Signals and Systems

Dr. Ehsan Moshksar
TA: Sina Sabouki

Final project

Helia Shames
Faeze Ebrahimi
Mohammadreza Naziri

Part 1

This MATLAB code performs the following tasks:

1. Set up the recording parameters:
   - `duration`: Specifies the duration of the recording in seconds.
   - `fs`: Sampling frequency, which is set to 19 kHz (19000 Hz).
   - `numBits`: Number of bits saved in each sample, set to 8.
   - `numChannels`: Number of audio channels, set to 1.

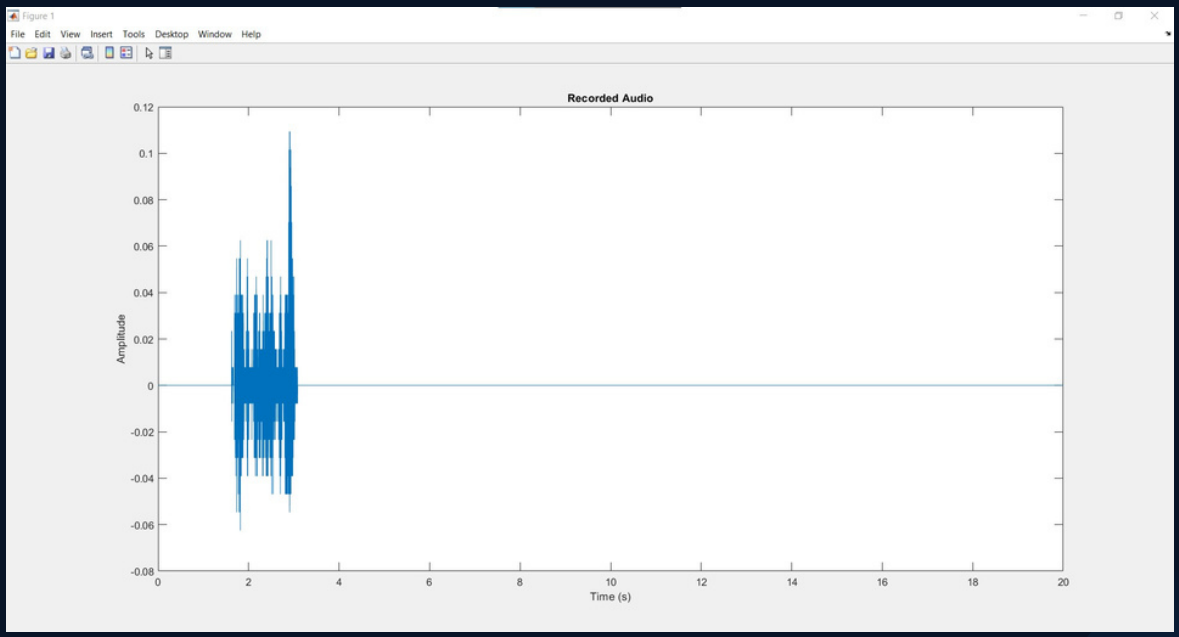
2. Create an `audiorecorder` object named `recObj` using the specified recording parameters.

3. Start the recording by displaying the message "Start speaking..." and using the `recordblocking` function to wait for the specified duration.

4. Display the message "Recording complete."

5. Retrieve the recorded audio data from the `audiorecorder` object using the `getaudiodata` function and store it in the variable `audioData`.

6. Plot the recorded audio waveform using the time axis calculated from the length of the audioData and the sampling frequency.



