SENSI EEG PREPROC

BAD-CHANNEL DETECTION MODULE

USER MANUAL

AMILCAR J. MALAVE AND BLAIR KANESHIRO

FEBRUARY 2026

Table of Contents

CHAPTER 1

Introduction

Electroencephalography (EEG) recordings may contain channels whose data deviate from expected physiological behavior due to poor contact, motion, hardware instability, or environmental interference. Identifying and removing such 'bad' channels is a critical quality-control step in EEG preprocessing, as even a small number of unstable channels can degrade downstream analyses, particularly spatial-filtering methods such as Principal Components Analysis (PCA) or Reliable Components Analysis (RCA), where spurious spatial covariance introduced by unstable channels can strongly bias component estimation, as well as connectivity estimation [5].

Channel failures manifest in multiple ways, including sustained noise, intermittent high-amplitude transients, abnormal variance, and poor agreement with neighboring electrodes. No single scalar metric reliably captures all of these failure modes, and fully automatic pipelines often either over-reject valid channels or fail to isolate subtle but disruptive artifacts [2]. As a result, bad-channel identification in practice is often dominated by manual review. Users are frequently required to inspect channels one by one, compare them against neighbors, and directly input their decisions into user interfaces or preprocessing scripts. This process introduces subjectivity and is labor-intensive, time-consuming, and difficult to scale to high-channel-count datasets. Moreover, repeated code editing during preprocessing not only slows analysis but also increases the risk of accidental errors and inconsistencies across subjects or sessions.

A major contributor to this manual burden is the difficulty of identifying channels that share similar artifact structure. Channels affected by the same underlying issue (such as electrode motion, eye activity, or global transients) often exhibit correlated abnormal behavior, but these relationships are not easily detected when channels are reviewed individually. As a result, users spend substantial time searching for patterns that are present in the data but not explicitly exposed by standard preprocessing tools.

1.1 Module Background and Design Motivation

This Bad-Channel Detection Module was developed as part of the Stanford Educational Neuroscience Initiative (SENSI) EEG preprocessing framework, *SENSI EEG PREPROC*, with the explicit goal of reducing manual labor and minimizing the need for users to modify code directly while cleaning EEG data. Rather than relying on repeated script edits or ad-hoc channel lists, the module provides a structured workflow in which automated analyses surface candidate problem channels and final decisions are made through an interactive interface. Another motivation for this module was the limitation of conventional bad-channel indicators when used in isolation. Metrics such as neighbor correlation, maximum amplitude, or overall variance often capture only static or global properties of channel quality. In practice, however, channel behavior can fluctuate over time. To address this, the module incorporates time-resolved versions of neighbor-correlation and variance metrics, allowing transient or intermittent failures to be identified more reliably. A further challenge addressed by the module is the interpretation of high-amplitude transient activity. Not all high-amplitude transients necessitate the removal of the implicated channel(s); when such events occur synchronously across multiple electrodes, they often reflect physiological activity — such as eyeblinks — or global artifacts that can later be isolated through other established approaches such as Independent Components Analysis (ICA) [1]. Automatically removing every channel that exhibits large transients is therefore undesirable.

To resolve this ambiguity and reduce review time, this module implements an automated clustering procedure that groups channels according to the temporal similarity of their high-amplitude transient events. By organizing channels with shared abnormal structure into clusters, the module allows users to review groups of related channels together, rather than inspecting each channel independently. This design choice substantially reduces manual effort while improving the consistency and transparency of bad-channel decisions, shifting manual review from exhaustive channel-by-channel inspection to targeted, cluster-level evaluation.

1.2 Conceptual Overview of the Workflow

At a high level, the bad-channel detection process follows five stages:

1. **Feature extraction per channel.** Multiple complementary features are computed to capture different channel failure modes, including amplitude extremes, signal variability, and disagreement with neighboring electrodes.
2. **Channel Scoring and Thresholding.** Channels are scored based on their extracted features, and are assigned an initial status of *good*, *suspicious*, or *bad*.
3. **Clustering of similar high-amplitude transient patterns.** High-amplitude transient events are identified using robust z-scoring and compared across channels based on their temporal co-occurrence. Channels exhibiting similar transient patterns are grouped together, revealing collections of electrodes affected by shared artifact structure (e.g., eye-related activity, motion artifacts, or globally unstable behavior).
4. **Preliminary classification.** Channel classifications are updated based on cluster-level context. These labels are passed to the manual review stage and are intended to prioritize user attention rather than to enforce automatic rejection.
5. **Interactive review and final decision.** An interactive user interface presents channels and clusters for inspection, allowing the user to efficiently confirm, override, or refine automated classifications before the list of bad channels is finalized.


Automated scores and clusters (Step 4) are **diagnostic tools, not final decisions**.

