Signal Processing Workflow: From Raw Data to FFT Analysis

3rd January 2025

Overview of the Process

This presentation explains the step-by-step process of transforming raw signal data into useful frequency-domain information:

1. Loading Raw Signal Data from '.mat' files.
2. Resampling the signals to a fixed time interval ($dt = 0.002$).
3. Filtering signals based on duration percentiles (15th and 85th).
4. Padding signals to match the length of the longest signal.
5. Computing the Fast Fourier Transform (FFT) of each padded signal.
6. Saving the results in CSV or MATLAB table format.

Each step is crucial for ensuring consistency, reliability, and correctness in signal processing.

Step 1: Loading Raw Signal Data

The raw data is stored in '.mat' files with a consistent naming scheme like `RSN{number}_P.mat`. The MATLAB array contains:

- The number of samples in the signal. (0th element)
- The sampling interval (dt), indicating how frequently the data was sampled. (1st element)
- The actual signal values, which represent the measurements at each time point. (starting from the 2nd element)

Why is this step necessary?

- Signals must be properly understood in terms of their number of samples and sampling rate before further processing.

We use MATLAB's `load` function to import these signals and proceed to the next steps.

Step 2: Resampling to a Fixed dt All signals are resampled to a fixed time interval ($dt = 0.002$) to ensure uniformity.

- Signals are recorded at different time intervals in the raw data.
- Standardizing the time interval ensures that all signals are comparable and can be processed using consistent techniques.
- Resampling is achieved through linear interpolation, ensuring that the signal is smoothly adjusted to the new time intervals.

Why is this step necessary?

- Resampling to lowest available dt helps in recording finer details of information.
- Over-sampling is better than under-sampling.
- It is important to use a consistent dt (in the current implementation - the lowest possible value) so that computations like FFT, which rely on uniform time intervals, are valid.

The signals are now uniformly sampled at the same dt of $0.002s$, making them suitable for further analysis.

Step 3: Filtering by Duration Percentiles

Not all signals are of equal duration, and some may be too short or too long for our analysis. We use duration filtering to retain only those signals within a specific range, defined by the 15th and 85th percentiles of all signal durations.

- First, we calculate the duration of each signal as the number of samples multiplied by the sampling interval (dt).
- Then, we compute the 15th and 85th percentiles of the durations to identify the range of acceptable signal lengths.
- Signals with durations falling outside this range are discarded.

Why is this step necessary?

- Signals that are too short may not contain enough information for meaningful analysis.
- Extremely long signals might introduce noise or computational inefficiency.

This step ensures that only signals of appropriate duration are retained for analysis.

Step 4: Padding Signals to Match the Maximum Length

After filtering, we have signals of varying lengths. To ensure compatibility during the FFT computation, all signals must be padded to the length of the longest signal.

- The longest signal is identified.
- Shorter signals are padded with zeros at the end to match this length.
- Padding is necessary because the FFT algorithm requires input signals to be of equal length.

Why is this step necessary?

- Signals of unequal length would cause issues during FFT computation, where each signal is expected to have the same number of data points.
- Zero-padding ensures that all signals are aligned and that the FFT calculation is applied consistently across all signals.

Padding ensures that the entire dataset is compatible with the FFT algorithm and can be analyzed uniformly.

Step 5: Computing the Fast Fourier Transform (FFT)

The core of the analysis involves transforming the signals into the frequency domain using the Fast Fourier Transform (FFT).

- The FFT converts a time-domain signal into a frequency-domain representation, revealing the frequency components of the signal.
- The FFT is essential for understanding the frequency content of a signal. Many signals, especially those in engineering and physics, can be more easily interpreted in the frequency domain.
- The positive frequency components contain all the useful information, as the negative frequencies are redundant for real-valued signals.
- Although padding the signals with zeros increases their length, the FFT effectively bypasses the problem of these added zeros by focusing on the frequency components of the actual signal, excluding the zero-padding.

Why is this step necessary?

- The FFT is essential for understanding the frequency content of signals, especially when analyzing oscillatory patterns, vibrations, or other periodic phenomena.
- Zero-padding the signal does not affect the accuracy of the frequency content analysis because FFT ignores the zero values when calculating the frequency spectrum.

FFT provides us with a powerful tool to analyze the frequency characteristics of each signal while handling padded zeros efficiently.

Step 6: Saving Results in CSV or MATLAB Table Format

Finally, the results of the FFT computations are saved in a structured table format for further analysis or visualization.

- The table contains two columns: the RSN number (extracted from the file name) and the corresponding FFT values.
- The results are saved as a MATLAB table and optionally exported to CSV format for easy sharing and external use.

Why is this step necessary?

- Storing results in a table allows easy access to the computed FFT data.
- It enables further analysis, such as statistical comparisons, visualization, and reporting.

The final output is a neatly organized table that can be analyzed, visualized, or shared with others.

Summary of the Process

Here's a recap of the steps involved in processing the signal data:

1. Load raw signal data from MATLAB '.mat' files.
2. Resample the signals to a fixed $dt = 0.002$ for uniformity.
3. Filter signals by duration percentiles (15th and 85th) to discard inappropriate signals.
4. Pad shorter signals to match the length of the longest signal for uniform FFT computation.
5. Compute the FFT to analyze the frequency components of each signal.
6. Save results in a structured table format (MATLAB or CSV).

Each step is critical for ensuring the quality, consistency, and validity of the data before applying advanced analysis techniques like FFT.