SOP: electrophysiological data pre-processing - human

Table of contents

Objective

This Standard Operating Procedure (SOP) provides a comprehensive description of the steps required to process raw electrophysiological recordings into pre-processed bipolar traces. These bipolar traces serve as the foundation for subsequent processing and analysis in nearly all projects involving human intracranial data. The document outlines the pre-processing pipeline as implemented in the **Corteo** framework, ensuring consistency and reproducibility across studies.
Furthermore, this SOP details the structure of the HDF5 files used to store both the raw and pre-processed data. This includes specifications for data organization, metadata annotations, and conventions adopted to facilitate seamless integration with downstream analysis pipelines.

By following this SOP, users will ensure that data is processed in a standardized manner, enabling collaborative research and reproducible results across projects.

Electrophy Nomenclature

- **Contact:** A single, physical, metallic element used to record electrical signals from neural tissue.
- **Electrode:** A collection of one or more contacts organized into a physical unit, such as a lead, grid, or strip, designed for specific electrophysiological applications.
- **Device:** A set of electrodes of the same type, forming a cohesive system for recording or stimulation purposes.
- **Trace:** A time series representing electrophysiological data recorded from one or more contacts.
- **Channel:** A trace derived from a single contact or a combination of contacts, as determined by the chosen montage.
- **Montage:** The configuration that defines how signals from contacts are combined or referenced to create channels. Examples include bipolar, monopolar, referential, or common average montages.

HDF5 nomenclature

The HDF5 structure organizes data hierarchically using **Groups**, **Datasets**, and **Attributes**, providing a flexible and efficient framework for storing complex datasets with metadata.

- **Group:**A Group functions like a folder in a file system. It can contain other Groups (sub-Groups) and Datasets, enabling a hierarchical organization of data.
- **Dataset:**A Dataset is an n-dimensional array of homogeneous data (i.e., all elements have the same data type). Unlike Groups, Datasets cannot contain sub-Datasets or sub-Groups.
- **Attribute:**Attributes are metadata elements attached to a Group or a Dataset. Each Attribute consists of a key-value pair that provides additional information about the entity it is associated with. Multiple Attributes can be added to a single Group or Dataset to store as much descriptive metadata as needed.

Channel Structure in the HDF5 File

Each channel is represented as a **Dataset** within the HDF5 file, named according to the channel name. These Datasets are organized into hierarchical Groups for clarity and accessibility.

- **Raw Traces:**
  - Stored within the **"raw"** Group.
  - Datasets are referenced by their channel names as recorded (e.g., the "natus" column in the lookup.xlsx file).
- **Bipolar Traces:**
  - Stored within the **"bipolar"** Group.

  - Datasets are referenced by their bipolar names, as defined in the lookup.xlsx file.
  - Subgrouped by **device** (e.g., "lead," "scalp," "grid," etc.).
  - Further subgrouped by **electrode** (e.g., "lead_A," "lead_B," etc.).

Channel Attributes

Every channel (Dataset) is annotated with the following Attributes to provide detailed metadata:

- **unit**: A string specifying the unit of measurement for the trace (e.g., "mV").
- **sfreq**: A float indicating the sampling frequency of the trace in Hertz (e.g., 1024Hz).
- **grade**: A string representing the quality of the trace. This attribute is user-defined in Corteo. Possible values include:
  - UNSPECIFIED: The quality has not been assessed.
  - NOISY: The trace is affected by noise.
  - IED: Interictal epileptiform discharges are present.
  - ICTAL: The trace captures ictal activity.
  - NORMAL: The trace shows no abnormalities.
- **n_samples**: An integer specifying the number of samples in the trace array.
- **processing**: A string documenting the sequence of processing steps applied to the trace. For example, a pre-processed trace might have the following processing chain:*"Re-reference to bipolar; Dampen noisy periods (Hann window); Bandpass filter 0.1-200Hz (FIR filter, firwin design); Notch filter 50Hz and harmonics (FIR filter, firwin design); Decimate to 512Hz;"*.

