Hump

1. Introduction

This project aims to automatically detect and analyze hump regions in the frequency spectrum of vibration signals. The method filters out sharp peaks, identifies high-energy bands using adaptive thresholds, and detects clusters of frequencies representing the hump.

2. Prerequisites

- **Programming Language:** Python 3.11+
- **Libraries:**
  - `numpy`
  - `pandas`
  - `librosa`
  - `matplotlib`

Install dependencies using:

pip install -r requirements.txt

3. Project Structure

Hump/

```
├── src/                # Source code modules
│   ├── Global_Translator.py
│   ├── hump.py
│   └── signal_processing.py
```

```
├── requirements.txt       # Python package requirements
└── README.md
```

4. Code Documentation

4.1. Preprocessing Script (`src/hump.py`):

"""

Functions:

- detect_hump_range_clusters(signal, min_freq=500, max_freq=None, top_percent=15, gap_threshold=100): Detects the most prominent frequency range (hump) in a signal using clustering of high-energy components.

- normalize_and_clip(signal_ft, std_factor=4): Normalizes and clips a frequency spectrum by zeroing out values above a statistical threshold.

- check_hump_with_iso_ratio(signal_obj, freq_range, iso_energy, std_factor=4, ratio_threshold=0.05): Checks whether the energy of a given frequency range (potential hump) is significant compared to the total energy (above 500 Hz), scaled by a given ISO energy value.

- is_hump_shifting_downward(signals: list, top_percent=15, min_shift_hz=1000): Checks whether the hump frequency is consistently shifting downward across the list of signal objects.

"""

```
def detect_hump_range_clusters(signal, min_freq=500,
max_freq=None, top_percent=15, gap_threshold=100):
    """
    Detects the most prominent frequency range (hump) in a signal using
clustering of high-energy components.

    This method selects the top top_percent energy frequencies (after
clipping high outliers)
```

```
    and groups them into clusters based on proximity (less than
gap_threshold Hz).
    The largest cluster is selected as the hump range.

    Parameters:
    - signal (TS): signal object containing fq (frequency array) and ft
(magnitude spectrum)
    - min_freq (int, optional): minimum frequency to consider for hump
detection (default: 500 Hz)
    - max_freq (int or None, optional): maximum frequency to consider
(default: last frequency in fq)
    - top_percent (int, optional): percentage of highest spectral
magnitudes to include (default: 15)
    - gap_threshold (int, optional): maximum allowed gap (in Hz) between
consecutive high-energy frequencies
                                to consider them part of the same cluster
(default: 100)

    Returns:
    - tuple: (start_freq, end_freq) of the detected hump range
    """

def normalize_and_clip(signal_ft, std_factor=4):
    """
    Normalizes and clips a frequency spectrum by zeroing out values
above a statistical threshold.

    Parameters:
    - signal_ft (np.ndarray): 1D array of spectral magnitudes (usually ft
from FFT)
    - std_factor (float, optional): number of standard deviations above
mean to set as threshold (default: 4)

    Returns:
    - np.ndarray: clipped spectrum where values above (mean +
std_factor * std) are set to zero
    """

def check_hump_with_iso_ratio(signal_obj, freq_range, iso_energy,
std_factor=4, ratio_threshold=0.05):
```

```
    """
    Checks whether the energy of a given frequency range (potential hump) is significant
    compared to the total energy (above 500 Hz), scaled by a given ISO energy value.

    Parameters:
    - signal_obj (TS): a signal object containing frequency (fq) and spectrum (ft)
    - freq_range (tuple): frequency range (start_freq, end_freq) to check for hump
    - iso_energy (float): scaling factor based on ISO baseline energy
    - std_factor (float, optional): factor for standard deviation clipping during normalization (default: 4)
    - ratio_threshold (float, optional): minimum ratio threshold to consider as significant (default: 0.05)

    Returns:
    - (bool, float):
        - True if scaled energy ratio in the given range is above the threshold, False otherwise
        - Computed energy ratio
    """


def is_hump_shifting_downward(signals: list, top_percent=15, min_shift_hz=1000):
    """
    Checks whether the hump frequency is consistently shifting downward
    across the list of signal objects.

    Parameters:
    - signals (list): list of TS signal objects (already loaded and preprocessed)
    - top_percent (int): percentage of top energy used for hump detection
    - min_shift_hz (float): minimum downward shift (in Hz) to consider as significant

    Returns:
    - bool: True if downward shift detected, else False
```

"""

5. How to Run

1. Clone this repository and install dependencies:
2. git clone https://github.com/fatememajdi/Freq-Hump-Analysis.git
3. cd Freq-Hump-Analysis
4. pip install -r requirements.txt
5. Input .txt signal files should be placed under data/
6. Run the notebook:
    - First: notebooks/evaluate.ipynb

6. Outputs

The project produces the following key outputs:

- **Hump Frequency Range**: A frequency interval (in Hz), returned as a tuple (start_freq, end_freq), indicating the region where abnormal energy (hump) is detected in the spectrum.
- **Downward Hump Trend Detection**: A boolean value (True or False) indicating whether the hump frequency range is shifting downward over time — a potential indicator of mechanical degradation.
- **Diagnostic Summary** *(optional)*: When analyzing multiple signals, a summary of detected humps, energy metrics, and trend status can be printed or logged for further inspection.