SIGNAL PROCESSING PROGRAMS

1. QRS detection(P6)

```
# --- Install dependencies ---
!pip install numpy scipy matplotlib wfdb
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
import wfdb

# --- Load ECG signal (MIT-BIH record 100 for example) ---
record = wfdb.rdrecord('100', sampto=2000, pn_dir='mitdb')
ecg = record.p_signal[:, 0]
fs = record.fs  # Sampling frequency

# --- Bandpass filter parameters ---
f_bandpass = [0.5, 40]   # Frequency range (Hz)
order_bandpass = 2

# --- Design Butterworth bandpass filter ---
b_bandpass, a_bandpass = signal.butter(order_bandpass,
                                       [f_bandpass[0] / (fs / 2), f_bandpass[1] / (fs / 2)],
                                       btype='bandpass')

# --- Apply filter to ECG signal ---
ecg_filtered = signal.filtfilt(b_bandpass, a_bandpass, ecg)

# --- Find peaks (R-peaks) ---
peaks, _ = signal.find_peaks(ecg_filtered, height=0.5, distance=round(0.6* fs))

# --- RR intervals & Heart rate ---
RR_intervals = np.diff(peaks) / fs
```

```
heart_rate = 60 / np.mean(RR_intervals)

# --- Define QRS window for visualization ---
QRS_window_pre = round(0.06 * fs)
QRS_window_post = round(0.06 * fs)

# --- Plot ECG with detected R-peaks ---
plt.figure(figsize=(12,6))
plt.plot(ecg_filtered, label='Filtered ECG', color='b')
plt.plot(peaks, ecg_filtered[peaks], 'ro', label='Detected R-peaks')
plt.title(f'QRS Detection (Estimated Heart Rate: {heart_rate:.2f} bpm)')
plt.xlabel('Samples')
plt.ylabel('Amplitude (mV)')
plt.legend()
plt.grid(True)
plt.show()
```

2. EEG classification(P9)

```
!pip install mne
import matplotlib.pyplot as plt
import numpy as np
from scipy.signal import butter , filtfilt
import mne
file_path="/content/S002R02.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/myquist
  high=highcut/myquist
  b,a=butter(order,[low,high],btype='band')
  return filtfilt(b,a,data)
delta_wave=bandpass_filter(data_fp1,0.5,4,fs,order)
theta_wave=bandpass_filter(data_fp1,4,8,fs,order)
alpha_wave=bandpass_filter(data_fp1,8,12,fs,order)
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.subplot(4,1,2)
plt.plot(t,theta_wave)
plt.title('Theta wave (4-8 Hz)')

plt.subplot(4,1,3)
plt.plot(t,alpha_wave)
plt.title('Alpha wave (8-12 Hz)')

plt.subplot(4,1,4)
plt.plot(t,beta_wave)
plt.title('Beta wave (12-30 Hz)')

plt.xlabel('Time(s)')
plt.ylabel('Amplitude')
```

3. **Power Spectrum(P8)**

```
!pip install mne numpy matplotlib scipy
import numpy as np
```

```
import matplotlib.pyplot as plt
from scipy.signal import welch
import mne
file_path="/content/S002R02.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()
```

4. **Statistical Parameters(P8)**

```
import scipy.io
from scipy.signal import butter, filtfilt
import numpy as np

file_paths=['/content/115m (15).mat','/content/202m (15).mat','/content/223m (5).mat']
fc=0.5
fs = 1000

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)

manual_mean_original = np.sum(ecg_signal) / len(ecg_signal)
mean_original=np.mean(ecg_signal)
std_original=np.std(ecg_signal)
var_original=np.var(ecg_signal)

manual_mean_filtered = np.sum(ecg_filtered) / len(ecg_filtered)
mean_filtered=np.mean(ecg_filtered)
std_filtered=np.std(ecg_filtered)
var_filtered=np.var(ecg_filtered)

print(f'Original Signal {i}')
print(f'  Mean: {mean_original:.2f}')
print(f'Mean (manual): {manual_mean_original:.2f}')
print(f'  Standard Deviation: {std_original:.2f}')
print(f'  Variance: {var_original:.2f}')

print(f'Filtered Signal {i}')
print(f'  Mean: {mean_filtered:.2f}')
print(f'Mean (manual): {manual_mean_filtered:.2f}')
print(f'  Standard Deviation: {std_filtered:.2f}')
print(f'  Variance: {var_filtered:.2f}')
```