HDF5 structure

- meta
  - Attribute: creation_date
  - Attribute: subject_id
  - Attribute: condition
  - Attribute: species
  - Attribute: start_timestamp
  - Attribute: duration
  - Attribute: utility_freq
- read_me
  - Attribute: version
- annotations
  - dataset: description
  - dataset: time
  - (⚠ not yet implemented) dataset: reviewer (initials of the user if the annotation has been reviewed)

- time_grades
  - dataset: text
  - dataset: time
  - dataset: duration
  - (⚠ not yet implemented) dataset: reviewer (initials of the user if the annotation has been reviewed)
- sleep_grades
  - dataset: text
  - dataset: time
  - dataset: duration
  - (⚠ not yet implemented) dataset: reviewer (initials of the user if the annotation has been reviewed)
- traces
  - raw
    - dataset: Fp1
      - attribute: sfreq
      - attribute: unit (e.g. uV)
      - attribute: grade (e.g. NOISY)
    - dataset: A_R1
      - attribute: sfreq (in Hz)
      - attribute: unit (string)
      - attribute: grade (string)
    - dataset: A_R2
    - dataset: TRIG
    - ...
  - bipolar
    - scalp
      - [scalp_name] (e.g. "scalp")
        - dataset: Fp2-F10
          - attribute: sfreq (in Hz)
          - attribute: unit (string)
          - attribute: grade (string)
          - attribute: nsamples (int)
          - attribute: processing (string, e.g. "*Re-reference to bipolar; Dampen noisy periods (Hann window); Bandpass filter 0.1-200Hz (FIR filter, firwin design); Notch filter 50Hz and harmonics (FIR filter, firwin design); Decimate to 512Hz; "*)
        - dataset: ...
    - grid
      - [grid_name_1] (e.g. "GridPost")
        - dataset: [channel1-channel2] (e.g. "gp1-gp2")

      - ...
    - [grid_name_2] (e.g. “GridAnt”)
      - ...
  - strip
    - ...
  - lead
    - [lead_name_1] (e.g. “A_R”)
      - dataset: [channel1-channel2] (e.g. “A_R1-A_R2”)
      - dataset: [channel2-channel3] (e.g. “A_R2-A_R3”)
      - ...
    - ...
- preprocessed_sleep
  - scalp
    - scalp
      - dataset: F3-P10
        - attribute: sfreq (in Hz)
        - attribute: unit (string)
        - attribute: grade (string)
        - attribute: nsamples (int)
        - attribute: processing (string)
      - ...
  - bio
    - EMG
      - dataset: EMG1
      - ...
    - ECG
      - dataset: ECG
      - ...

Steps in brief

1. **Re-referencing to bipolar**
   a. Scalp EEG: specific bipolar montage for scalp
   b. Intracranial EEG (grid, strip, lead): for grid, strip and lead, the trace of contact n + 1 is subtracted from the contact n of the same electrode.
   Other traces: for subscalp, bio and misc, re-referencing is currently undefined, resulting in no processing of the traces
   c. ⚠ these traces are not copied into “bipolar” h5 group.
2. **Stimulation artifact removal**

Intracranial EEG (grid, strip, lead): only intracranial traces (grid, strip, leads) are processed to remove stimulation (opening of switch matrix, start of stimulation, closing the switch matrix). Implementation of this step was originally developed by Ellen van Maren in Matlab, with kriging algo; it has been translated to Python and tested by Roland Widmer. Artefact times are hard-coded, absolute delta times based on the annotated trigger times.

   a. ⚠ User need the protocol log from NeLS

Dampen noisy times

3. All bipolar traces (scalp, grid, strip, lead) are set to 0 at timepoints graded as NOISY by a user. For a smooth transition from valid values to 0 values, the Hann window is used, with a half-width of 0.1s.

Bandpass filter

4. All bipolar traces (scalp, grid, strip, lead) are bandpass filtered from 0.1Hz to 200Hz.

Power line noise

5. All bipolar traces (scalp, grid, strip, lead) are notch filtered to power line noise (±0.5Hz) and its harmonics, up to 200Hz (high cut frequency from bandpass filter).

Resample

6. All bipolar traces (scalp, grid, strip, lead) are resampled to 512Hz.

Steps with algo

1. Re-referencing to bipolar

Scalp EEG

For scalp, the bipoles are the following:

Fp2-F10, Fp2-F8, Fp2-F4, F10-T10, T10-P10, P10-O2,
Fp1-F9, Fp1-F7, Fp1-F3, F9-T9, T9-P9, P9-O1,
F8-T4, T4-T6, T6-O2,
F7-T3, T3-T5, T5-O1,
F4-C4, C4-P4, P4-O2,
F3-C3, C3-P3, P3-O1,
Fz-Cz, Cz-Pz, LOC-ROC.

