LAB Questions-

1. **QRS detection**

**MATLAB**:

clc; clear; close all;

data = load('115m (15).mat');

ecg = data.val(:);

fs = 360;

fc\_bandpass = [0.5, 40];

order\_bandpass = 2;

[b\_bandpass, a\_bandpass] = butter(order\_bandpass, fc\_bandpass / (fs / 2), 'bandpass');

ecg\_filtered = filtfilt(b\_bandpass, a\_bandpass, ecg);

[~, peaks] = findpeaks(ecg\_filtered, 'MinPeakHeight', 0.5, 'MinPeakDistance', round(0.6 \* fs));

RR\_intervals = diff(peaks) / fs;

heart\_rate = 60 / mean(RR\_intervals);

fprintf('Heart Rate: %.2f bpm\n', heart\_rate);

QRS\_window\_pre = round(0.06 \* fs);

QRS\_window\_post = round(0.06 \* fs);

figure;

first\_R\_peak = peaks(2);

start\_idx = max(first\_R\_peak - QRS\_window\_pre, 1);

end\_idx = min(first\_R\_peak + QRS\_window\_post, length(ecg\_filtered));

single\_QRS\_complex = ecg\_filtered(start\_idx:end\_idx);

time\_single = (start\_idx:end\_idx) / fs;

plot(time\_single, single\_QRS\_complex);

title('Single QRS Complex (One Cardiac Cycle)');

xlabel('Time (s)');

ylabel('Amplitude');

figure;

subplot(3, 1, 1);

plot((1:length(ecg)) / fs, ecg);

title('Original ECG Signal');

xlabel('Time (s)');

ylabel('Amplitude');

subplot(3, 1, 2);

plot((1:length(ecg\_filtered)) / fs, ecg\_filtered);

title('Filtered ECG Signal (Baseline Wandering and Noise Removed)');

xlabel('Time (s)');

ylabel('Amplitude');

subplot(3, 1, 3);

hold on;

for i = 1:length(peaks)

start\_idx = max(peaks(i) - QRS\_window\_pre, 1);

end\_idx = min(peaks(i) + QRS\_window\_post, length(ecg\_filtered));

QRS\_complex = ecg\_filtered(start\_idx:end\_idx);

plot((start\_idx:end\_idx) / fs, QRS\_complex);

end

title('Extracted QRS Complexes');

xlabel('Time (s)');

ylabel('Amplitude');

hold off;

**PYTHON**:

import numpy as np

import scipy.io

import scipy.signal as signal

import matplotlib.pyplot as plt

data = scipy.io.loadmat("/content/115m (15).mat")

ecg = data['val'].flatten()

fs = 360

fc\_bandpass = [0.5, 40]

order\_bandpass = 2

b\_bandpass, a\_bandpass = signal.butter(order\_bandpass, [fc / (fs / 2) for fc in fc\_bandpass], 'bandpass')

ecg\_filtered = signal.filtfilt(b\_bandpass, a\_bandpass, ecg)

peaks, \_ = signal.find\_peaks(ecg\_filtered, height=0.5, distance=round(0.6 \* fs))

RR\_intervals = np.diff(peaks) / fs

heart\_rate = 60 / np.mean(RR\_intervals)

QRS\_window\_pre = round(0.06 \* fs)

QRS\_window\_post = round(0.06 \* fs)

plt.figure()

first\_R\_peak = peaks[1]

start\_idx = max(first\_R\_peak - QRS\_window\_pre, 0)

end\_idx = min(first\_R\_peak + QRS\_window\_post, len(ecg\_filtered))

single\_QRS\_complex = ecg\_filtered[start\_idx:end\_idx]

time\_single = np.arange(start\_idx, end\_idx) / fs

plt.plot(time\_single, single\_QRS\_complex)

plt.title('Single QRS Complex (One Cardiac Cycle)')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.figure()

plt.subplot(3, 1, 1)

plt.plot(np.arange(len(ecg)) / fs, ecg)

plt.title('Original ECG Signal')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.subplot(3, 1, 2)

plt.plot(np.arange(len(ecg\_filtered)) / fs, ecg\_filtered)

plt.title('Filtered ECG Signal (Baseline Wandering and Noise Removed)')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.subplot(3, 1, 3)

for i in range(len(peaks)):

start\_idx = max(peaks[i] - QRS\_window\_pre, 0)

end\_idx = min(peaks[i] + QRS\_window\_post, len(ecg\_filtered))

QRS\_complex = ecg\_filtered[start\_idx:end\_idx]

plt.plot(np.arange(start\_idx, end\_idx) / fs, QRS\_complex)

plt.title('Extracted QRS Complexes')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.tight\_layout()

plt.show()