IMAGE PROCESSING PROGRAMS

1. Boundary detection(P11)

```
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/MRI.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()
```

2. Image fusion using haar transform(P12)

```
!pip install pywt scikit-image opencv-python
import numpy as np
import cv2
import pywt
import matplotlib.pyplot as plt
from skimage.metrics import peak_signal_noise_ratio as psnr
```

```
# --- Read images in grayscale ---
i1 = cv2.imread('/content/CT.jpg', cv2.IMREAD_GRAYSCALE)
i2 = cv2.imread('/content/MRI.jpg', cv2.IMREAD_GRAYSCALE)

# --- Ensure same size ---
if i1.shape != i2.shape:
    i2 = cv2.resize(i2, (i1.shape[1], i1.shape[0]))

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

# --- Haar 2D DWT ---
cA1, (cH1, cV1, cD1) = pywt.dwt2(i1, 'haar')
cA2, (cH2, cV2, cD2) = pywt.dwt2(i2, 'haar')

# --- Fusion using max rule ---
cA = np.maximum(cA1, cA2)
cH = np.maximum(cH1, cH2)
cV = np.maximum(cV1, cV2)
cD = np.maximum(cD1, cD2)

# --- Inverse DWT ---
fused_image = pywt.idwt2((cA, (cH, cV, cD)), 'haar')

# --- Fix datatype for viewing and PSNR ---
```

```
fused_image = np.clip(fused_image, 0, 255)
fused_image_uint8 = fused_image.astype(np.uint8)

plt.subplot(1,3,3)
plt.imshow(fused_image_uint8, cmap='gray')
plt.title('DWT Max Fusion Rule')
plt.show()

h, w = i1.shape
fused_image_uint8_cropped = fused_image_uint8[:h, :w]

psnr_1 = psnr(i1, fused_image_uint8_cropped)
psnr_2 = psnr(i2, fused_image_uint8_cropped)

print(f'PSNR with respect to Source Image 1: {psnr_1:.4f}')
print(f'PSNR with respect to Source Image 2: {psnr_2:.4f}')
```

3. Image fusion without using haar transform(P13)

```
!pip install pywavelets matplotlib opencv-python
import cv2
import pywt
import matplotlib.pyplot as plt

img=cv2.imread('/content/MRI.jpg',cv2.IMREAD_GRAYSCALE)
cA,(cH,cV,cD)=pywt.dwt2(img,'haar')

plt.figure(figsize=(10,8))
plt.subplot(2,2,1)
plt.imshow(cA,cmap='gray')
plt.title('Approximation Coefficients')
```

```
plt.axis('off')

plt.subplot(2,2,2)
plt.imshow(cH,cmap='gray')
plt.title('Horizontal Detail Coefficients')
plt.axis('off')

plt.subplot(2,2,3)
plt.imshow(cV,cmap='gray')
plt.title('Vertical Detail Coefficients')
plt.axis('off')

plt.subplot(2,2,4)
plt.imshow(cD,cmap='gray')
plt.title('Diagonal Detail Coefficients')
plt.axis('off')
plt.show()

reconstructed=pywt.idwt2((cA,(cH,cV,cD)),'haar')
plt.figure(figsize=(10,4))

plt.subplot(1,2,1)
plt.imshow(img,cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1,2,2)
plt.imshow(reconstructed,cmap='gray')
plt.title('Reconstructed Image')
plt.axis('off')
plt.show()
```

