Linear Predictive Coding

What goes on?

What does it mean?

LPC is a technique for modeling speech signals by estimating the spectral envelope. It uses past samples in a linear combination with coefficients estimated by methods like autocorrelation, covariance, or the Burg algorithm. Pre-emphasis is applied to boost higher frequencies for better accuracy. Limitations include sensitivity to pitch estimation errors and the need for stationary signals.

What are we demonstrating?

Spectral Envelope Noise Reduction

Spectral Envelope -

Spectral envelope in speech processing is estimated using LPC, which is widely used to construct the frequency response of an LPC model. It has applications in speech synthesis, voice conversion, speech recognition, speech coding, and more. Ongoing advancements in spectral envelope estimation techniques continue to enhance accuracy and versatility in speech processing.

Noise reduction

Noise reduction is the process of removing noise from a signal while preserving the desired components. LPC-based approaches are commonly used in speech signals due to their ability to adapt to spectral characteristics. Noise reduction algorithms can be optimized for different types of noise, but may have limitations like introducing artifacts or distortions.

Code

```
import numpy as np
import scipy.signal as sps
import matplotlib.pyplot as plt
import wave

# Open the input WAV file in read binary mode
input_file = wave.open("male.wav", "rb")

# Get the sample rate, number of channels, sample width, and number of frames of the input WAV file
sample_rate = input_file.getframerate()
num_channels = input_file.getnchannels()
sample_width = input_file.getsampwidth()
num_frames = input_file.getnframes()

# Read the wave data from the input WAV file
wave_data = input_file.readframes(num_frames)

# Convert the wave data to an audio signal numpy array
audio_signal = np.frombuffer(wave_data, dtype=np.int16)

# Normalize the audio signal to the range [-1, 1]
audio_signal = audio_signal / np.iinfo(np.int16).max

# Add Gaussian noise to the audio signal
noise = np.random.normal(0, 0.1, audio_signal.shape)
noisy_signal = audio_signal + noise

# Scale the noisy signal to the range [-32768, 32767]
noisy_signal = noisy_signal * np.iinfo(np.int16).max
```

```
# Convert the noisy signal numpy array to bytes
noisy_data = noisy_signal.astype(np.int16).tobytes()

# Open the output WAV file in write binary mode
output_file = wave.open("noisy.wav", "wb")

# Set the sample rate, number of channels, sample width, and number of frames of the output WAV file
output_file.setframerate(sample_rate)
output_file.setnchannels(num_channels)
output_file.setsampwidth(sample_width)

# Write the noisy data to the output WAV file
output_file.writeframes(noisy_data)

# Close the input and output WAV files
input_file.close()
output_file.close()

# Set LPC parameters
order = 1 # Order of LPC analysis
frame_length = 0.01 # Frame length in seconds
frame_shift = 0.01 # Frame shift in seconds
sampling_rate = 1000 # Sampling rate of audio signal

# Divide the noisy signal into frames
frame_length_samples = int(frame_length * sampling_rate)
frame_shift_samples = int(frame_shift * sampling_rate)
frames = np.array([noisy_signal[i:i+frame_length_samples] for i in range(0, len(noisy_signal)-frame_length_samples, frame_shift_samples)])
```

```
# Apply LPC analysis to each frame
coefficients = np.zeros((order, len(frames)))
for i, frame in enumerate(frames):
    # Calculate the autocorrelation of the frame
    autocorrelation = np.correlate(frame, frame, mode='full')
    # Extract the desired range of the autocorrelation
    autocorrelation = autocorrelation[frame_length_samples-1:frame_length_samples+order]
    # Compute the 'r' vector
    r = -autocorrelation[1:]
    # Compute the Toeplitz matrix
    matrix = np.zeros((order, order))
    for j in range(order):
        matrix[j,j:] = autocorrelation[j:j+order-j]
        matrix[j:,j] = matrix[j,j:]
    # Solve the linear system and obtain the LPC coefficients
    coefficients[:, i] = np.dot(np.linalg.inv(matrix), r)
```

```
# LPC noise reduction
filtered_signal = np.zeros_like(noisy_signal)
for i, frame in enumerate(frames):
    # Compute the indices of the current frame in the filtered signal
    start = i * frame_shift_samples
    end = start + frame_length_samples
    # Filter the current frame with the corresponding LPC coefficients
    predicted_frame = sps.lfilter(np.append(1, -coefficients[:, i]), 1, frame)
    # Add the filtered frame to the filtered signal
    filtered_signal[start:end] += predicted_frame

# Round and convert filtered signal to int16 format
nr_signal = np.round(filtered_signal * np.iinfo(np.int16).max).astype(np.int16)

# Create a new wave file
output_file = wave.open("noise_reduced.wav", "wb")
```

```
# Set the frame rate, number of channels, and sample width of the wave file
output_file.setframerate(sample_rate)
output_file.setnchannels(num_channels)
output_file.setsampwidth(sample_width)

# Scale the filtered signal to the range of int16 values
nr = filtered_signal * np.iinfo(np.int16).max

# Convert the filtered signal to bytes in int16 format
filtered_data = filtered_signal.astype(np.int16).tobytes()

# Write the filtered data to the wave file
output_file.writeframes(filtered_data)

# Close the output file
output_file.close()
```

```python
# Plot original signal, noisy signal, and LPC noise reduction
fig, axs = plt.subplots(3, 1, figsize=(8, 12))
fig.suptitle('Signal and Noise Reduction After LPC')
axs[0].plot( audio_signal)
axs[0].set_title('Original Signal')
axs[1].plot( noisy_signal)
axs[1].set_title('Noisy Signal')
axs[2].plot(filtered_signal)
axs[2].set_title('LPC Noise Reduction')
plt.show()

# Plot original and filtered signal's spectral envelope
freq, orig_spec_env = sps.periodogram(noisy_signal, fs=sampling_rate, window='hamming', scaling='spectrum')
_, filt_spec_env = sps.periodogram(filtered_signal, fs=sampling_rate, window='hamming', scaling='spectrum')
plt.plot(freq, 20 * np.log10(np.abs(orig_spec_env)), label='Original Signal')
plt.plot(freq, 20 * np.log10(np.abs(filt_spec_env)), label='Filtered Signal')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude (dB)')
plt.title('Spectral Envelope')
plt.legend()
plt.show()
```

```
# Plot predicted frame
frame_index = 10 # Example frame index
predicted_frame = sps.lfilter(np.append(1, -coefficients[:, frame_index]), 1, frames[frame_index])
plt.figure()
plt.title('Predicted Frame')
plt.xlabel('Sample Index')
plt.ylabel('Amplitude')
plt.plot(frames[frame_index], label='Original Frame')
plt.plot(predicted_frame, label='Predicted Frame')
plt.legend()
plt.show()


# Calculate SNR for noisy and noise-reduced signals
noise_power = np.mean(noise**2) # Calculate the power of the noise signal
signal_power = np.mean(audio_signal**2) # Calculate the power of the original audio signal
noisy_signal_power = np.mean(noisy_signal**2) # Calculate the power of the noisy audio signal
nr_signal_power = np.mean(filtered_signal**2) # Calculate the power of the noise-reduced audio signal
snr_noisy = 20 * np.log10(signal_power / noise_power) # Calculate the SNR for the noisy signal
snr_nr = 20 * np.log10(signal_power / nr_signal_power) # Calculate the SNR for the noise-reduced signal
```