1. **R Peaks**

**MATLAB**:

clc;

clear;

close all;

data = load('115m (15).mat');

ecg = data.val(:); fs = 360;

fc\_bandpass = [0.5, 40];

order\_bandpass = 2;

[b\_bandpass, a\_bandpass] = butter(order\_bandpass, fc\_bandpass / (fs / 2), 'bandpass');

ecg\_filtered = filtfilt(b\_bandpass, a\_bandpass, ecg);

[pks, locs] = findpeaks(ecg\_filtered, 'MinPeakHeight', 0.5, 'MinPeakDistance', round(0.6 \* fs));

RR\_intervals = round(diff(locs) / fs, 2);

heart\_rate = 60 / mean(RR\_intervals

fprintf('R-R Intervals (s): %.2f\n', RR\_intervals);

fprintf('Heart Rate: %.2f bpm\n', heart\_rate);

% Plot ECG signal with detected R-peaks

figure;

plot((1:length(ecg\_filtered)) / fs, ecg\_filtered, 'b');

hold on;

plot(locs / fs, ecg\_filtered(locs), 'ro', 'MarkerSize', 8, 'LineWidth', 1.5);

title('ECG Signal with Detected R-Peaks');

xlabel('Time (s)');

ylabel('Amplitude');

legend('Filtered ECG Signal', 'R Points');

grid on;

**PYTHON**:

import numpy as np

import scipy.io

import scipy.signal as signal

import matplotlib.pyplot as plt

data = scipy.io.loadmat("/115m (15).mat")

ecg = data['val'].flatten()

fs = 360

fc\_bandpass = [0.5, 40]

order\_bandpass = 2

b\_bandpass, a\_bandpass = signal.butter(order\_bandpass, [fc / (fs / 2) for fc in fc\_bandpass], 'bandpass')

ecg\_filtered = signal.filtfilt(b\_bandpass, a\_bandpass, ecg)

peaks, \_ = signal.find\_peaks(ecg\_filtered, height=0.5, distance=round(0.6 \* fs))

RR\_intervals = np.round(np.diff(peaks) / fs, 2)

heart\_rate = 60 / np.mean(RR\_intervals)

print("R-R Intervals (s):", RR\_intervals)

print(f"Heart Rate: {heart\_rate:.2f} bpm")

plt.figure(figsize=(12, 6))

plt.plot(np.arange(len(ecg\_filtered)) / fs, ecg\_filtered, label='Filtered ECG Signal')

plt.plot(peaks / fs, ecg\_filtered[peaks], 'ro', markersize=8, label='R Points')

plt.title('ECG Signal with Detected R-Peaks')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.legend()

1. **Statistical parameters of ECG signals**

**MATLAB:**

clc; clear; close all;

file\_paths = {'115m (15).mat', '202m (15).mat', '223m (5).mat'};

fs = 360;

fc = 0.5;

for i = 1:length(file\_paths)

file\_path = file\_paths{i};

data = load(file\_path);

ecg\_signal = data.val(:);

[b, a] = butter(4, fc / (fs / 2), 'high');

ecg\_filtered = filtfilt(b, a, ecg\_signal);

mean\_original = mean(ecg\_signal);

std\_original = std(ecg\_signal);

var\_original = var(ecg\_signal);

mean\_filtered = mean(ecg\_filtered);

std\_filtered = std(ecg\_filtered);

var\_filtered = var(ecg\_filtered);

fprintf(' Original Signal:\n');

fprintf(' Mean value: %.4f\n', mean\_original);

fprintf(' Standard Deviation: %.4f\n', std\_original);

fprintf(' Variance: %.4f\n', var\_original);

fprintf(' Filtered Signal:\n');

fprintf(' Mean value: %.4f\n', mean\_filtered);

fprintf(' Standard Deviation: %.4f\n', std\_filtered);

fprintf(' Variance: %.4f\n', var\_filtered);

fprintf('\n');

end

**PYTHON**:

import numpy as np

import scipy.io

from scipy.signal import butter, filtfilt

file\_paths = ['/content/115m (15).mat', '/content/202m (15).mat', '/content/223m (5).mat']

fs = 360

fc = 0.5

for i, file\_path in enumerate(file\_paths, start=1):

data = scipy.io.loadmat(file\_path)

ecg\_signal = data['val'].flatten()

b, a = butter(4, fc / (fs / 2), btype='high')

ecg\_filtered = filtfilt(b, a, ecg\_signal)

mean\_original = np.mean(ecg\_signal)

std\_original = np.std(ecg\_signal)

var\_original = np.var(ecg\_signal)

mean\_filtered = np.mean(ecg\_filtered)

std\_filtered = np.std(ecg\_filtered)

var\_filtered = np.var(ecg\_filtered)

print(f' Original Signal:')

print(f' Mean value: {mean\_original:.4f}')

print(f' Standard Deviation: {std\_original:.4f}')

print(f' Variance: {var\_original:.4f}')

print(f' Filtered Signal:')

print(f' Mean value: {mean\_filtered:.4f}')

print(f' Standard Deviation: {std\_filtered:.4f}')

print(f' Variance: {var\_filtered:.4f}')

print()