4. Wavelet on ECG(P14)

```
!pip install mne
import numpy as np
import matplotlib.pyplot as plt
import pywt
import scipy.io
import mne

# Load ECG MAT file
data = scipy.io.loadmat('/content/115m (15).mat')
y = data['val'].flatten()

# Initialize holders (sizes automatically replaced by actual CA/CD)
l1 = h1 = None
l2 = h2 = None
l3 = h3 = None

yy = y.copy()

# Perform 3-level DWT with MATLAB-like padding
for i in range(1, 4):
    CA, CD = pywt.dwt(yy, 'db1', mode='sym')

    # Store coefficients
    if i == 1:
        l1, h1 = CA, CD
    elif i == 2:
        l2, h2 = CA, CD
```

```
    else:
        l3, h3 = CA, CD

    yy = CA

    # Plot coefficients
    plt.figure(figsize=(10, 6))
    plt.subplot(2, 1, 1)
    plt.plot(CA)
    plt.title(f'Approximation Coefficients at Level {i}')
    plt.subplot(2, 1, 2)
    plt.plot(CD, 'r')
    plt.title(f'Detail Coefficients at Level {i}')
    plt.tight_layout()
    plt.show()

# Thresholding (Hard threshold)
thres = 1.5
h1[np.abs(h1) < thres] = 0
h2[np.abs(h2) < thres] = 0
h3[np.abs(h3) < thres] = 0

# Reconstruct (reverse order)
ll2 = pywt.idwt(l3, h3, 'db1', mode='sym')
ll1 = pywt.idwt(ll2, h2, 'db1', mode='sym')
xx  = pywt.idwt(ll1, h1, 'db1', mode='sym')

# Plot Original vs Denoised
plt.figure(figsize=(10, 6))
plt.plot(y, 'b', label='Original (Noisy)')
plt.plot(xx, 'r', label='Denoised')
plt.title('ECG Signal (Original vs Denoised)')
```

```
plt.legend()
plt.grid(True)
plt.show()

# Compute MSE
diff = y - xx
MSE = np.mean(diff ** 2)
print(f'MSE (ECG): {MSE:.4f}')
```

5. Wavelet on EEG(P15)

```
import numpy as np
import matplotlib.pyplot as plt
import pywt
import scipy.io
import mne

# Read EDF file
raw = mne.io.read_raw_edf('/content/S002R02.edf', preload=True)

eeg_signal = raw.get_data(picks=[0])[0]

l1 = h1 = None
l2 = h2 = None
l3 = h3 = None

yy = eeg_signal.copy()

# Perform 3 levels of DWT
for i in range(1, 4):
    CA, CD = pywt.dwt(yy, 'db1', mode='sym')
```

```
    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.subplot(2, 1, 2)
    plt.plot(CD, 'r')
    plt.title(f'Detail Coefficients at Level {i}')

    plt.tight_layout()
    plt.show()

# Thresholding
thres = 1.5
h1[np.abs(h1) < thres] = 0
h2[np.abs(h2) < thres] = 0
h3[np.abs(h3) < thres] = 0

# Reconstruction
ll2 = pywt.idwt(l3, h3, 'db1', mode='sym')
ll1 = pywt.idwt(ll2, h2, 'db1', mode='sym')
xx  = pywt.idwt(ll1, h1, 'db1', mode='sym')
```

```
# Plot Original vs Denoised
plt.figure(figsize=(10, 6))
plt.plot(eeg_signal, 'b', label='Original (Noisy)')
plt.plot(xx, 'r', label='Denoised')
plt.legend()
plt.title('EEG Signal (Original vs Denoised)')
plt.grid(True)
plt.show()

# Compute MSE
MSE = np.mean((eeg_signal - xx)**2)
print(f'MSE (EEG): {MSE:.4f}')
```

***RAG, Excel(2 sheets), Hugging Face ,**

Prompt Eng , LangChain – Codes will be

provided. Learn only the theory *