7. Calculate the power signal per sample by taking the mean of the squared absolute values of the audioData.

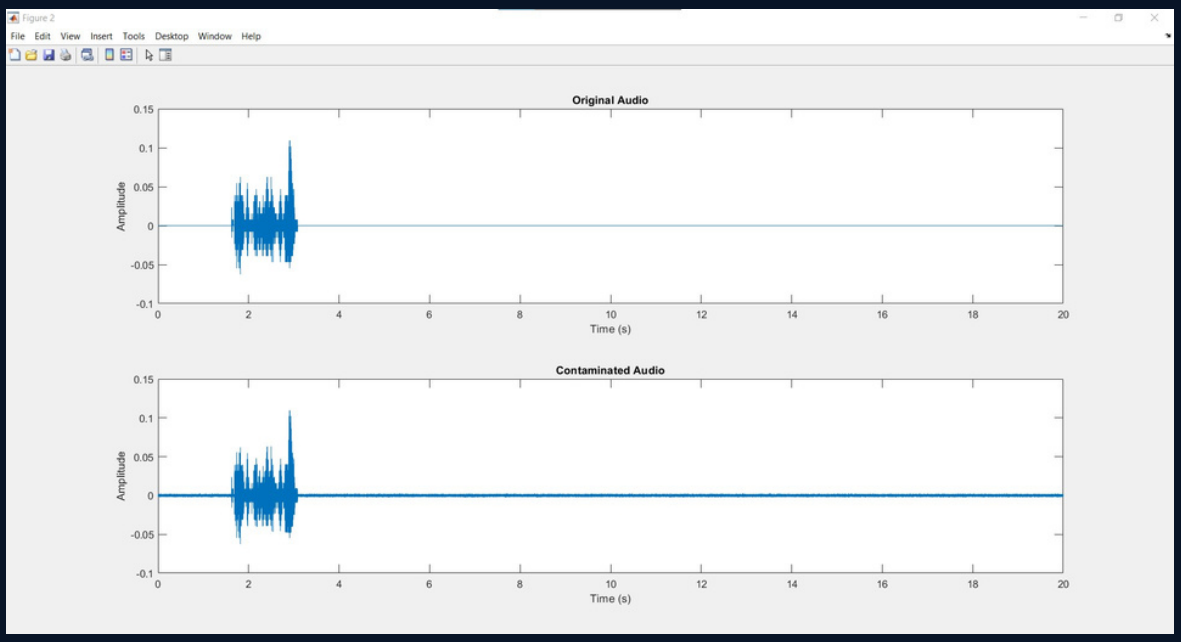
8. Convert the power signal per sample to decibels (dB) using the logarithmic scale.

9. Display the calculated power signal per sample.

10. Specify the desired signal-to-noise ratio (SNR) in dB. In this case, the SNR is set to 15 dB.

11. Add white Gaussian noise to the recorded audio using the `awgn` function. The noise is added based on the specified SNR and the power signal per sample in dB.

12. Plot the original audio waveform and the contaminated audio waveform (audio with added noise) for visual comparison.



13. Save the recorded audio and contaminated audio as .wav files in the "Audios" directory. If the directory doesn't exist, it will be created.

In summary, this code records audio for a specified duration, adds Gaussian noise to the recorded audio based on the desired SNR, and saves the original and contaminated audio for further analysis or processing.

Question: Why we calculate power signal per sample?
Answer:
In MATLAB, when adding AWGN (Additive White Gaussian Noise) to a signal, it is common to calculate the power of the noise per sample. This is done to ensure that the noise power is consistent regardless of the length or number of samples in the signal.

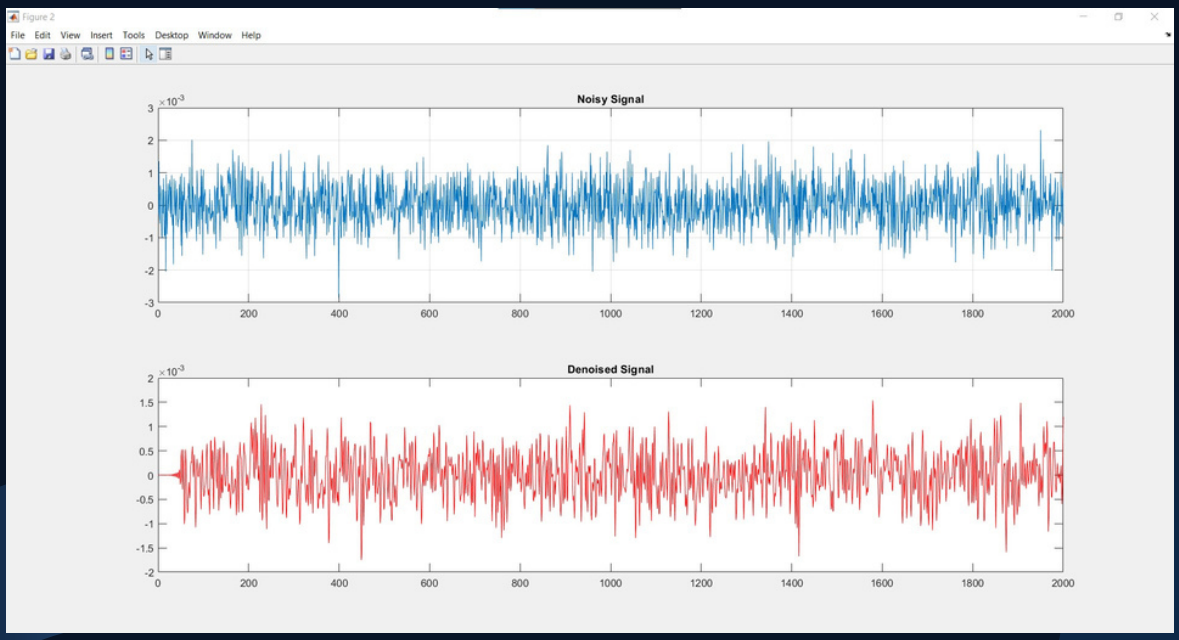
The power of a signal is defined as the average energy per unit time. In the case of discrete-time signals, the power per sample is calculated as the average energy per sample.