1. **EEG waves classification - alpha, beta, theta, and delta**

**MATLAB:**

clc; clear; close all;

info = edfinfo("S001R02.edf");

data = edfread("S001R02.edf");

dataFp1 = edfread("S001R02.edf", 'SelectedSignals', 'Fp1.');

dataFp1\_cellarray = table2array(dataFp1);

dataFp1\_array = vertcat(dataFp1\_cellarray{:});

Fs = 160; order = 4;

t = (length(dataFp1\_array)) / Fs;

% Delta (0.5 - 4 Hz)

[b\_delta, a\_delta] = butter(order, [0.5, 4]/(Fs/2), 'bandpass');

delta\_wave = filtfilt(b\_delta, a\_delta, dataFp1\_array);

% Theta (4 - 8 Hz)

[b\_theta, a\_theta] = butter(order, [4, 8]/(Fs/2), 'bandpass');

theta\_wave = filtfilt(b\_theta, a\_theta, dataFp1\_array);

% Alpha (8 - 12 Hz)

[b\_alpha, a\_alpha] = butter(order, [8, 12]/(Fs/2), 'bandpass');

alpha\_wave = filtfilt(b\_alpha, a\_alpha, dataFp1\_array);

% Beta (12 - 30 Hz)

[b\_beta, a\_beta] = butter(order, [12, 30]/(Fs/2), 'bandpass');

beta\_wave = filtfilt(b\_beta, a\_beta, dataFp1\_array);

figure;

subplot(4, 1, 1); plot(t, delta\_wave);

title('Delta Wave (0.5 - 4 Hz)'); xlabel('Sample Number'); ylabel('Amplitude');

subplot(4, 1, 2); plot(t, theta\_wave);

title('Theta Wave (4 - 8 Hz)'); xlabel('Sample Number'); ylabel('Amplitude');

subplot(4, 1, 3); plot(t, alpha\_wave);

title('Alpha Wave (8 - 12 Hz)'); xlabel('Sample Number'); ylabel('Amplitude');

subplot(4, 1, 4); plot(t, beta\_wave);

title('Beta Wave (12 - 30 Hz)'); xlabel('Sample Number'); ylabel('Amplitude');

**PYTHON**:

pip install mne numpy matplotlib scipy

import numpy as np

import matplotlib.pyplot as plt

from scipy.signal import butter, filtfilt

import mne

file\_path = "S001R02.edf"

raw = mne.io.read\_raw\_edf(file\_path, preload=True)

data\_fp1 = raw.get\_data(picks=['Fp1.']).flatten()

fs = int(raw.info['sfreq'])

t = np.arange(len(data\_fp1)) / fs

order = 4

def bandpass\_filter(data, lowcut, highcut, fs, order=4):

nyquist = 0.5 \* fs

low = lowcut / nyquist

high = highcut / nyquist

b, a = butter(order, [low, high], btype='band')

return filtfilt(b, a, data)

# Delta (0.5 - 4 Hz)

delta\_wave = bandpass\_filter(data\_fp1, 0.5, 4, fs, order)

# Theta (4 - 8 Hz)

theta\_wave = bandpass\_filter(data\_fp1, 4, 8, fs, order)

# Alpha (8 - 12 Hz)

alpha\_wave = bandpass\_filter(data\_fp1, 8, 12, fs, order)

# Beta (12 - 30 Hz)

beta\_wave = bandpass\_filter(data\_fp1, 12, 30, fs, order)

plt.figure()

plt.subplot(4, 1, 1)

plt.plot(t, delta\_wave)

plt.title('Delta Wave (0.5 - 4 Hz)')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.subplot(4, 1, 2)

plt.plot(t, theta\_wave)

plt.title('Theta Wave (4 - 8 Hz)')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.subplot(4, 1, 3)

plt.plot(t, alpha\_wave)

plt.title('Alpha Wave (8 - 12 Hz)')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.subplot(4, 1, 4)

plt.plot(t, beta\_wave)

plt.title('Beta Wave (12 - 30 Hz)')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.tight\_layout()

