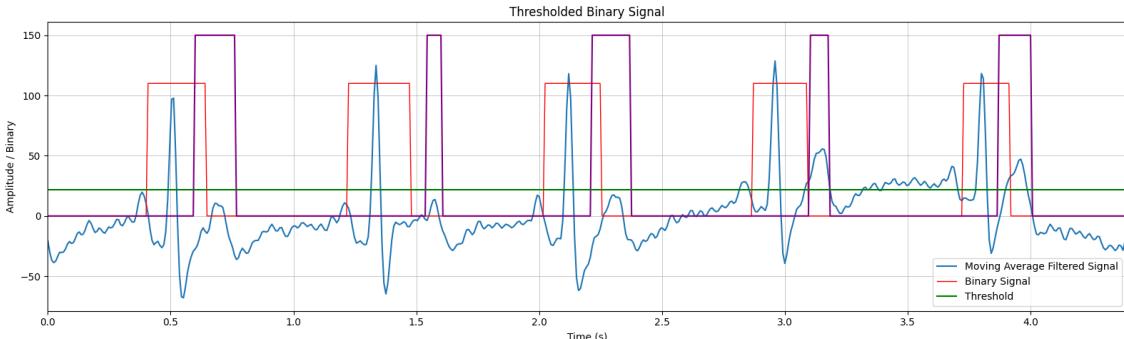
Pada hasil di atas dapat dilihat bahwa dengan cara manual dapat menentukan segmen T dengan lebih baik, namun cara ini akan sulit jika data sinyalnya ada banyak dan ini hanya merupakan solusi alternatif untuk menentukan T Segment saja.

3.1.15. Hasil Segmentasi QRS dan T

Pada hasil segmentasi QRS dan T ditemukan threshold yang membatasi keduanya, dimana ketika digabungkan akan memperlihatkan batasan-batasan mereka masing-masing, adapun hasil yang rapat yaitu disebabkan untuk threshold qrs dengan cara biasa namun untuk threshold t secara manual hitungnya sehingga untuk sinyal berdekatan nya sedikit rapat adapun program nya yaitu hanya di plot biasa, dengan output segmentasi sebagai berikut :



Gambar 42. Hasil Segmentasi QRS dan T Sinyal Levy



Gambar 43. Hasil Segmentasi QRS dan T Sinyal Davis

Pada hasil segmentasi tersebut bisa dilihat untuk masing-masing daerah nya dan hasil yang ditampilkan memang belum sempurna namun bisa dikembangkan lagi dan diperbaus lagi kedepannya.

3.2. Study Case Final Project

```
# BPF dengan Fungsi Windowing
# Tentukan Parameternya
fs = 125 # Sampling frequency (Hz)
fcl = 8   # Lower cutoff frequency (Hz)
fch = 15  # Higher cutoff frequency (Hz)
M_BP = 25 # Order of the Bandpass filter

# Definisikan Blackman Window nya
def blackman_window(M):
    w_black = np.zeros(2 * M + 1)
    for i in range(-M, M + 1):
        w_black[i + M] = 0.42 + 0.5 * np.cos(i * np.pi / M) + 0.08
    * np.cos(2 * i * np.pi / M)
    return w_black

# Definisi dari koefisien bandpass filter
def bandpass_filter_coeff(fcl, fch, fs, M_BP):
    h_bandpass = np.zeros(2 * M_BP + 1)
    for n in range(-M_BP, M_BP + 1):
        if n == 0:
            h_bandpass[n + M_BP] = (2 * np.pi * fch / fs) / np.pi -
            (2 * np.pi * fcl / fs) / np.pi
        else:
            h_bandpass[n + M_BP] = (np.sin(2 * np.pi * fch * n / fs) / (np.pi * n)) - (np.sin(2 * np.pi * fcl * n / fs) / (np.pi * n))
    return h_bandpass

# Hitung Koefisien bandpass filter
```

```

h_bandpass = bandpass_filter_coeff(fcl, fch, fs, M_BP)

# Masukkan Blackman Window ke dalam koefisien BPF
w_black = blackman_window(M_BP)
h_bandpass_windowed = h_bandpass * w_black

# Forward Filtering
N = len(df_MAV_backward)
x_filtered_forward = np.zeros(N)
for i in range(N):
    for j in range(-M_BP, M_BP + 1):
        if 0 <= i - j < N:
            x_filtered_forward[i] += h_bandpass_windowed[j + M_BP]
            * df_MAV_backward[i - j]

# Backward Filtering
x_filtered_backward_BPF = np.zeros(N)
for i in range(N-1, -1, -1):
    for j in range(-M_BP, M_BP + 1):
        if 0 <= i + j < N:
            x_filtered_backward_BPF[i] += h_bandpass_windowed[j +
M_BP] * x_filtered_forward[i + j]

filtered_bandpass = x_filtered_backward_BPF

# Plot bandpass terfilter
plt.figure(figsize=(15, 5))
plt.plot(waktu_sinyal, x_filtered_backward_BPF, label='Bandpass
Filtered Signal (Forward + Backward)')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Bandpass Filtered ECG Signal (Forward and Backward)')
plt.legend()
plt.grid(True)
plt.xlim(waktu_sinyal.min(), waktu_sinyal.max())
plt.show()

# DFT Sinyal Asli

# Fungsi dari Discrete Fourier Transform (DFT)
def DFT(x):
    N = len(x)
    X_real = np.zeros(N)
    X_imag = np.zeros(N)

    for k in range(N):
        for n in range(N):