Code

```
print("SNR for noisy signal: {:.2f} dB".format(snr_noisy)) # Print the SNR for the noisy signal
print("SNR for noise-reduced signal: {:.2f} dB".format(snr_nr)) # Print the SNR for the noise-reduced signal

# Print the LPC coefficients
print(coefficients)
```

Working :

LPC analysis obtains LPC coefficients for each frame in the input signal, representing a linear prediction model based on past values. It involves autocorrelation computation, range extraction, Toeplitz matrix computation, and linear system solving to obtain the LPC coefficients.

Working :

Autocorrelation of each frame is computed by taking the dot product of the frame with a delayed version of itself. This function extracts a range of values used to compute LPC coefficients, based on the LPC model order.

$$r_k = \sum_{n=k+1}^{N} x[n]x[n-k], \quad k = 0, 1, \ldots, K,$$

Working :

LPC coefficients are obtained by solving linear equations using autocorrelation values. The coefficients represent a linear prediction model of each frame based on past values, obtained by computing the inverse of the Toeplitz matrix.

$$\boldsymbol{r} = \begin{bmatrix} r_1 & r_2 & \cdots & r_K \end{bmatrix}^\top .$$

$$\boldsymbol{R} = \begin{bmatrix} r_1 & r_2 & \cdots & r_K \\ r_2 & r_1 & \cdots & r_{K-1} \\ \vdots & \vdots & \ddots & \vdots \\ r_K & r_{K-1} & \cdots & r_1 \end{bmatrix} .$$

Working :

The filter that is applied to each frame is actually the inverse filter of the LPC analysis filter, meaning that it has the opposite frequency response. By applying this filter to the frame, we are effectively canceling out the noise component of the frame, while keeping the original signal intact.

$$\boldsymbol{Ra} = -\boldsymbol{r},$$

$$\boldsymbol{a} = -\boldsymbol{R}^{-1}\boldsymbol{r}.$$

$$\hat{x}[n] = \sum_{k=1}^{K} a_k x[n-k],$$

Working :

- The LPC (Linear Predictive Coding) analysis is not only used for speech signal compression, but also for noise reduction applications.

- After estimating the LPC coefficients for each frame, a noise reduction step is performed. The original signal is divided into overlapping frames and a filter, which is the inverse of the LPC analysis filter, is applied to each frame.

- This filter cancels out the noise component of the frame, resulting in a filtered version of the original signal with reduced noise.

Applications :

Speech processing and coding

LPC is used in codecs such as GSM and G.729 to compress speech signals for transmission.

Speaker diarization

LPC can be used to identify speakers in audio recordings based on their speech characteristics

Challenges with non-speech signals

Applying LPC to non-speech signals like music can be challenging, and alternative techniques such as harmonic modelling may be used instead.

LPC in finding pitch:

combination of filter coefficient and excitation signal. by eliminating the filter coefficient resonant frequency can be determined using which pitch is determined

Applications :

Speech coding and compression:

EX:CELP algorithm. LPC can be effectively represent sprectral envelope of speech signal, used to reduce bit rate.

Speech synthesis:

New speech waveform is generated by applying the set of LPC coefficient to a source signal, resulting speech waveform will have a spectral envelope similar to the original speech signal.

Speech recognition:

It is used in conjunction with other techniques such as HMMs or neural to identify words or phase is spoken language. It can be used to extract format frequency or spectral energy distribution.

Advatanges of LPC over other techniques

Advantages of LPC:

- Effective spectral envelope representation: It uses small number of coefficients.
- Robustness to noise: LPC can be used to model speech signals in presence of noise.
- Low computational complexity: It can be implemented efficiently using standard signal processing algorithm.
- Easy interpretation: Easy to interpret the results as the physical interpretation of coefficients are easy.

THANK YOU!!