1. **Power Spectrum**

**MATLAB:**

clc; clear; close all;

info = edfinfo("S001R02.edf");

data = edfread("S001R02.edf");

dataFp1 = edfread("S001R02.edf", 'SelectedSignals', 'Fp1.');

dataFp1\_cellarray = table2array(dataFp1);

dataFp1\_array = vertcat(dataFp1\_cellarray{:});

Fs = 160;

t = (0:5000) / Fs;

[pxx, f\_welch] = pwelch(dataFp1\_array, [], [], [], Fs);

figure;

plot(f\_welch, 10\*log10(pxx));

title('Power Spectral Density of EEG Signal (Fp1)'); xlabel('Frequency (Hz)'); ylabel('Power/Frequency (dB/Hz)');

grid on;

**PYTHON**:

pip install mne numpy matplotlib scipy

import numpy as np

import matplotlib.pyplot as plt

from scipy.signal import welch

import mne

file\_path = "S001R02.edf"

raw = mne.io.read\_raw\_edf(file\_path, preload=True)

Fs = int(raw.info['sfreq'])

data, times = raw.get\_data(picks=['Fp1.'], return\_times=True)

dataFp1\_array = data.flatten()

f\_welch, pxx = welch(dataFp1\_array, fs=Fs, nperseg=1024)

plt.figure()

plt.plot(f\_welch, 10 \* np.log10(pxx))

plt.title('Power Spectral Density of EEG Signal (Fp1)')

plt.xlabel('Frequency (Hz)')

plt.ylabel('Power/Frequency (dB/Hz)')

plt.grid()

plt.show()

1. **Auto and Cross-Correlation of EEG**

**MATLAB**:

%Cross-Correlation

clc; clear all; close all;

info = edfinfo("S001R02.edf");

data = edfread("S001R02.edf");

dataFp1 = edfread("S001R02.edf", 'SelectedSignals', 'Fp1.');

dataFp2 = edfread("S001R02.edf", 'SelectedSignals', 'Fp2.');

dataFp1\_cellarray = table2array(dataFp1);

dataFp2\_cellarray = table2array(dataFp2);

dataFp1\_array = vertcat(dataFp1\_cellarray{:});

dataFp2\_array = vertcat(dataFp2\_cellarray{:});

min\_len = min(length(dataFp1\_array), length(dataFp2\_array));

dataFp1\_array = dataFp1\_array(1:min\_len);

dataFp2\_array = dataFp2\_array(1:min\_len);

[cross\_corr, lags] = xcorr(dataFp1\_array, dataFp2\_array, 'coeff');

figure;

plot(lags, cross\_corr);

title('Cross-Correlation between Fp1 and Fp2 EEG Signals');

xlabel('Lag (Samples)'); ylabel('Cross-Correlation');

grid on;

% Autocorrelation

[auto\_corr, lags\_auto] = xcorr(dataFp1\_array, 'coeff');

[auto\_corr1, lags\_auto1] = xcorr(dataFp2\_array, 'coeff');

figure;

subplot(2,1,1);

plot(lags\_auto, auto\_corr);

title('Autocorrelation of Fp1 EEG Signal');

xlabel('Lag (Samples)');

ylabel('Autocorrelation');

grid on;

subplot(2,1,2);

plot(lags\_auto1, auto\_corr1);

title('Autocorrelation of Fp2 EEG Signal');

xlabel('Lag (Samples)');

ylabel('Autocorrelation');

grid on;

**PYTHON**:

pip install mne numpy matplotlib scipy

import numpy as np

import matplotlib.pyplot as plt

from scipy.signal import correlate

import mne

file\_path = "S001R02.edf"

raw = mne.io.read\_raw\_edf(file\_path, preload=True)

data\_fp1 = raw.get\_data(picks=['Fp1.']).flatten()

data\_fp2 = raw.get\_data(picks=['Fp2.']).flatten()

min\_len = min(len(data\_fp1), len(data\_fp2))

data\_fp1 = data\_fp1[:min\_len]

data\_fp2 = data\_fp2[:min\_len]

cross\_corr = correlate(data\_fp1, data\_fp2, mode='full', method='auto')

lags\_cross = np.arange(-min\_len + 1, min\_len)

cross\_corr /= np.max(np.abs(cross\_corr))

auto\_corr\_fp1 = correlate(data\_fp1, data\_fp1, mode='full', method='auto')

lags\_auto\_fp1 = np.arange(-min\_len + 1, min\_len)

auto\_corr\_fp1 /= np.max(np.abs(auto\_corr\_fp1))

auto\_corr\_fp2 = correlate(data\_fp2, data\_fp2, mode='full', method='auto')

lags\_auto\_fp2 = np.arange(-min\_len + 1, min\_len)

