A group of graphs showing different types of data

Description automatically generated with medium confidence

A close-up of a graph

Description automatically generated

**Similarities:**

Both methods allow for the extraction of the signal's amplitude (or power) and phase information at specific frequencies over time, as evidenced by the plotted real components and power for both 5 Hz and 25 Hz frequencies. The plots demonstrate that both methods can capture changes in the signal's power and phase over time, which is crucial for time-frequency analysis in EEG data. Each method seems to resolve the frequency components at 5 Hz and 25 Hz, showing the expected behavior of the signal at these frequencies.

**Differences:**

The wavelet transform typically provides better frequency resolution due to the variable window size that adapts based on the frequency. This might be evident in the power and real component plots at 5 Hz and 25 Hz, where the wavelet plots may show more detailed variations compared to the Hilbert transform. The Hilbert transform often results in a smoother representation of the signal, which might make it less capable of capturing rapid transient changes compared to the wavelet transform. This could be seen in the smoother curves in the Hilbert transform plots. The amplitude scales differ between the two methods, which could be due to different normalization or scaling factors applied during the transforms. The wavelet power appears to be on a larger scale than the Hilbert power, suggesting differences in how the two methods estimate signal energy.

Here are some potential causes for the differences observed in the magnitude and patterns between the Hilbert and Wavelet transform outputs:

**Normalization:**

The normalized Morlet wavelet is designed to have unit energy, which means that the magnitude of the wavelet convolution output will be influenced by the signal itself rather than the wavelet. This normalization ensures that the convolution reflects the actual energy present in the signal at the frequency of interest. In contrast, the non-normalized Morlet wavelet does not have this property, and its convolution with the signal can produce outputs with magnitudes that reflect the wavelet's energy. This can lead to differences in the magnitudes between normalized and non-normalized wavelet convolutions.

**Windowing Effect:**

The Morlet wavelet is essentially a windowed sinusoid, and its width is governed by the number of cycles specified. A Morlet wavelet with a smaller number of cycles will have a shorter time window, resulting in better temporal resolution but worse frequency resolution. Conversely, more cycles will give better frequency resolution but worse temporal resolution.

**Bandwidth and Frequency Resolution:**

The bandwidth of the filter used in the Hilbert transform can affect the range of frequencies analyzed around the central frequency. A narrow bandwidth may not capture all the energy present in a broad signal spectrum, while a wide bandwidth might include more noise or other signal components. The Wavelet transform inherently has better frequency resolution due to the wavelet's adaptability to different frequencies.

**Analysis Method:**

The Hilbert transform uses the analytic signal to compute the amplitude envelope and instantaneous phase of the signal. It assumes a narrowband signal where the instantaneous frequency does not vary much. The Wavelet transform provides a more localized time-frequency representation of the signal. This can lead to more pronounced transient features in the wavelet power and real component plots, as it can adapt to changes in frequency content over time.

**Averaging Process:**

For the wavelet transform, I am summing the power or amplitude before averaging, which could amplify any variations or noise in the signal. For the Hilbert transform, I am directly averaging the squared absolute values or the real parts, which can smooth out these variations.

**Phase Relationship in FFT:**

In my Hilbert transform computation, I am manipulating the phase of the FFT components to create the analytic signal, which can lead to differences in how the signal's amplitude and phase are represented compared to the Wavelet transform.

In conclusion, the choice of whether to normalize a wavelet or not and the method used to compute the analytic signal can have significant effects on the resulting time-frequency representation.

# Codes

**import** numpy **as** np

**from** scipy.signal **import** firwin, filtfilt

**from** scipy.fft **import** fft, ifft

**import** matplotlib.pyplot **as** plt

In [117]:

**def** filter\_eeg\_data(single\_trial\_data, sfreq, freq, bandwidth**=**3):

nyq **=** sfreq **/** 2

freqmin **=** freq **-** bandwidth **/** 2

freqmax **=** freq **+** bandwidth **/** 2

data\_length **=** len(single\_trial\_data)

max\_filter\_order **=** data\_length **//** 3

filter\_order **=** min(round(3 **\*** (sfreq **/** freqmin)), max\_filter\_order)

b **=** firwin(filter\_order, [freqmin, freqmax], pass\_zero**=False**, window**=**'hamming', fs**=**sfreq)

padlen **=** min(len(b) **\*** 3, data\_length **-** 1)

filtered\_data **=** filtfilt(b, 1, single\_trial\_data, padlen**=**padlen)

**return** filtered\_data

**def** compute\_fft\_ifft(data, fs):

fft\_data **=** np**.**fft**.**fft(data, axis**=**0)

*# Create a complex copy of "f"*

complex\_fft\_data **=** 1j **\*** fft\_data *# Multiplying by the complex operator "i"*

*# Identify the positive and negative frequencies*

n **=** fft\_data**.**shape[0]

positive\_frequencies **=** np**.**arange(0, n **//** 2 **+** 1)

negative\_frequencies **=** np**.**arange(n **//** 2 **+** 1, n)