Intracranial EEG (grid, strip, lead)

For grid, strip and lead, the negTrace is subtracted from the posTrace (e.g. contact 8 – contact 9). For a lead of 6 contacts, we have the following (the first contact is at the tip of the lead, the last contact is at the tail, the tail is the closest part to the skull):

contact01 - contact02, contact02-contact03, contact03-contact04, contact04-contact05, contact05-contact06

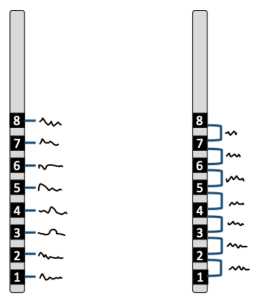


Figure: From Monopolar to Bipolar Montage. During data recording, each channel measures the EEG signal in a **monopolar derivation**, where the signal is referenced to a common physical reference electrode (e.g., on the left mastoid, scalp, or another designated location). This setup captures the absolute voltage at each channel relative to the reference (left panel). In a **bipolar montage**, the signal from each channel within the same electrode lead is re-referenced by subtracting the signal of its immediate neighboring channel. This transformation enhances the spatial resolution of the data and reduces the influence of global artifacts or noise shared across channels (right panel).

Other traces

For subscalp, bio and misc, re-referencing is currently undefined, resulting in no processing of the traces

⚠ these traces are not copied into "bipolar" h5 group.

☝ **if you think that another trace should appear in the bipolar group of traces in h5, please notify it**

Code

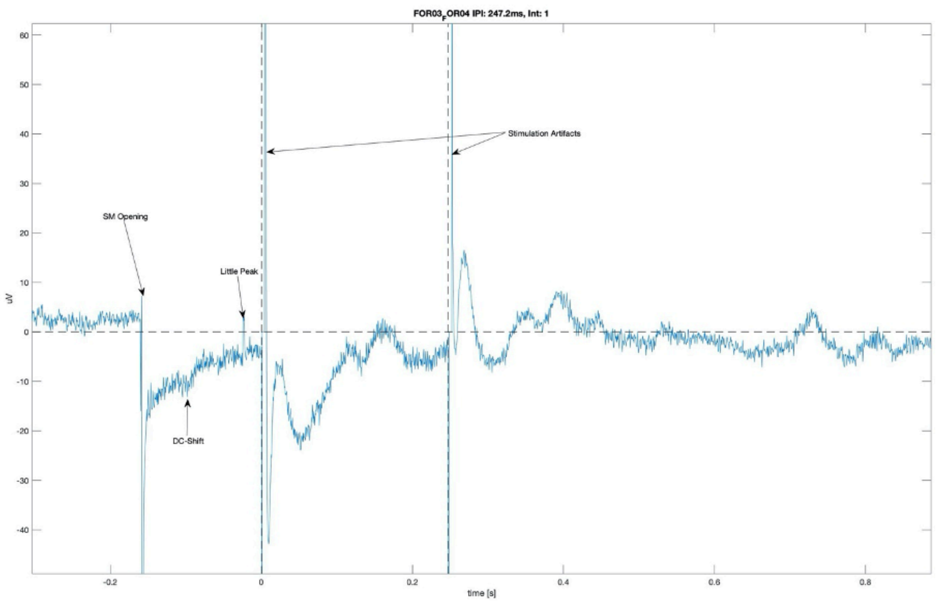
```
data = np.array(dataset_positive_channel[()]) -
np.array(dataset_negative_channel[()])

# Append to attribute processingStep
"Re-reference to bipolar"
```

⚠ the grade attribute attached to the bipolar channel is NOISY if any of positive or negative channel is NOISY, if not NOISY it is ICTAL if any of both channels is ICTAL, if not ICTAL it is IED if any of both channels is IED, if not IED it is NORMAL if both channels are NORMAL, else UNSPECIFIED.