auto\_corr\_fp2 /= np.max(np.abs(auto\_corr\_fp2))

plt.figure()

plt.plot(lags\_cross, cross\_corr)

plt.title('Cross-Correlation between Fp1 and Fp2 EEG Signals')

plt.xlabel('Lag (Samples)')

plt.ylabel('Cross-Correlation')

plt.grid()

plt.figure()

plt.subplot(2,1,1)

plt.plot(lags\_auto\_fp1, auto\_corr\_fp1)

plt.title('Autocorrelation of Fp1 EEG Signal')

plt.xlabel('Lag (Samples)')

plt.ylabel('Autocorrelation')

plt.grid()

plt.figure()

plt.subplot(2,1,2)

plt.plot(lags\_auto\_fp2, auto\_corr\_fp2)

plt.title('Autocorrelation of Fp2 EEG Signal')

plt.xlabel('Lag (Samples)')

plt.ylabel('Autocorrelation')

plt.grid()

plt.show()

1. **Haar Transform - Image Fusion**

**MATLAB:**

clc;

clear;

close all;

i1 = imread('i\_2.jpg');

if size(i1,3) == 3

i1 = rgb2gray(i1);

end

i2 = imread('i\_1.jpg');

if size(i2,3) == 3

i2 = rgb2gray(i2);

end

[m,n] = size(i2);

figure;

subplot(1,3,1);

imshow(uint8(i1));

title('Source Image 1');

subplot(1,3,2);

imshow(uint8(i2));

title('Source Image 2');

X = cell(1,4);

Y = cell(1,4);

Z = cell(1,4);

[X{1},X{2},X{3},X{4}] = dwt2(i1, 'haar');

[Y{1},Y{2},Y{3},Y{4}] = dwt2(i2, 'haar');

for i = 1:4

Z{i} = max(X{i}, Y{i});

end

IF = idwt2(Z{1}, Z{2}, Z{3}, Z{4}, 'haar');

subplot(1,3,3);

imshow(IF, []);

title('DWT Max Fusion Rule');

% Calculate PSNR

psnr\_1 = psnr(uint8(IF), uint8(i1));

fprintf('PSNR wrt Source Image 1 is : %.4f\n', psnr\_1);

psnr\_2 = psnr(uint8(IF), uint8(i2));

fprintf('PSNR wrt Source Image 2 is : %.4f\n', psnr\_2);

**PYTHON**:

!pip install PyWavelets

import numpy as np

import cv2

import pywt

import matplotlib.pyplot as plt

from skimage.metrics import peak\_signal\_noise\_ratio as psnr

i1 = cv2.imread('/content/WhatsApp Image 2024-11-11 at 11.51.11\_71fab04a.jpg', cv2.IMREAD\_GRAYSCALE)

i2 = cv2.imread('/content/WhatsApp Image 2024-11-11 at 11.51.11\_a916711d.jpg', cv2.IMREAD\_GRAYSCALE)

plt.figure(figsize=(12,8))

plt.subplot(1, 3, 1)

plt.imshow(i1, cmap='gray')

plt.title('Source Image 1')

plt.subplot(1, 3, 2)

plt.imshow(i2, cmap='gray')

plt.title('Source Image 2')

coeffs1 = pywt.dwt2(i1, 'haar')

coeffs2 = pywt.dwt2(i2, 'haar')

cA1, (cH1, cV1, cD1) = coeffs1

cA2, (cH2, cV2, cD2) = coeffs2

cA = np.maximum(cA1, cA2)

cH = np.maximum(cH1, cH2)

cV = np.maximum(cV1, cV2)

cD = np.maximum(cD1, cD2)

fused\_image = pywt.idwt2((cA, (cH, cV, cD)), 'haar')

plt.subplot(1, 3, 3)

plt.imshow(fused\_image, cmap='gray')

plt.title('DWT Max Fusion Rule')

plt.show()

# Calculate PSNR values

psnr\_1 = psnr(i1, fused\_image)

psnr\_2 = psnr(i2, fused\_image)

print(f'PSNR with respect to Source Image 1: {psnr\_1:.4f}')

print(f'PSNR with respect to Source Image 2: {psnr\_2:.4f}')