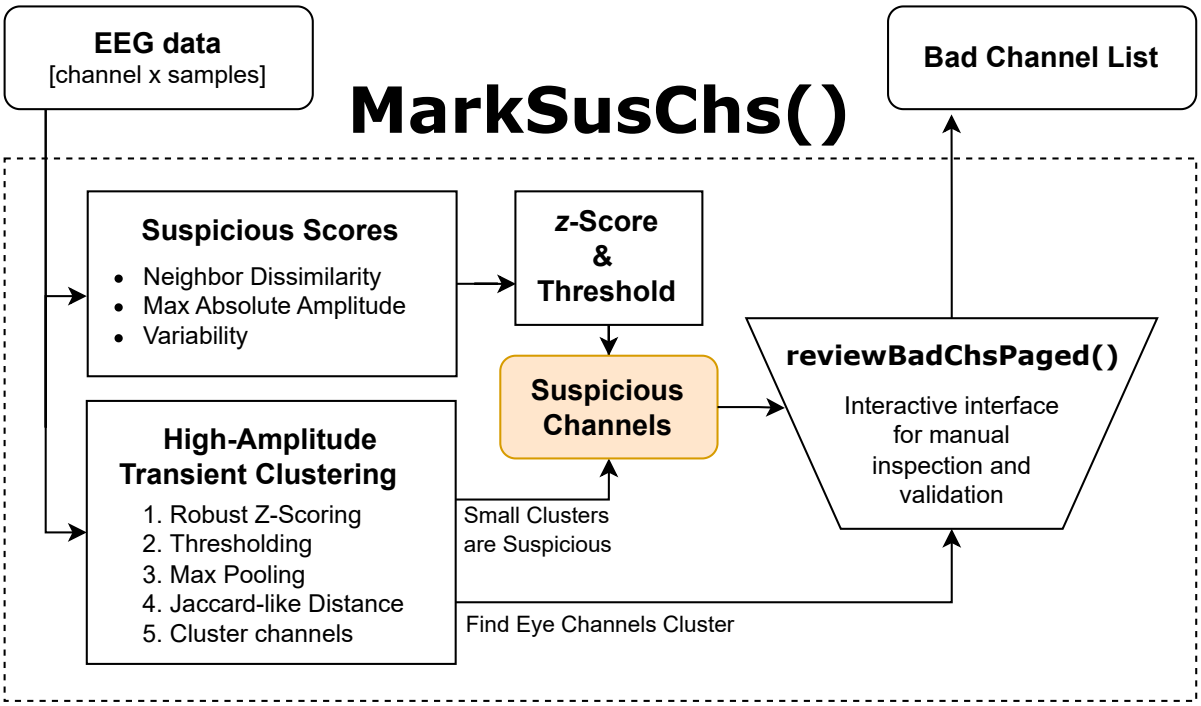


Figure 1: **Overview of the Bad-Channel Detection Module.** EEG data undergo automated feature-based scoring and clustering to identify suspicious channels. The user then reviews and confirms channel status via an interactive interface before finalizing the bad-channel list.

Channel Status Definitions

Throughout the Module and user interface, channels are assigned one of three statuses:

- **Good.** Channels that behave consistently with their neighbors and exhibit no abnormal feature values.
- **Suspicious.** Channels that exhibit one or more abnormal features but may still reflect physiological activity (e.g., eye movements or task-related structure) and/or artifacts that can be removed by other means (e.g., ICA).
- **Bad.** Channels dominated by non-physiological artifacts or instability and recommended for removal.

Only channels explicitly marked as **bad** during the interactive review are returned in the final bad-channel list.

Role of Clustering

Clustering is used to expose structure, not to automatically classify channels. Channels are grouped based on the similarity of their transient event timing, which often reveals mean-

ingful patterns such as eye-related activity or shared noise sources. A cluster may contain entirely good channels, entirely bad channels, or a mixture of both. Final classification always remains a user decision.

1.3 Scope and Intended Use

This Module is intended for use as a quality-control step in EEG preprocessing, typically after basic filtering and before spatial decomposition methods such as ICA. Thresholds and parameters are heuristic by design and may require tuning depending on recording conditions and montage density.

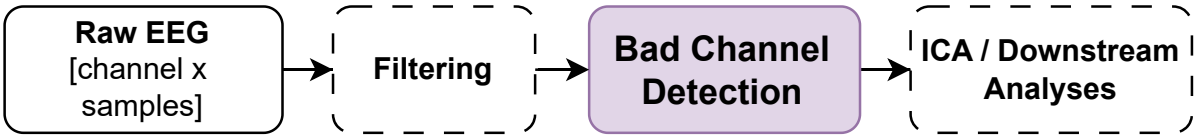


Figure 2: **Intended use.**

Design Philosophy

The central design principle of this module is **guided efficiency**: automated metrics reduce the search space, clustering reveals shared artifact structure, and the interactive interface minimizes code modification while preserving user control. This balance allows the preprocessing workflow to scale to large datasets while reducing manual effort and the likelihood of accidental errors.

1.4 Acknowledgments

The authors thank members of SENSI and collaborating labs at Stanford University for their feedback and early testing of the Module. We especially acknowledge Philip Hernandez, Neha Rajagopalan, and Madison Bunderson for their insights delivered from having performed manual review and classification of bad channels in SENSI's EEG datasets. Philip Hernandez additionally provided detailed feedback on early algorithm behavior, including the rationale for channel rejection and identification of failure modes that informed subsequent refinements.

This User Manual was prepared using the *Extensive LaTeX Guide* Overleaf template by Armin Dubert.[1]

[1] https://www.overleaf.com/articles/extensive-latex-guide/vdgrdqdwjhtk

Author Information and Contributions

- **Amilcar J. Malave** — Graduate School of Education, Stanford University, Stanford, CA, USA
- **Blair Kaneshiro** — Graduate School of Education, Stanford University, Stanford, CA, USA

Both authors approve the release of the code package and contributed as follows:

- Designed the algorithm and implemented the code: AJM.
- Validated and integrated into larger *SENSI EEG PREPROC* pipelines: AJM, BK.
- Documented the Module: AJM.
- Created illustrative analyses: AJM.
- Wrote the preprint: AJM.
- Provided supervision, feedback, and editing: BK.