2. Stimulation artifact removal

Intracranial EEG (grid, strip, lead)

Only intracranial traces (grid, strip, leads) are processed to remove stimulation (opening of switch matrix, start of stimulation, closing the switch matrix).



```
def _find_artifact_peak_indices(data: np.ndarray, trig_index_1: int,
trig_index_2: int, f_sample: int) -> tuple[int, int, int]:
# Switch matrix opening artifact, ~160ms before stim
pk_dur = round(0.005 * f_sample)
index_start = round(trig_index_1 - 0.250 * f_sample)
index_stop = round(trig_index_1 - 0.090 * f_sample)
pk = _find_peak_using_line_length(data[index_start:index_stop],
pk_dur)
peak_index_opening = round(pk + index_start)

# Little peak, ~25ms before stim
pk_dur = round(0.005 * f_sample)
index_start = round(trig_index_1 - 0.030 * f_sample)
index_stop = round(trig_index_1 - 0.017 * f_sample)
pk = _find_peak_using_line_length(data[index_start:index_stop],
pk_dur)
peak_index_little_peak = round(pk + index_start)

# Switch matrix closing, ~300ms after stim
pk_dur = round(0.003 * f_sample)
index_start = round(trig_index_2 + 0.240 * f_sample)
index_stop = round(trig_index_2 + 0.500 * f_sample)

if index_stop < len(data):
pk = _find_peak_using_line_length(data[index_start:index_stop],
pk_dur)
peak_index_closing = round(pk + index_start)
else:
```

```
peak_index_closing = round(trig_index_2 + 0.278 * f_sample)

return peak_index_opening, peak_index_little_peak, peak_index_closing


def _interpolate_kriging(complete_channel: np.ndarray, peak_index:
int, peak_window_offset: list[int, int], factor: int):
artifact_length = np.sum(peak_window_offset) + 1

pre_median = np.median(complete_channel[peak_index -
peak_window_offset[0] - 5:peak_index - peak_window_offset[0] + 1])
pre_std = np.std(complete_channel[peak_index - peak_window_offset[0]
- artifact_length - 10:peak_index - peak_window_offset[0] - 10 + 1])
post_median = np.median(complete_channel[peak_index +
peak_window_offset[1]:peak_index + peak_window_offset[1] + 4])

res = np.linspace(pre_median, post_median, num=artifact_length,
endpoint=False)
res += np.random.normal(loc=0, scale=pre_std / factor,
size=res.shape)

return res


def remove_stimulation_artifacts(data: np.array, f_sample: float,
stim_logs: pd.DataFrame, channel_name: str = "") -> None:
"""
Applies artifact removal techniques to EEG data and optionally plots
results.
"""
ttl = stim_logs["TTL"].astype(int).to_numpy()
ttl_pp = stim_logs["TTL_PP"].astype(int).to_numpy()
channel = data.copy()

window_opening_offset = np.round(np.array([0.005, 0.005]) *
f_sample).astype(int)
window_little_peak_offset = np.round(np.array([0.002, 0.005]) *
f_sample).astype(int)
window_closing_offset = np.round(np.array([0.003, 0.003]) *
f_sample).astype(int)
window_stimulation_offset = np.round(np.array([0.002, 0.017]) *
f_sample).astype(int)

trigger_1_indices = ttl
```

```
trigger_2_indices = ttl_pp
n = len(channel)
for i, stim in stim_logs.iterrows():
if stim["name_pos"] + "-" + stim["name_neg"] != channel_name:
# response channel
peak_index_opening, peak_index_little_peak, peak_index_closing =
_find_artifact_peak_indices(channel, trigger_1_indices[i],
trigger_2_indices[i], f_sample=f_sample)

if peak_index_opening - window_opening_offset[0] > 0 and
peak_index_opening + window_opening_offset[1] + 1 < n:
# Remove artifact peaks
channel[peak_index_opening -
window_opening_offset[0]:peak_index_opening +
window_opening_offset[1] + 1] = _interpolate_kriging(channel,
peak_index_opening, window_opening_offset, 3)

if peak_index_little_peak - window_little_peak_offset[0] > 0 and
peak_index_little_peak + window_little_peak_offset[1] + 1 < n:
channel[peak_index_little_peak - window_little_peak_offset[0]:
peak_index_little_peak + window_little_peak_offset[1] + 1] =
_interpolate_kriging(channel, peak_index_little_peak,
window_little_peak_offset, 3)

if peak_index_closing - window_closing_offset[0] > 0 and
peak_index_closing + window_closing_offset[1] + 1 < n:
channel[peak_index_closing - window_closing_offset[0]:
peak_index_closing + window_closing_offset[1] + 1] =
_interpolate_kriging(channel, peak_index_closing,
window_closing_offset, 2)

# Remove stimulations peaks
if trigger_1_indices[i] - window_stimulation_offset[0] > 0 and
trigger_1_indices[i] + window_stimulation_offset[1] + 1 < n:
channel[trigger_1_indices[i] - window_stimulation_offset[0]:
trigger_1_indices[i] + window_stimulation_offset[1] + 1] =
_interpolate_kriging(channel, trigger_1_indices[i],
window_stimulation_offset, 3)

if trigger_2_indices[i] > trigger_1_indices[i] and
trigger_2_indices[i] - window_stimulation_offset[0] > 0 and
trigger_2_indices[i] + window_stimulation_offset[1] + 1 < n:
channel[trigger_2_indices[i] - window_stimulation_offset[0]:
trigger_2_indices[i] + window_stimulation_offset[1] + 1] =
_interpolate_kriging(channel, trigger_2_indices[i],
window_stimulation_offset, 3)
return channel

# Append to attribute processingStep
"Remove stim-artefacts for leads"
```