```

```

        X_real[k] += x[n] * np.cos(2 * np.pi * k * n / N)
        X_imag[k] -= x[n] * np.sin(2 * np.pi * k * n / N)

mag = np.sqrt(X_real**2 + X_imag**2)

return mag

# Hitung DFT dari sinyal yang telah direstorasi
magDFT = DFT(h_bandpass_windowed)

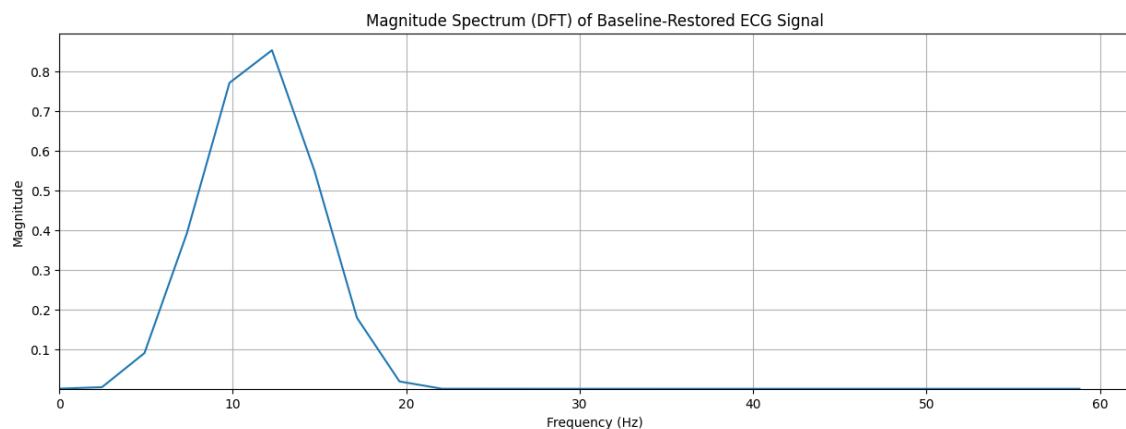
# Plot magnitude spectrum
freq = np.arange(len(magDFT)) * fs / len(magDFT)
freq_half = freq[:len(magDFT)//2] # Half of the frequency range

plt.figure(figsize=(15, 5))
plt.plot(freq_half, magDFT[:len(magDFT)//2])
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude')
plt.xlim(waktu_sinyal.min())
plt.ylim(magDFT[:len(magDFT)//2].min())
plt.title('Magnitude Spectrum (DFT) of Baseline-Restored ECG Signal')
plt.grid(True)
plt.show()

```

Program 15. BPF dengan Blackman Windowing

Pada program di atas dapat dilihat bahwa pada BPF biasa setelah diberikan study case ketika final project kemarin, terdapat penambahan untuk fungsi windowing dimana untuk yang digunakan yaitu blackman windowing. Lalu langsung dapat didefinisikan saja sesuai dengan rumus yang di PPT dan setelah itu hasilnya di DFT kan sehingga didapatkan bentuk seperti ini :



Gambar 44. Hasil Blackman Window di BPF

Dari hasil tersebut didapatkan bentuk berupa seperti pada PPT yang diberikan untuk Blackman Windowing.

BAB IV

KESIMPULAN

Pada Final Project Mata Kuliah Dasar Pengolahan Sinyal, telah dilakukan projek dimana memiliki tujuan untuk mengolah ECG mentah yang diberikan untuk dilakukan proses pengolahan secara digital. Setelah menyelesaikan proyek akhir ini terutama bagian QRS dan T serta perhitungan heart rate, ada beberapa hal yang bisa saya simpulkan. Sinyal ECG memiliki beberapa bagian penting, yaitu P, QRS, dan T. Dengan menggunakan metode seperti DFT, filter lowpass, filter bandpass, dan filter moving average, saya bisa mendeteksi bagian-bagian ini secara terpisah sehingga bisa memahami lebih baik setiap bagiannya.

Namun, saya masih mengalami kesulitan dalam mendeteksi bagian T dari sinyal ECG. Kesalahan ini terjadi karena hasil filter yang kurang optimal, sehingga thresholding tidak bekerja dengan baik. Meskipun program yang digunakan cukup untuk memisahkan komponen-komponen sinyal, masih ada yang perlu diperbaiki agar deteksi bagian T lebih baik dan bisa sesuai pada tempatnya.

Mata kuliah Dasar Pengolahan Sinyal mengajarkan pentingnya pengolahan sinyal. Mata kuliah ini memberikan dasar yang kuat dalam pengolahan sinyal, yang penting untuk pengembangan teknologi kesehatan dan aplikasi lainnya. Saya mendapatkan pengetahuan teoritis dan keterampilan praktis dalam menggunakan teknik pengolahan sinyal untuk menyelesaikan masalah nyata. Pengalaman ini harapannya akan menjadi dasar yang baik untuk lanjutan dan pengembangan di masa depan.

DAFTAR PUSTAKA

- Jevon, P. (2010) *Understanding the ECG: A Practical Approach*. Oxford: Wiley-Blackwell.
- Karris, S.T. (2007) *Signals and Systems with MATLAB Computing and Simulink Modeling*. Orchard Publications.
- Lippincott Williams & Wilkins. (2015) *ECG Interpretation Made Incredibly Easy*. 6th edn. Philadelphia: Lippincott Williams & Wilkins.
- Oppenheim, A.V. & Schafer, R.W. (2010) *Discrete-Time Signal Processing*. 3rd edn. Upper Saddle River, NJ: Prentice Hall.
- Proakis, J.G. & Manolakis, D.G. (2007) *Digital Signal Processing: Principles, Algorithms, and Applications*. 4th edn. Upper Saddle River, NJ: Prentice Hall.
- Smith, S.W. (2003) *Digital Signal Processing: A Practical Guide for Engineers and Scientists*. Newnes.