Support

For questions, comments, or feature requests, please contact Amilcar Malave <amilcar.malave1997@gmail.com> and Blair Kaneshiro <blairbo@stanford.edu>.

Declaration on the Usage of AI

The authors used OpenAI's GPT-5 and Google's gemini-2.5-flash models for assistance with documentation formatting, function summaries, LaTeX structure, and for brainstorming and exploring conceptual ideas. Additionally, these tools were used to improve flow, word choice, and structure of the User Manual and preprint. No AI tools were used for data analysis, algorithm development, evaluation, or initial drafts of the User Manual or preprint.

License

SENSI EEG PREPROC Bad-Channel Detection Module is released under the MIT License,[2] as follows:

[2] https://choosealicense.com/licenses/mit/

If a code file within the toolbox originated from outside of "*SENSI EEG PREPROC* Bad-Channel Detection Module" and already contained a license, that original license is retained.

Citing the Module

The Bad-Channel Detection Module release comprises two deliverables: the codebase [10] with examples [11] and a preprint summarizing the release [cite when ready].

If using the Bad-Channel Detection Module or the associated figures or methods, please cite the following:

- **GitHub repository**
  Malave, A. J., & Kaneshiro, B. (2025). Bad-Channel Detection Module (v1.1): A MATLAB framework for semi-automated EEG bad-channel detection and review. Stanford University.
  https://github.com/edneuro/SENSI-EEG-Preproc-bad-ch
- **Release Preprint**
  Malave, A. J., & Kaneshiro, B. (2025). ———— .

release project on github and update

add preprint

If using the example data accompanying this software release in other projects, please cite the following:

- **Dataset**
  Malave, A. J. (2025). Example EEG data for the SENSI EEG PREPROC Bad-Channel Detection Module [Data set]. Stanford Digital Repository.
  https://doi.org/10.25740/dg856vy8753

CHAPTER 2

Getting Started

2.1 Toolbox and Resources

2.1.1 GitHub Repository

The *SENSI EEG PREPROC* Bad-Channel Detection Module is available as a publicly accessible GitHub repository.[3] The citation for the repository is associated with the linked instance on the Stanford Digital Repository. As noted in Chapter 1.4, this Module is published under an MIT license.

2.1.2 User Manual and Preprint

The *SENSI EEG PREPROC* Bad-Channel Detection Module release includes two forms of written documentation.

First, the User Manual (this document) is provided in the GitHub repository. It offers comprehensive documentation of the Module, including a brief background on bad-channel selection, guidance on getting started, a walkthrough of the code, illustrative analyses, and detailed documentation of the main and helper functions.

Next, the preprint of the release is available on bioRxiv [cite here when ready]. It provides a more narrative overview of the Module's structure, functions, and illustrative analyses.

add preprint

[3] https://github.com/edneuro/SENSI-EEG-Preproc-bad-ch

2.1.3 Example Data

Due to file size, example data files used in the illustrative analyses are not included on this Module's repository on GitHub and hence this folder contains only a `readme.txt` file. They are instead provided as a separate dataset through the Stanford Digital Repository (SDR) [11].[4]

The example script in the main folder of the repository **`(example.m)`** downloads the data files from their SDR URLs into the **ExampleData** folder in the user's local instance of "SENSI EEG PREPROC Bad-Channel Detection Module". More information on the example data files is provided in the README of the online dataset.

For more information, the downloading procedure is provided in Section 4.1.

2.2 Setup and Installation

2.2.1 Operating Systems and MATLAB versions

This Module was validated on both Windows and Linux operating systems, and we anticipate that the current release will work similarly across operating systems, but this has not been comprehensively validated.

This Module was developed and tested on MATLAB R2024b. The Module may work with earlier versions of MATLAB, but it has not been fully tested on previous versions.

2.2.2 Dependencies

The software requires one MATLAB toolbox: Statistics and Machine Learning Toolbox[5].

Run the following command in MATLAB to determine whether the Toolbox is installed:

```
>> ver
```

This lists all installed toolboxes.

2.2.3 Installation

Clone or download the latest version of the SENSI EEG PREPROC Bad-Channel Detection Module at: https://github.com/edneuro/SENSI-EEG-Preproc-bad-ch

[4] https://purl.stanford.edu/dg856vy8753
[5] https://www.mathworks.com/products/statistics.html

After downloading the codebase, the user can run the following in the MATLAB command window to add the entire Module to their **path**:

```
> CODEBASE_PATH = 'path/to/your/directory/in/char/format';
> addpath(genpath(CODEBASE_PATH));
```

Some of the path specifications in **example.m** are relative to the current working directory. If absolute folder paths are not specified (described later), the user must ensure that MATLAB is operating from the Module's main folder.