⚠ User need the protocol log from NeLS
⚠ Stimulation artefact removal doesn't remove the stimulation artefact of stimulated bipole and bipoles sharing a contact with the stimulated bipole. For these traces, the signal cannot be recovered by design. We could think about applying a Hann window similarly to what is done to dampen NOISY periods.
Note: Implementation of this step was originally developed by Ellen van Maren in Matlab, with **kriging algo**; it has been translated to Python and tested by Roland Widmer. Artefact times are hard-coded, absolute delta times based on the annotated trigger times.
Note2: this step is subjected to change regarding how to read log files.

3. Dampen noisy times

All bipolar traces (scalp, grid, strip, lead) are set to 0 at timepoints graded as NOISY by a user. For a smooth transition from valid values to 0 values, the Hann window is used, with a half-width of 0.1s.

```
def filterNoisyEvents(data: np.ndarray, noisyOnsets: np.ndarray,
noisyDurations: np.ndarray,
sFreq: float, windowSize: float = 0.1) -> np.ndarray:
"""
Dampens noisy events within data by setting values of noisy events to
zero and applying Hann windows
(https://en.wikipedia.org/wiki/Hann_function) around starting and
ending points to get smoothed transitions

:param data: 1D data array
:param noisyOnsets: 1D array of noisy events onsets in seconds from
start of recording (t=0s corresponds to data[0])
:param noisyDurations: 1D array of noisy events duration in seconds
:param sFreq: data sampling frequency in Hertz
:param windowSize: length of the smoothed transition periods in
seconds (half of the full Hann window)
:return: data array wih noisy events being zeroed and having smoothed
transition periods
"""
assert len(data.shape) == 1 # 1D data array
assert len(noisyOnsets.shape) == 1 # 1D noisyOnsets array
assert len(noisyDurations.shape) == 1 # 1D noisyDurations array
nSamples = data.shape[0]

# Create Hann window
window = np.rint(windowSize * sFreq).astype(int)
hannWindow = 1 - hann(2 * window)
hannLeft = hannWindow[0:window]
hannRight = hannWindow[window + 1:]

# Create Hann mask
hannMask = np.ones(data.shape[0]+2*window) # add left and right pads
of window length to prevent from borders effect
noisyOnsetsIndices = np.rint(noisyOnsets * sFreq).astype(int)
noisyDurationsIndices = np.rint(noisyDurations * sFreq).astype(int)
for i in range(noisyOnsetsIndices.shape[0]):
eventStart = max(0, min(int(noisyOnsetsIndices[i]), nSamples-1))
eventStop = min(int(noisyOnsetsIndices[i] +
noisyDurationsIndices[i]), nSamples)
eventDuration = max(1, eventStop - eventStart)

hannMask[eventStart:eventStop+2*window-1] = np.concatenate((hannLeft,
np.zeros(eventDuration), hannRight))

mask = hannMask[window:-window] # remove pads to go back to data
array length

return mask * data

# Append to attribute processingStep
'Dampen noisy periods (Hann window)'
```