*# Separate "Complexf" and "f" for positive and negative frequencies*

positive\_complex **=** complex\_fft\_data[positive\_frequencies]

negative\_complex **=** complex\_fft\_data[negative\_frequencies]

*# Rotate Complexf by -90 degrees for positive frequencies*

*# and by +90 degrees for negative frequencies*

positive\_complex **\*=** np**.**exp(**-**1j **\*** np**.**pi **/** 2)

negative\_complex **\*=** np**.**exp(1j **\*** np**.**pi **/** 2)

*# Add the rotated positive Complexf to the positive frequency coefficients ("f")*

fft\_data[positive\_frequencies] **+=** positive\_complex

*# Do the same for the negative frequencies*

fft\_data[negative\_frequencies] **+=** negative\_complex

*# Take the inverse Fourier transform (ifft)*

analytic\_signal **=** np**.**fft**.**ifft(fft\_data, axis**=**0)

**return** analytic\_signal

**def** normalized\_morlet\_wavelet(frequency, num\_cycles, sampling\_rate, duration**=**2):

t **=** np**.**linspace(**-**duration**/**2, duration**/**2, int(sampling\_rate **\*** duration), endpoint**=False**)

sine\_wave **=** np**.**exp(2j **\*** np**.**pi **\*** frequency **\*** t) *# Complex sine wave*

amplitude\_envelope **=** np**.**exp(**-**t**\*\***2 **\*** (np**.**pi **\*** frequency **/** num\_cycles)**\*\***2) *# Gaussian envelope*

wavelet **=** sine\_wave **\*** amplitude\_envelope

wavelet **/=** np**.**sqrt(np**.**sum(np**.**abs(wavelet)**\*\***2)) *# Normalize wavelet to have unit energy*

**return** wavelet

**def** morlet\_wavelet(frequency, num\_cycles, sampling\_rate, duration**=**2):

t **=** np**.**linspace(**-**duration**/**2, duration**/**2, int(sampling\_rate **\*** duration), endpoint**=False**)

sine\_wave **=** np**.**exp(2j **\*** np**.**pi **\*** frequency **\*** t) *# Complex sine wave*

amplitude\_envelope **=** np**.**exp(**-**t**\*\***2 **\*** (np**.**pi **\*** frequency **/** num\_cycles)**\*\***2) *# Gaussian envelope*

wavelet **=** sine\_wave **\*** amplitude\_envelope

**return** wavelet

**def** wavelet\_convolution(single\_trial\_data, wavelet):

n\_signal **=** len(single\_trial\_data)

n\_wavelet **=** len(wavelet)

n\_convolution **=** n\_signal **+** n\_wavelet **-** 1

data\_fft **=** np**.**fft**.**fft(single\_trial\_data, n\_convolution)

wavelet\_fft **=** np**.**fft**.**fft(wavelet, n\_convolution)

convolution\_result\_fft **=** data\_fft **\*** wavelet\_fft

convolution\_result **=** np**.**fft**.**ifft(convolution\_result\_fft)

start **=** (n\_wavelet **-** 1) **//** 2

end **=** start **+** n\_signal

**return** convolution\_result[start:end]

In [118]:

mat\_file\_path **=** 'sampleEEGdata.mat'

mat\_data **=** scipy**.**io**.**loadmat(mat\_file\_path)

eeg\_data **=** mat\_data['EEG']

*# Access the data and channel location fields*

eeg\_data\_field **=** eeg\_data['data'][0, 0]

chanlocs\_field **=** eeg\_data['chanlocs'][0, 0]

sampling\_rate **=** eeg\_data['srate'][0, 0][0, 0]

channel\_labels **=** [chanlocs\_field['labels'][0][i][0] **for** i **in** range(chanlocs\_field['labels']**.**shape[1])]

cz\_index **=** channel\_labels**.**index('Cz')

cz\_data **=** eeg\_data\_field[cz\_index, :, :] *# Data is in the format [channels, times, trials]*

frequencies **=** [5, 25]

num\_trials **=** cz\_data**.**shape[1]

sampling\_rate **=** 256

num\_cycles **=** 4

filtered\_data **=** {}

**for** freq **in** frequencies:

filtered\_data[freq] **=** np**.**zeros\_like(cz\_data)

**for** trial **in** range(num\_trials):

filtered\_data[freq][:, trial] **=** filter\_eeg\_data(cz\_data[:, trial], sampling\_rate, freq)

fft\_results **=** {}

power **=** {}

real\_component **=** {}

phase **=** {}

*# FFT and analytic signal computation*

fft\_results **=** {freq: compute\_fft\_ifft(filtered\_data[freq], sampling\_rate) **for** freq **in** frequencies}

*# Extract power and real components*

analytic\_signals\_fft **=** {

freq: {

'power': np**.**mean(np**.**abs(signal)**\*\***2, axis**=**1),

'real': np**.**mean(np**.**real(signal), axis**=**1)