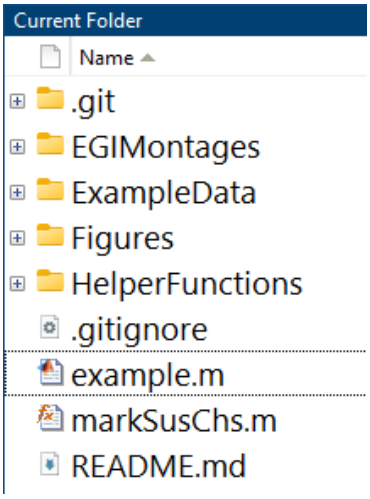


Figure 3: **Current folder.** Ensure that the MATLAB current folder is set to the module directory when running `example.m`.

2.3 Using the Module

This Module is function-based and does not include a full graphical interface. All operations are carried out through MATLAB functions, with an interactive review user interface (UI) used only for the final bad-channel decisions. Users should be comfortable calling functions from the MATLAB command window or from their own preprocessing scripts.

2.3.1 Folder Overview

The top-level folder of the Module contains the following directories and files:

- ***EGIMontages/*** — Contains the montage-specific `neighboringElectrodesXXX.mat` files (where `XXX` is the number of channels) used for neighbor–dissimilarity computation. These files are selected automatically based on the number of channels in the data. Details on the neighbor definitions are provided in subsection 2.5.

- ***ExampleData/*** — Contains only a `readme.txt` file by default. Example datasets used by `example.m` are downloaded into this folder automatically.
- ***Figures/*** — Default folder used by `example.m` to save generated figures.
- ***HelperFunctions/*** — Contains all helper functions used by `markSusChs()`. These functions are not typically called directly by the user and are documented separately in Chapter 6.
- ***example.m*** — A complete example script showing how to configure the Module, run `markSusChs()`, and inspect the outputs. This is the recommended starting point for new users.
- ***markSusChs.m*** — The main function of the Module. This function computes bad-channel scores, performs threshold-based clustering, and launches the interactive review UI for final decisions.
- ***README.md*** — A brief overview of the Module.
- ***LICENSE*** — License for the repository (MIT license).
- **User Manual.pdf** (this file) — User Manual for the current release.
- ***.gitignore*** — Git configuration (no user action required).

2.3.2 Main and Helper Functions

Only one function is intended to be called directly by the user:

`markSusChs()` — Main entry point of the Module.

Syntax

```
[susMask, badChList, plotStruct] = markSusChs(xIn, fs, INFO, EOG, saveFigs, ...
saveFolder, saveName)
```

All other functions in **HelperFunctions/** are internal helper functions called by `markSusChs()` during windowing, feature computation, thresholding, clustering, and figure generation. These helper functions are documented in Chapter 6.

Flags, clustering, and review workflow. Beyond returning channel labels, `markSusChs()` produces a set of user-facing diagnostic figures. The *general-check flags plot* summarizes multiple feature-based indicators of abnormal channel behavior to guide user attention. Detailed guidance on how to interpret each flag is provided in Section 3.5.

Final channel decisions are made during the interactive review stage, where flagged channels are presented in cluster-aware groups for direct inspection of time-domain signals. The

interactive review interface and decision controls are described in Section 3.6. Only channels explicitly confirmed as *bad* during this review are returned in the final bad-channel list.

2.4 Input and Output Specifications for markSusChs()

2.4.1 Input Data Requirements

This Module functions operate on variables that are already loaded to the workspace (rather than filenames). It is therefore the responsibility of the user to have the necessary data already loaded and correctly formatted.

The markSusChs() function operates on continuous or epoched EEG data. In larger *SENSI EEG PREPROC* data-cleaning pipelines that are under development, the function operates on data that have already undergone basic filtering and epoching steps.

Inputs

markSusChs() operates on variables already loaded into the workspace. The function expects the following inputs:

- xIn — EEG data matrix of shape [channels × samples] (in μV).
- fs — Sampling rate in Hz. Example: fs = 250.
- INFO — Configuration struct containing the configuration fields. **The mathematical rationale for these values is discussed in section 3. Guidelines for selecting appropriate values are provided in subsection 4.2. However, the following values serve as a good starting point:**
  - INFO.badCh = [ ]; (Initialize bad-channel list)
  - INFO.badChs.winSec = 10; (Time window (s) for time-dependent features)
  - INFO.badChs.hopSec = 10; (hop size (s))
  - INFO.badChs.win_sec = 0.200; (Window length (s) for max-pooled)
  - INFO.badChs.eps = 0.8; (Distance matrix clustering threshold)
  - INFO.badChs.minClusterUI = 7; (min cluster size for suspicious tag)
  - INFO.badChs.alpha = .25; (plot transparency for overlays)
  - INFO.badChs.nCols = 2; (number of columns in UI plotting layout)
  - INFO.badChs.ref; (Ref channel for visual comparison)
  - INFO.badChs.chMax = 1000; (bad-channel amplitude cutoff µV)

  - `INFO.badChs.neighborDissThresh = 0.3;` (neighbor dissimilarity threshold)

- `EOG` — Specification of eye channels, given as either:
  - a vector of channel indices (e.g., `[125 126]`), or
  - a struct with fields (e.g., `EOG.v`, `EOG.h`)
- `saveFigs` — Logical flag controlling whether figures are saved (e.g., `saveFigs = 1`).
- `saveFolder` — Output folder path for saved figures.
- `saveName` — Base filename used when saving figures (e.g., `saveName = 'test'`).