Note: We apply a **Hann window** to the data to taper the edges of time segments, reducing spectral leakage and minimizing the influence of noise or abrupt transitions. The smooth tapering suppresses high-frequency artifacts introduced by discontinuities, ensuring a cleaner representation of the underlying signals in the frequency domain. This improves the accuracy of subsequent analyses, such as Fourier transforms or time-frequency decompositions.
⚠ Noisy times have to be marked and saved prior to running the bipolar computation

4. Bandpass filter

All bipolar traces (scalp, grid, strip, lead) are bandpass filtered from 0.1Hz to 200Hz, with a FIR filter using firwin design (reproduction of the MNE filter by Riccardo Cusinato). The low cut frequency has been set to 0.1Hz to allow study of slow waves whose range starts at 0.25Hz. The high cut frequency has been set to 200Hz for a resampling set at 512Hz (see below).

```
def _firwinDesign(N, freq, gain, window, sfreq):
"""
Construct a FIR filter using firwin.
From MNE source https://github.com/mne-tools/mne-
python/blob/maint/1.6/mne/filter.py#L474
Author: Riccardo Cusinato, CCN lab, University of Bern
Date: 2024.Jan.17
"""
_length_factors = dict(hann=3.1, hamming=3.3, blackman=5.0)
assert freq[0] == 0
assert len(freq) > 1
assert len(freq) == len(gain)
assert N % 2 == 1
h = np.zeros(N)
prev_freq = freq[-1]
prev_gain = gain[-1]
if gain[-1] == 1:
h[N // 2] = 1 # start with "all up"
assert prev_gain in (0, 1)
for this_freq, this_gain in zip(freq[::-1][1:], gain[::-1][1:]):
assert this_gain in (0, 1)
if this_gain != prev_gain:
# Get the correct N to satisfy the requested transition bandwidth
transition = (prev_freq - this_freq) / 2.0
this_N = int(round(_length_factors[window] / transition))
this_N += 1 - this_N % 2 # make it odd
if this_N > N:
raise ValueError(
"The requested filter length %s is too short "
"for the requested %0.2f Hz transition band, "
"which requires %s samples" % (N, transition * sfreq / 2.0, this_N)
)
# Construct a lowpass
this_h = scipy_signal.firwin(
this_N,
(prev_freq + this_freq) / 2.0,
window=window,
fs=freq[-1] * 2,
)
assert this_h.shape == (this_N,)
offset = (N - this_N) // 2
```

```
if this_gain == 0:
h[offset:N - offset] -= this_h
else:
h[offset:N - offset] += this_h
prev_gain = this_gain
prev_freq = this_freq
return h


def _bandpassFilterAsMNE(l_freq: float, h_freq: float, sfreq: float):
"""
Construct a bandpass FIR filter using firwin.
From MNE source https://github.com/mne-tools/mne-
python/blob/maint/1.6/mne/filter.py#L474
Author: Riccardo Cusinato, CCN lab, University of Bern
Date: 2024.Jan.17

l_freq: highpass freq
h_freq: lowpass freq
sfreq: sampling freq
"""
l_stop_band = min(max(l_freq * 0.25, 2.0), l_freq)
h_stop_band = min(max(h_freq * 0.25, 2.0), sfreq / 2.0 - h_freq)
l_stop = l_freq - l_stop_band
h_stop = h_freq + h_stop_band
filt_length = round(3.3 / min(l_stop_band, h_stop_band) * sfreq)
filt_length += (filt_length - 1) % 2
freq = [l_stop, l_freq, h_freq, h_stop]
gain = [0, 1, 1, 0]
if h_stop != sfreq / 2.0:
freq += [sfreq / 2.0]
gain += [0]
if l_stop != 0:
freq = [0.] + freq
gain = [0] + gain
order = np.argsort(freq)
freq = [freq[o] for o in order]
gain = [gain[o] for o in order]
freq = np.array(freq) / (sfreq / 2.)
h = _firwinDesign(filt_length, freq, gain, "hamming", sfreq)
return h

def bandpassFilterAsMNE(data: [float], sFreq: float, lowpassFreq:
float, highpassFreq: float) -> [float]:
```

```
h = _bandpassFilterAsMNE(highpassFreq, lowpassFreq, sFreq)
return scipy_signal.convolve(data, h, mode='same')

# Append to attribute processingStep
f'Bandpass filter {0.1}-{200}Hz (FIR filter, firwin design)'
```

Note: the **bandpass filter method** from MNE has been rewriten to stay independent from mne package thanks to Riccardo C.. Rewriting an MNE function with SciPy allows us to reduce external dependencies and improve performance by eliminating unnecessary overhead. It gives us full control over the implementation, ensuring it's optimized for our specific needs while keeping the codebase lighter and more maintainable. This also simplifies debugging and enhance long-term flexibility for our project.