1. **DWT Data Fusion**

**MATLAB:**

% Wavelet denoising using DWT (db1) on **ECG signal**

clc; clear; close all;

x = load('115m (15).mat');

y = x.val;

l1 = zeros(1,512);

h1 = zeros(1,512);

l2 = zeros(1,256);

h2 = zeros(1,256);

l3 = zeros(1,128);

h3 = zeros(1,128);

yy = y;

for i = 1:3

[CA, CD] = dwt(yy, 'db1');

if i == 1

l1 = CA;

h1 = CD;

elseif i == 2

l2 = CA;

h2 = CD;

else

l3 = CA;

h3 = CD;

end

yy = CA;

figure;

subplot(2, 1, 1); plot(CA);

title(['Approximation Coefficients at Level ', num2str(i)]);

legend('Approximation Coefficient');

subplot(2, 1, 2); plot(CD, 'r');

title(['Detail Coefficients at Level ', num2str(i)]);

legend('Detail Coefficient');

end

thres = 1.5;

h1(abs(h1) < thres) = 0;

h2(abs(h2) < thres) = 0;

h3(abs(h3) < thres) = 0;

ll2 = idwt(l3, h3, 'db1');

ll1 = idwt(ll2, h2, 'db1');

xx = idwt(ll1, h1, 'db1');

figure;

plot(y, 'b');

hold on;

plot(xx, 'r');

legend('Original (Noisy)', 'Denoised');

title('Original and Denoised ECG Signal');

xlabel('Samples');

ylabel('Amplitude');

grid on;

diff = y - xx;

diff = diff(:);

SE = sum(diff.^2);

MSE = SE/length(diff);

fprintf('MSE1: %.4f', MSE)

% Wavelet denoising using DWT (db1) on **EEG signal**

clc; clear; close all;

eeg\_data = edfread('S001R02.edf');

disp(eeg\_data);

eeg\_signal = eeg\_data{:, 1};

if iscell(eeg\_signal)

eeg\_signal = cell2mat(eeg\_signal);

end

l1 = zeros(1, 512);

h1 = zeros(1, 512);

l2 = zeros(1, 256);

h2 = zeros(1, 256);

l3 = zeros(1, 128);

h3 = zeros(1, 128);

yy = eeg\_signal;

for i = 1:3

[CA, CD] = dwt(yy, 'db1');

if i == 1

l1 = CA;

h1 = CD;

elseif i == 2

l2 = CA;

h2 = CD;

else

l3 = CA;

h3 = CD;

end

yy = CA;

figure;

subplot(2, 1, 1);

plot(CA);

title(['Approximation Coefficients at Level ', num2str(i)]);

legend('Approximation Coefficients');

subplot(2, 1, 2);

plot(CD, 'r');

title(['Detail Coefficients at Level ', num2str(i)]);

legend('Detail Coefficients');

end

thres = 1.5;

h1(abs(h1) < thres) = 0;

h2(abs(h2) < thres) = 0;

h3(abs(h3) < thres) = 0;

ll2 = idwt(l3, h3, 'db1');

ll1 = idwt(ll2, h2, 'db1');

xx = idwt(ll1, h1, 'db1');

figure;

plot(eeg\_signal, 'b');

hold on;

plot(xx, 'r');

legend('Original (Noisy)', 'Denoised');

title('Original and Denoised EEG Signal');

xlabel('Samples');

ylabel('Amplitude');

grid on;

diff = eeg\_signal - xx;

SE = sum(diff.^2);

MSE = SE / length(diff);

fprintf('MSE: %.4f\n', MSE);

**PYTHON**:

***ECG:***

import numpy as np

import matplotlib.pyplot as plt

import pywt

import scipy.io

data = scipy.io.loadmat('/content/115m (15).mat')

y = data['val'].flatten()

l1, h1 = np.zeros(512), np.zeros(512)

l2, h2 = np.zeros(256), np.zeros(256)

l3, h3 = np.zeros(128), np.zeros(128)

yy = y.copy()

for i in range(1, 4):

CA, CD = pywt.dwt(yy, 'db1')

if i == 1:

l1, h1 = CA, CD

elif i == 2:

l2, h2 = CA, CD

else:

l3, h3 = CA, CD

yy = CA

plt.figure(figsize=(10, 6))

plt.subplot(2, 1, 1)

plt.plot(CA)

plt.title(f'Approximation Coefficients at Level {i}')

plt.legend(['Approximation Coefficients'])

plt.subplot(2, 1, 2)

plt.plot(CD, 'r')

plt.title(f'Detail Coefficients at Level {i}')

plt.legend(['Detail Coefficients'])

plt.tight\_layout()

plt.show()

thres = 1.5

h1[np.abs(h1) < thres] = 0

h2[np.abs(h2) < thres] = 0

h3[np.abs(h3) < thres] = 0

ll2 = pywt.idwt(l3, h3, 'db1')

ll1 = pywt.idwt(ll2, h2, 'db1')

xx = pywt.idwt(ll1, h1, 'db1')

plt.figure(figsize=(10, 6))

plt.plot(y, 'b', label='Original (Noisy)')

plt.plot(xx, 'r', label='Denoised')

plt.legend()

plt.title('Original and Denoised ECG Signal')

plt.xlabel('Samples')

plt.ylabel('Amplitude')

plt.grid(True)

plt.show()

diff = y - xx

SE = np.sum(diff\*\*2)