Neighbor Maps. In addition to these explicit inputs, the Module also requires a valid neighbor–electrode definition for the current montage. For standard EGI HydroCel nets, the appropriate `neighboringElectrodesXXX.mat` file (where `XXX` specifies the number of electrodes) is loaded automatically from the `EGIMontages/` folder based on the number of channels in `xIn`. Users working with non-EGI or custom montages must provide a compatible neighbor file, as described in subsection 2.5.

2.4.2 Outputs from `markSusChs()`

- `susMask` — A vector of length *nCh*, the elements of which contain the following values: 0 = good | 1 = suspicious | 2 = bad
- **badChList** — Indices of channels marked as bad after UI review.
- `plotStruct` — Struct containing per-channel metrics, clustering assignments, thresholded features, and information required for the diagnostic figures.

2.5 Neighbor Electrode Definitions

This Module computes a neighbor–dissimilarity metric, which requires that each electrode be assigned a set of spatial neighbors. These neighbor lists are stored in montage-specific `.mat` files that are loaded automatically during execution.

Location of Neighbor Files. These files are located here:

```
EGIMontages/
```

During execution, the Module selects the correct file based on the number of channels in the input data.

How the Neighbor Files Were Created. The neighbor relationships were derived by hand via inspection of actual EGI HydroCel [9, 6] Sensor Nets and BioSemi Headcaps [3]; this approach was found to produce more accurate neighbor assignments than inspection of sensor layout documents. Electrodes were assigned neighbors according to immediate spatial adjacency. Relevant manufacturer documentation for each system is available online.[6] [7]

Each file follows the naming pattern:

`neighboringElectrodesXXX.mat`

where `XXX` is the number of channels (e.g., `neighboringElectrodes128.mat` is for the EGI 128-channel net). The file contains a cell array named `neighbors` of size $[nCh \times 1]$, where each element is a vector listing the neighbors of that electrode.

For example:

```
neighbors = {
    [8; 2; 122; 121; 125];     % neighbors of channel 1
    [1; 8; 9; 3; 123; 122];    % neighbors of channel 2
    ...
};
```

Available Montages. Currently the Module supports the following montages:

EGI 128-channel HydroCel Sensor Net. Data are acquired from 129 channels including the vertex reference. The following `.mat` files correspond to or are derivatives of this montage:

- `neighboringElectrodes124.mat`: All 128 electrodes (no vertex), with the four electrodes on the face (125–128) omitted.
- `neighboringElectrodes125.mat`: All 128 electrodes plus the vertex reference (129), with the four electrodes on the face (125–128) omitted.
- `neighboringElectrodes126.mat`: All 128 electrodes (no vertex), with the two cheek electrodes (126 and 127) omitted.
- `neighboringElectrodes127.mat`: All 128 electrodes plus the vertex reference (129), with the two cheek electrodes (126 and 127) omitted.
- `neighboringElectrodes128.mat`: All 128 electrodes (no vertex).
- `neighboringElectrodes129.mat`: All 128 electrodes plus the vertex reference (129).

[6] Electrical Geodesics, Inc. (2007). *HydroCel Geodesic Sensor Net Technical Manual*, Appendix B
[7] BioSemi *Headcap Specifications and Electrode Layouts*: https://www.biosemi.com/headcap2.htm.

EGI 256-channel HydroCel Sensor net. Data are acquired from 257 channels including the vertex reference. The following `.mat` files correspond to or are derivatives of this montage:

- `neighboringElectrodes256.mat`: All 256 electrodes (no vertex).
- `neighboringElectrodes257.mat`: All 256 electrodes plus the vertex reference (257).

Note: The original neighbor assignments for the 128-channel and 256-channel EGI nets can be found in the files `createNeighboringElectrodesFrom129.m` and `createNeighboringElectrodesFrom257.m`, respectively. Additional derivative EGI montages and supporting files can be accessed in an external GitHub repository.[8]

BioSemi 64-channel cap. The file `neighboringElectrodes64.mat` contains the neighbor-electrode assignments for the BioSemi 64-channel cap. The script `createNeighboringElectrodesFrom64.m` also contains the neighbor assignments and steps to save out the `.mat` file.

Easycap 74-channel cap. The file `neighboringElectrodes74.mat` contains the neighbor-electrode assignments for the 74-channel Easycap montage corresponding to the EEG data reported by Cichy and Pantazis [4].

Using a Different Montage. If the user employs a sensor montage not included in the code package, a new neighbor file must be created. The required format is a MATLAB cell array named `neighbors`, with one entry per channel:

```
neighbors{ch} = [list of neighbor indices];
```

Once created, the file should be saved as:

```
neighboringElectrodesXXX.mat
```

where `XXX` is the number of channels (e.g., `neighboringElectrodes32.mat`).

Where to Place Custom Files. Custom neighbor files should be placed in the `EGIMontages/` folder (or in a custom folder inside the Module directory). The user must ensure that it replaces any existing file for that montage; otherwise, the Module may load the wrong montage file.

During execution, `neighborCorrFromWindows()` (helper function inside `markSusChs.m`) searches for a file whose name matches the channel count of the input data. If no matching file is found, the Module returns an error prompting the user to supply a valid neighbor-definition file.

[8] https://github.com/blairkan/BKanMatEEGToolbox/tree/master

The loading syntax is explained in subsection 6.2.