} **for** freq, signal **in** fft\_results**.**items()

}

avg\_power\_wavelet **=** {freq: np**.**zeros(cz\_data**.**shape[0]) **for** freq **in** frequencies}

avg\_real\_wavelet **=** {freq: np**.**zeros(cz\_data**.**shape[0]) **for** freq **in** frequencies}

**for** freq **in** frequencies:

**for** trial **in** range(num\_trials):

*# Filter the data*

filtered\_data **=** filter\_eeg\_data(cz\_data[:, trial], sampling\_rate, freq)

*# Wavelet Convolution*

wavelet **=** morlet\_wavelet(freq, num\_cycles, sampling\_rate)

convolved\_data **=** wavelet\_convolution(filtered\_data, wavelet)

avg\_power\_wavelet[freq] **+=** np**.**abs(convolved\_data)

avg\_real\_wavelet[freq] **+=** np**.**real(convolved\_data)

avg\_power\_wavelet[freq] **/=** num\_trials

avg\_real\_wavelet[freq] **/=** num\_trials

*# Normalized Morlet Wavelet Convolution*

avg\_power\_wavelet\_n **=** {freq: np**.**zeros(cz\_data**.**shape[0]) **for** freq **in** frequencies}

avg\_real\_wavelet\_n **=** {freq: np**.**zeros(cz\_data**.**shape[0]) **for** freq **in** frequencies}

**for** freq **in** frequencies:

wavelet\_n **=** normalized\_morlet\_wavelet(freq, num\_cycles, sampling\_rate) *# Use the normalized wavelet*

**for** trial **in** range(num\_trials):

filtered\_data\_n **=** filter\_eeg\_data(cz\_data[:, trial], sampling\_rate, freq)

convolved\_data\_n **=** wavelet\_convolution(filtered\_data, wavelet)

avg\_power\_wavelet\_n[freq] **+=** np**.**abs(convolved\_data)**\*\***2

avg\_real\_wavelet\_n[freq] **+=** np**.**real(convolved\_data)

avg\_power\_wavelet\_n[freq] **/=** num\_trials

avg\_real\_wavelet\_n[freq] **/=** num\_trials

In [119]:

times **=** np**.**arange(cz\_data**.**shape[0]) **/** sampling\_rate

**for** freq **in** frequencies:

plt**.**figure(figsize**=**(20, 8)) *# Increased width to accommodate 2x2 subplots*

*# Wavelet Power*

plt**.**subplot(2, 3, 1) *# This is the first plot in a 2x2 subplot layout*

plt**.**plot(times, avg\_power\_wavelet[freq], label**=**f'Wavelet Power {freq} Hz')

plt**.**title(f'Wavelet Power at {freq} Hz')

plt**.**xlabel('Time (s)')

plt**.**ylabel('Power')

plt**.**legend()

*# Normalized Wavelet Power*

plt**.**subplot(2, 3, 2)

plt**.**plot(times, avg\_power\_wavelet\_n[freq], label**=**f'Wavelet Power {freq} Hz')

plt**.**title(f'Normalized Wavelet Power at {freq} Hz')

plt**.**xlabel('Time (s)')

plt**.**ylabel('Power')

plt**.**legend()

*# Hilbert Power*

plt**.**subplot(2, 3, 3) *# This is the second plot, on the right of the first plot*

plt**.**plot(times, analytic\_signals\_fft[freq]['power'], label**=**f'Hilbert Power {freq} Hz')

plt**.**title(f'Hilbert Power at {freq} Hz')

plt**.**xlabel('Time (s)')

plt**.**ylabel('Power')

plt**.**legend()

*# Wavelet Real Component*

plt**.**subplot(2, 3, 4) *# This is the third plot, below the first plot*

plt**.**plot(times, avg\_real\_wavelet[freq], label**=**f'Wavelet Real {freq} Hz')

plt**.**title(f'Wavelet Real Component at {freq} Hz')

plt**.**xlabel('Time (s)')

plt**.**ylabel('Amplitude')

plt**.**legend()

*# Normalized Wavelet Real Component*

plt**.**subplot(2, 3, 5)

plt**.**plot(times, avg\_real\_wavelet\_n[freq], label**=**f'Wavelet Power {freq} Hz')

plt**.**title(f'Normalized Wavelet Real Component at {freq} Hz')

plt**.**xlabel('Time (s)')

plt**.**ylabel('Power')

plt**.**legend()

*# Hilbert Real Component*

plt**.**subplot(2, 3, 6) *# This is the fourth plot, below the second plot*

plt**.**plot(times, analytic\_signals\_fft[freq]['real'], label**=**f'Hilbert Real {freq} Hz')

plt**.**title(f'Hilbert Real Component at {freq} Hz')

plt**.**xlabel('Time (s)')

plt**.**ylabel('Amplitude')

plt**.**legend()

plt**.**tight\_layout()

plt**.**show()