MSE = SE / len(diff)

print(f'MSE: {MSE:.4f}')

***EEG:***

pip install mne numpy matplotlib scipy

import numpy as np

import matplotlib.pyplot as plt

import pywt

import mne

eeg\_file = '/content/S001R02.edf'

raw = mne.io.read\_raw\_edf(eeg\_file, preload=True)

eeg\_signal = raw.get\_data(picks=[0])[0]

l1, h1 = np.zeros(512), np.zeros(512)

l2, h2 = np.zeros(256), np.zeros(256)

l3, h3 = np.zeros(128), np.zeros(128)

yy = eeg\_signal.copy()

# Perform 3-level DWT using 'db1'

for i in range(1, 4):

CA, CD = pywt.dwt(yy, 'db1')

if i == 1:

l1, h1 = CA, CD

elif i == 2:

l2, h2 = CA, CD

else:

l3, h3 = CA, CD

yy = CA

plt.figure(figsize=(10, 6))

plt.subplot(2, 1, 1)

plt.plot(CA)

plt.title(f'Approximation Coefficients at Level {i}')

plt.legend(['Approximation Coefficients'])

plt.subplot(2, 1, 2)

plt.plot(CD, 'r')

plt.title(f'Detail Coefficients at Level {i}')

plt.legend(['Detail Coefficients'])

plt.tight\_layout()

plt.show()

thres = 1.5

h1[np.abs(h1) < thres] = 0

h2[np.abs(h2) < thres] = 0

h3[np.abs(h3) < thres] = 0

ll2 = pywt.idwt(l3, h3, 'db1')

ll1 = pywt.idwt(ll2, h2, 'db1')

xx = pywt.idwt(ll1, h1, 'db1')

plt.figure(figsize=(10, 6))

plt.plot(eeg\_signal, 'b', label='Original (Noisy)')

plt.plot(xx, 'r', label='Denoised')

plt.legend()

plt.title('Original and Denoised EEG Signal')

plt.xlabel('Samples')

plt.ylabel('Amplitude')

plt.grid(True)

plt.show()

diff = eeg\_signal - xx

SE = np.sum(diff\*\*2)

MSE = SE / len(diff)

print(f'MSE: {MSE:.4f}')

1. Skeletonisation - only MATLAB

clc;

clear all;

close all;

% Read the input image

imgr = imread('patient2ct1-masked.jpg');

% Convert to grayscale if the image is RGB

if size(imgr, 3) == 3 % Check if the image has 3 color channels

imgr = rgb2gray(imgr); % Convert to grayscale

end

% Binarize the grayscale image

img1 = imbinarize(imgr);

img=not(img1);

img=bwareaopen(img,1500);

b=[0 1 0;1 1 1;0 1 0];

intialopen=imopen(img,b);

sum=img - intialopen;

erodeimg=img;

while true

erodeimg=imerode(erodeimg,b);

openimg=imopen(erodeimg,b);

sum=sum + (erodeimg - openimg);

if all(erodeimg(:) == 0)

break;

end

end

subplot(1,3,1);

imshow(img1);

subplot(1,3,2);

imshow(img);

subplot(1,3,3);

imshow(sum);

1. **Boundary Detection of CT Image**

**MATLAB:**

clc;

clear all;

close all;

image\_path = 'patient2ct1-masked.jpg';

img = imread(image\_path);

if size(img, 3) == 3

img = rgb2gray(img);

end

level = graythresh(img);

binary\_img = imbinarize(img, level);

binary\_img = imopen(binary\_img, strel('disk', 5)); % Remove small objects

binary\_img = imclose(binary\_img, strel('disk', 15)); % Close small holes

[B, L] = bwboundaries(binary\_img, 'noholes');

figure;

imshow(img);

hold on;

for k = 1:length(B)

boundary = B{k};

plot(boundary(:,2), boundary(:,1), 'y', 'LineWidth', 4);

end

for k = 1:length(B)

boundary = B{k};

fill(boundary(:,2), boundary(:,1), 'w', 'FaceAlpha', 0.3);

end

title('Brain Boundary Marked');

hold off;