CHAPTER 3

markSusChs

3.1 Overview

This chapter details the full processing workflow implemented by the `markSusChs()` function. The goal is to provide a conceptual and mathematical explanation of how suspicious and bad channels are detected, clustered, and reviewed. Each processing stage corresponds to the subsections that follow, beginning with a function-level overview and continuing with mathematical formulations of the feature-based flagging and clustering procedures.

`markSusChs()`

Syntax

```
[susMask, badChList, plotStruct] = markSusChs(xIn, fs, INFO, EOG, saveFigs, ...
saveFolder, saveName)
```

Description. This is the main entry point of the Bad-Channel Detection Module. It automates the detection of suspicious and bad EEG channels through multiple quantitative metrics, spatial clustering, and an interactive user review interface. The function computes per-channel quality features; identifies potential artifacts through thresholding; groups channels exhibiting similar high-amplitude transients; and guides the user through a confirmation interface before finalizing the bad-channel list.

Input and Output specifications can be found in subsection 2.4

Notes.

- Data must be shape [channels × time].
- Channels in "Eye Cluster" are never automatically flagged as bad.
- Thresholds (`neighborDissThresh`, `chMax`, etc.) are empirically derived based on observed EEG signal characteristics and may be adjusted for different datasets.

`markSusChs()` Processing Flow

Figure 4 illustrates the full data flow implemented by `markSusChs()`. EEG data are first segmented into windows and analyzed through feature-based metrics to derive a suspiciousness mask. Channels exhibiting extreme deviations are flagged as potential outliers. A second stage identifies clusters of channels based on co-occurrence of high-amplitude transients. These clusters, along with an automatically identified eye cluster, are then presented to the user for visual confirmation through an interactive interface. The finalized bad-channel list is stored alongside all intermediate metrics in `plotStruct`.

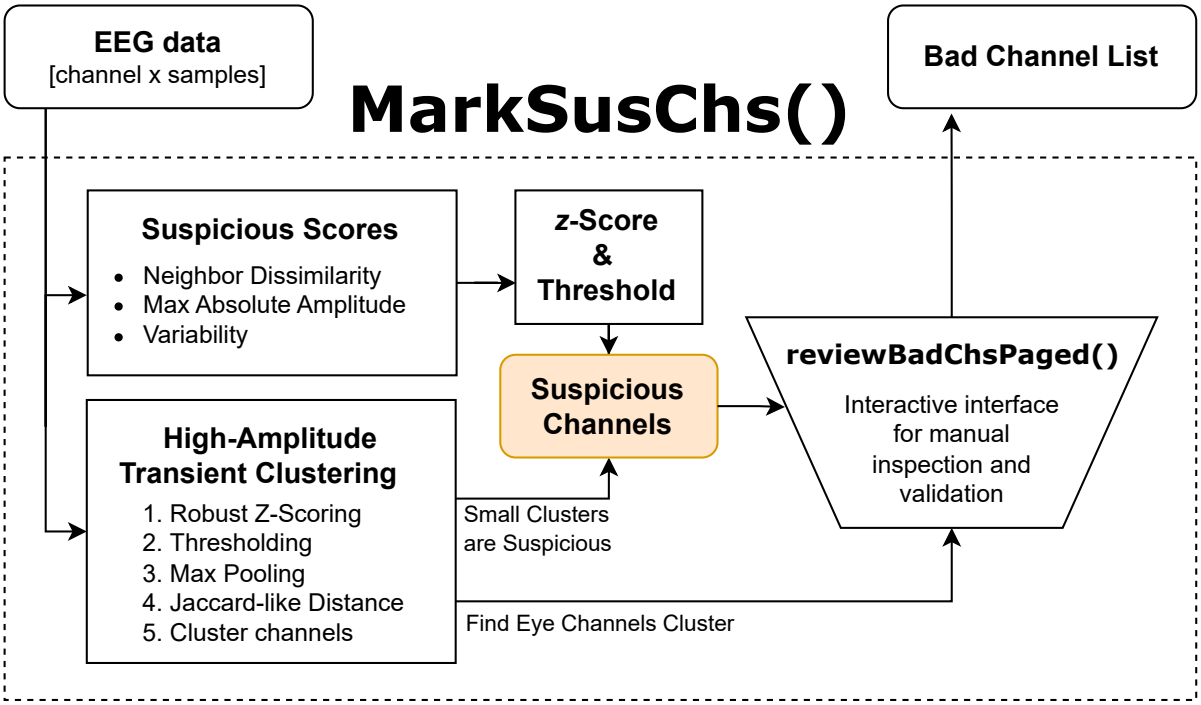


Figure 4: **Processing flow of `markSusChs()`.** EEG data are evaluated through multiple quantitative metrics, clustered by "high-amplitude transient" similarity, and reviewed interactively before generating the final bad-channel list.

3.2 Suspicious Scores (General Checks)

The first stage of `markSusChs()` evaluates each channel using several feature-based checks that are sensitive to different failure modes:

1. Temporal dissimilarity with spatial neighbors.
2. Large transient amplitudes.
3. Abnormal variance (overall and time resolved).