When adding AWGN to a signal in MATLAB, the awgn function requires the input noise power to be specified in terms of power per sample. By specifying the noise power per sample, you can control the amount of noise added to the signal regardless of its length or number of samples.

Part 2

This MATLAB code defines a function named `LPFilter` that performs denoising on a given audio signal. Here's a step-by-step explanation of the code:

1. Set up some variables for input recognition and initialization. `flag1` and `flag2` are flags to indicate if certain operations should be performed. `fp` is the default pass frequency for the filter.
2. Perform input recognition based on the number of input arguments (`nargin`) passed to the function. The code handles three possible cases:
   - If no input arguments are provided, the defaults from step 2 are used.
   - If one input argument is provided, it is either the noisy signal or the pass frequency.
   - If two input arguments are provided, they are the noisy signal and either the pass frequency or the flag for comparison.
   - If three input arguments are provided, they are the noisy signal, pass frequency, and the flag for comparison.
3. Design the filter using a Parks-McClellan optimal equiripple FIR filter design (`firceqrip`). The filter parameters include filter order `n`, sampling frequency `fs`, and attenuation specifications (`ap` and `ast` for passband and stopband ripple).
4. Perform denoising if `flag2` is set to 1. Convolve the noisy signal with the designed filter using the `conv` function and store the denoised signal in the variable `dSignal`. Additionally, write the denoised signal to a WAV file named `"FilteredAud.wav"` using `audiowrite`.
5. Perform comparison if `flag1` is set to 1. Display the original noisy signal using `audioplayer` and plot it. Calculate and display the signal power and variance. If `flag2` is also set to 1, display and play the denoised signal. Plot the noisy and denoised signals, calculate and display their power and variance. The comparison is done using audio playback and visual representations of the signals.



6. The function ends, returning the filter and denoised signal as output arguments if applicable.

In summary, this code defines a denoising function that takes in a noisy audio signal, applies a FIR filter, and optionally compares the original and denoised signals. The denoised signal and filter are returned as outputs.

Part 3

A lowpass filter is designed to attenuate or remove high-frequency components from a signal while allowing low-frequency components to pass through. However, it cannot completely remove all the noise present in a signal for the following reasons:

1. Noise may have frequency components that overlap with the desired signal: In some cases, the noise may have frequency components that are similar to the frequency components of the desired signal. In such situations, a lowpass filter may not be able to completely remove the noise without affecting the desired signal.

2. Noise may be spread across a wide frequency range: Some types of noise, such as white noise, have a relatively flat frequency spectrum and contain energy across a wide range of frequencies. A lowpass filter is designed to attenuate high-frequency components but may not effectively remove noise that is spread across a wide frequency range.

3. Noise may have a higher power level than the desired signal: If the power level of the noise is significantly higher than that of the desired signal, a lowpass filter may not be able to sufficiently attenuate the noise to an acceptable level.

4. Limitations of the filter design: The effectiveness of a lowpass filter in removing noise depends on its design parameters such as cutoff frequency, filter order, and filter type. If these parameters are not appropriately chosen or if the filter design itself is not optimal, the filter may not be able to remove all the noise.

In practice, a combination of filtering techniques, such as using multiple filters or employing adaptive filtering algorithms, may be necessary to effectively reduce noise in a signal.