**PYTHON:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

from skimage import measure

from skimage.morphology import disk, opening, closing

from skimage.filters import threshold\_otsu

image\_path = '/content/patient2ct1-masked.jpg'

img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

thresh = threshold\_otsu(img)

binary\_img = img > thresh

binary\_img = opening(binary\_img, disk(5)) # Remove small objects

binary\_img = closing(binary\_img, disk(15)) # Close small holes

contours = measure.find\_contours(binary\_img, 0.8)

fig, ax = plt.subplots()

ax.imshow(img, cmap='gray')

for contour in contours:

ax.plot(contour[:, 1], contour[:, 0], color='yellow', linewidth=2)

for contour in contours:

ax.fill(contour[:, 1], contour[:, 0], 'w', alpha=0.3)

ax.set\_title('Brain Boundary Marked')

plt.axis('off')

plt.show()

1. Data fusion of 2 images and calculate the performance metrics(SNR, RMSE, SD, Entropy)

import cv2

import numpy as np

import matplotlib.pyplot as plt

from skimage.metrics import mean\_squared\_error, peak\_signal\_noise\_ratio

from scipy.stats import entropy

image1 = cv2.imread('/content/i\_1.jpg', cv2.IMREAD\_GRAYSCALE)

image2 = cv2.imread('/content/i\_2.jpg', cv2.IMREAD\_GRAYSCALE)

if image1.shape != image2.shape:

image2 = cv2.resize(image2, (image1.shape[1], image1.shape[0]))

fused\_image = cv2.addWeighted(image1, 0.5, image2, 0.5, 0)

# Root Mean Squared Error (RMSE)

rmse = np.sqrt(mean\_squared\_error(image1, fused\_image))

# Signal-to-Noise Ratio (SNR)

signal\_power = np.mean(fused\_image \*\* 2)

noise\_power = np.mean((fused\_image - image1) \*\* 2)

snr = 10 \* np.log10(signal\_power / noise\_power)

# Standard Deviation (SD)

std\_dev = np.std(fused\_image)

# Entropy

histogram, \_ = np.histogram(fused\_image, bins=256, range=(0, 256))

entropy\_value = entropy(histogram + 1e-9)

print(f"RMSE: {rmse:.2f}")

print(f"SNR: {snr:.2f} dB")

print(f"Standard Deviation (SD): {std\_dev:.2f}")

print(f"Entropy: {entropy\_value:.2f}")

plt.figure(figsize=(12, 8))

plt.subplot(3, 1, 1)

plt.imshow(image1, cmap='gray')

plt.title('Image 1')

plt.axis('off')

plt.subplot(3, 1, 2)

plt.imshow(image2, cmap='gray')

plt.title('Image 2')

plt.axis('off')

plt.subplot(3, 1, 3)

plt.imshow(fused\_image, cmap='gray')

plt.title('Fused Image')

plt.axis('off')

plt.tight\_layout()

plt.show()

1. **Implementation of filtering technique of ECG signal for low-frequency component removal.**

import numpy as np

import matplotlib.pyplot as plt

from scipy.io import loadmat

from scipy.signal import butter, filtfilt

data1 = loadmat("/115m (15).mat")

ecg\_signal1 = data1['val'].flatten()

fs = 360 # Sampling frequency

fc = 0.5 # Cut-off frequency

b1, a1 = butter(4, fc / (fs / 2), btype='high')

filtered\_ecg1 = filtfilt(b1, a1, ecg\_signal1)

plt.figure(figsize=(12, 8))

plt.subplot(3, 2, 1)

plt.plot(ecg\_signal1)

plt.title('Original ECG Signal')

plt.xlabel('Samples')

plt.ylabel('Amplitude')

plt.subplot(3, 2, 2)

plt.plot(filtered\_ecg1)

plt.title('High Pass Filtered ECG Signal (Python)')

plt.xlabel('Samples')

plt.ylabel('Amplitude')

plt.tight\_layout()

plt.show()

1. **Implementation of filtering technique of ECG signal for high-frequency component removal**

import numpy as np

import matplotlib.pyplot as plt

from scipy.io import loadmat

from scipy.signal import butter, filtfilt

data1 = loadmat("/115m (15).mat")

ecg\_signal1 = data1['val'].flatten()

fs = 360 # Sampling frequency

fc = 40 # Cut-off frequency

b, a = butter(4, fc / (fs / 2), btype='low')

filtered\_ecg1 = filtfilt(b, a, ecg\_signal1)

plt.figure(figsize=(12, 8))

plt.subplot(3, 2, 1)

plt.plot(ecg\_signal1)

plt.title('Original ECG Signal')

plt.xlabel('Samples')

plt.ylabel('Amplitude')

plt.subplot(3, 2, 2)

plt.plot(filtered\_ecg1)

plt.title('Low Pass Filtered ECG Signal (Python)')

plt.xlabel('Samples')

plt.ylabel('Amplitude')

plt.tight\_layout()

plt.show()