Each feature produces a per-channel *score* that is then transformed and thresholded. Channels with at least one non-zero feature score are initially tagged as *suspicious*; channels with extremely large amplitudes are directly tagged as *bad*. Eye channels, as specified in the `EOG` input, are never forced to bad, and are at most labeled suspicious. The resulting integer mask $\mathbf{s} = [s_1, \ldots, s_C]$ with values $\{0 = \text{good}, 1 = \text{suspicious}, 2 = \text{bad}\}$ forms the preliminary `susMask` that is later refined by clustering and interactive review.

To describe this stage, we first detail each feature and its rationale, and then explain how the feature scores are standardized, thresholded, and combined into the per-channel flags.

3.2.1 Neighbor Dissimilarity

Rationale. EEG channels that lose contact, experience localized noise, or become unstable often drift away from the spatial pattern of their neighboring electrodes. Because most neural activity has a smooth spatial structure across the scalp, a consistently decorrelated channel is a strong indicator of poor signal quality [8]. In long recordings, this decorrelation may be intermittent. For this reason, the Module computes *time-resolved* (per window) neighbor-based measures.

Time-resolved spatial neighbor dissimilarity. The input data $\mathbf{X} \in \mathbb{R}^{C \times T}$ are first divided into overlapping windows using `makeWindowsFromX()`, yielding $\mathbf{X}^{(w)}$ for windows $w = 1, \ldots, W$.

For each channel c and window w, `neighborCorrFromWindows()` computes the median correlation between channel c and each of its spatial neighbors. The median is used to minimize the influence of corrupted or noisy neighboring channels. Let $\mathcal{N}_c$ be the neighbor set defined by the EGI montage, then

$$r_{c,w}^{(j)} = \operatorname{corr}\left(x_c^{(w)},\, x_j^{(w)}\right), \qquad j \in \mathcal{N}_c,$$

and the per-window neighbor similarity is the median of these pairwise correlations:

$$\rho_{c,w} = \operatorname{median}_{j \in \mathcal{N}_c} r_{c,w}^{(j)}.$$

Neighbor *dissimilarity* is then defined as

$$d_{c,w} = 1 - |\rho_{c,w}|.$$

These values are then smoothed across windows using `smoothFeature()`, which applies a moving median and and limits the influence of extreme values:

$$\tilde{d}_{c,w} = \mathrm{Smooth}\big(d_{c,w}\big),$$

where Smooth$(\cdot)$ denotes the moving-median operation implemented in `smoothFeature()` along the window dimension.

Rationale for smoothing the dissimilarity measure. Window-level neighbor dissimilarity is informative but noisy. Even with good electrodes, windowed correlations fluctuate due to limited window length and customary EEG nonstationarity, which together impose small jitter in the pairwise neighbor correlations computed by `neighborCorrFromWindows()`. These fluctuations can obscure the underlying pattern of whether a channel truly decorrelates from its neighbors.

To stabilize the measure and emphasize only meaningful, persistent deviations, the Module applies a moving-median filter across windows using `smoothFeature()`. This smoothing suppresses small random fluctuations, and reveals the true trend of neighbor dissimilarity.

Figure 5 illustrates the effect. Panel A shows raw dissimilarity values $d_{c,w}$, which exhibit high variability. Panel B shows the smoothed series $\widetilde{d}_{c,w}$, where persistent deviations become clearer and suitable for channel-quality scoring.

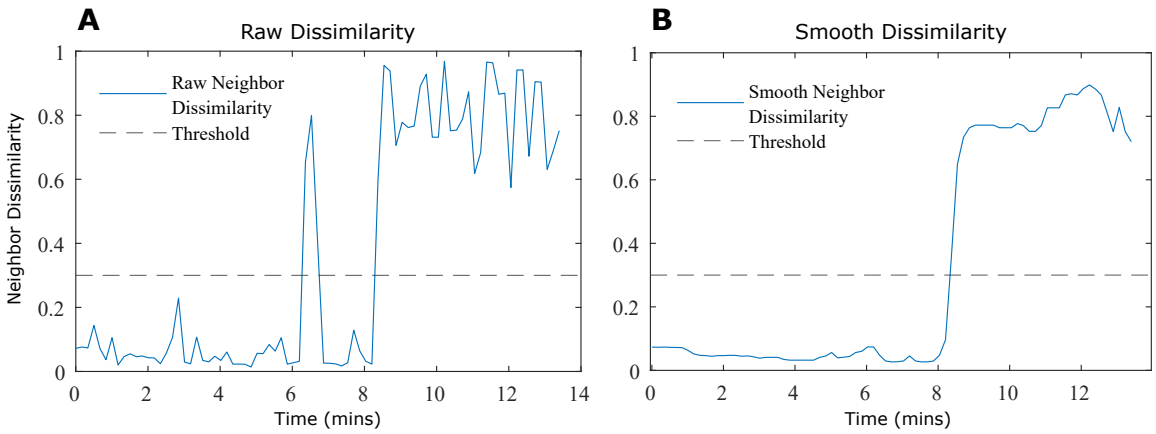


Figure 5: **Effect of smoothing on windowed neighbor dissimilarity. (A)** Raw window-level dissimilarity shows high variability due to finite window effects and natural EEG signal characteristics. **(B)** After applying the moving-median smoothing from `smoothFeature()`, transient noise is suppressed and sustained decorrelations emerge clearly. This example shows the channel became decorrelated with its neighbors around 8 mins into the recording, and stayed decorrelated until the end of the session. The threshold (set at 0.3) indicates which channels will be tagged as suspicious.