5. Power line noise

All bipolar traces (scalp, grid, strip, lead) are notch filtered to power line noise (±0.5Hz) and its harmonics, up to 200Hz (high cut frequency from bandpass filter).

```
def notchFilter(data: [float], sFreq: float, notch: float) ->
[float]:
"""Applies a notch filter to remove a specified frequency and its
harmonics from the input data."""
if notch >= sFreq / 2:
raise ValueError("Notch frequency must be less than half the sampling
frequency.")

filtered_data = mne_filter.notch_filter(data, sFreq, np.arange(notch,
201, notch), method='fir')
return filtered_data

# Append to attribute processingStep
f'Notch filter {50}Hz and harmonics (FIR filter, firwin design)'
```

Note: The MNE implementation of the **notch filter method** is used, with a FIR filter and a sharp transition bandwidth of 1Hz.
Note2: The same comment in **4. bandpass filter** applies here. I will re-implement the MNE function to depend only on numpy and scipy package for notch filtering, with the exact same resulting filtered_data

6. Resample

All bipolar traces (scalp, grid, strip, lead) are resampled to 512Hz.

```
def decimate(data: [float], decimateFactor: int) -> [float]:
```

```
return scipy_signal.decimate(data, q=decimateFactor, ftype='fir',
zero_phase=True)

# Append to attribute processingStep
f'Decimate to {512}Hz'
```

Note: The new sFreq is chosen to be a multiple of the original sFreq (512 Hz for an original sFreq of 1024 Hz, 500Hz for an original sFreq of 1 kHz or 2 kHz), so the **decimate method** can be used. This decimate algo is a FIR filter-based method that reduces the sampling rate by an integer factor, applying a low-pass filter to prevent aliasing. The phase distortion is minimal, with `zero_phase = True`. It is computationally efficient compared to resample and out of the most it maintains time-domain signal characteristics.

Note

- Rescaling is not integrated in pre-processing as the real amplitude of traces may be necessary for some analyses. It can be applied afterwards for projects requiring it.
- Stimulation artefact removal doesn't remove the stimulation artefact of stimulated bipole and bipoles sharing a contact with the stimulated bipole. For these traces, the signal cannot be recovered by design. We could think about applying a Hann window similarly to what is done to dampen NOISY periods. 👆 **what is your point of view?**
- Dampening applied before bandpass filtering: In most scenarios, this sequence ensures better noise suppression and prevents the introduction of artifacts during filtering.
    - Minimizes Filter Artifacts: High-amplitude noise can cause ringing or distortions during filtering. Dampening first reduces the noise amplitude, preventing such artifacts.
    - Preserves Filter Performance: Extreme noise can affect the filter's behavior, particularly for FIR filters with precise frequency responses. Dampening ensures the filter works optimally on meaningful signal components.
    - Smooth Transitions with Hann Window: The Hann window creates smooth, tapered transitions, avoiding sharp discontinuities that might otherwise introduce spurious frequency components.
    - Improves Downstream Signal Quality: Dampening first ensures that only the desired signal is emphasized in the bandpass filtering step, leading to a cleaner and more interpretable final output.

- FIR filters vs Butterworth filter:
  Mike Cohen, Analyzing neural time series data: theory and practice, MIT Press: *For most situations when using the filter-Hilbert method for time-frequency decomposition of EEG data, FIR filters are preferred over IIR filters. FIR filters are more stable and less likely to introduce nonlinear phase distortions. Although the computational cost is a bit higher compared to IIR filters because of FIR's increased filter order (filter order is discussed in section 14.5), in practice this is not a major limitation for modern computers. Most of the rest of this chapter covers FIR filter design and application because of the advantages of FIR over IIR filters. The Butterworth IIR filter is introduced briefly in section 14.7.*