Aggregated neighbor-dissimilarity score. After smoothing, the dissimilarity series $\widetilde{d}_{c,w}$ is thresholded to retain only windows that show meaningful decorrelation. Values below the threshold (0.3) are suppressed:

$$\widehat{d}_{c,w} = \begin{cases} 0, & \widetilde{d}_{c,w} < 0.3, \\ \widetilde{d}_{c,w}, & \text{otherwise.} \end{cases}$$

To make the suspicious tagging maximally sensitive, **this Module marks a channel as suspicious if *any* window exceeds this threshold.** This design intentionally biases the procedure toward over-detection. The interactive review interface (Section 6.1) allows fast visual confirmation, so the cost of flagging extra channels is low, while missing a problematic channel would be more detrimental.

Suspicious Score. A channel is considered *suspicious* by this feature if its smoothed neighbor dissimilarity exceeds the threshold in *any* time window:

$$\widetilde{d}_{c,w} \geq 0.3 \text{ for any window } w \quad \Longrightarrow \quad \text{isSuspicious}(c) = 1.$$

The continuous aggregated neighbor-dissimilarity measure D_c summarizes how strongly each channel deviates from its spatial neighbors across time. It is computed as the mean

of the smoothed window-level dissimilarity and provides a stable, interpretable indicator of channel–neighbor mismatch:

$$D_c = \frac{1}{W} \sum_{w=1}^{W} \widehat{d}_{c,w},$$

where channels with large D_c values are more likely to be "bad".

Although D_c is not used directly in the automatic suspicious-channel classification, D_c values are displayed in the summary plots and serve as an important visual cue during the interactive review, where they help guide final user decisions.

3.2.2 Maximum Absolute Amplitude

Rationale. Some electrodes produce intermittent high-amplitude spikes even after filtering, usually due to unstable contact, motion, cable noise, or impedance changes. Anecdotally, we have found that large-amplitude transients hinder the performance of downstream ICA calculations; this is likely due to distortion of the statistical structure that ICA and other multivariate decomposition methods rely upon.[9] For this reason, the software includes steps to identify large-amplitude channels.

add reference to footnote

For each channel c, the maximum absolute amplitude is computed as

$$a_c = \max_t |x_c(t)|.$$

A ***direct* bad rule** uses this raw amplitude:

$$\text{if} \quad a_c \geq \texttt{chMax} \quad \Rightarrow \quad \text{isBad}(c) = 1.$$

where `chMax` is typically set to 1000 μV (*SENSI* value for data collected using the EGI system).

Suspicious Score. As maximum amplitudes follow a heavy-tailed distribution, the values are first log-transformed before z-scoring:

$$\ell_c = \log(a_c).$$

These log-amplitude values are standardized across channels,

$$z_c^{(\max)} = \frac{\ell_c - \mu_\ell}{\sigma_\ell},$$

where μ_ℓ and σ_ℓ are the mean and standard deviation, respectively, of the log-amplitudes across all channels.

[9] In particular, ICA typically begins by whitening the data using the covariance matrix. Channels with extreme amplitudes inflate their variance and dominate this covariance, forcing ICA to assign components to model these non-physiological artifacts instead of separating real neural or ocular/muscle sources.

Only outliers are relevant for detecting problematic channels. A channel is therefore considered *suspicious* by this feature:

$$\text{if} \quad z_c^{(\max)} \geq 2 \quad \Rightarrow \quad \text{isSuspicious}(c) = 1.$$

3.2.3 Overall Variability and Temporal Variability

Rationale. Healthy EEG channels typically exhibit moderate and relatively stable variability. Very low variability suggests a nearly flat or disconnected electrode, while very high variability suggests excessive noise or unstable impedance [8]. Moreover, some channels alternate between reasonable and pathological variability over time, e.g., when sensors are intermittently bumped or a lead is loose. The Module therefore evaluates both of the following:

1. *Overall variability*, computed across the entire recording, and
2. *Temporal variability*, based on how the variance of each channel changes across windows.

Overall variability. For each channel c, the variance over the full recording is

$$v_c = \text{Var}[x_c].$$

Purpose of temporal variability. Temporal variability evaluates whether a channel maintains internally consistent signal properties across the recording. Many electrode-related issues, such as changes in impedance, partial detachment, or motion of the cap or sensor net, alter the distribution of the signal at different times. By examining variance within fixed windows, the Module detects when a channel transitions between different signal regimes across the recording.

Because channels naturally differ in amplitude and baseline scaling, raw windowed variances cannot be compared directly. The data are therefore first transformed using `zScoreRobust()`,

$$Z = \texttt{zScoreRobust}(\mathbf{X}).$$

This transformation normalizes each channel to its own robust scale and suppresses the influence of isolated spikes. This ensures that variance changes over time reflect meaningful shifts within each channel.

The robust-transformed data are segmented using `makeWindowsFromX()`, producing windows $Z^{(w)}$. For each channel c and window w, the windowed variance is

$$v_{c,w} = \text{Var}\big(Z_c^{(w)}\big).$$

Since window-level variance can fluctuate due to small, non-meaningful jitter, the series $\{v_{c,w}\}$ is smoothed across windows using the same moving-median operation described in Rationale for smoothing the dissimilarity measure:

$$\tilde{v}_{c,w} = \mathrm{Smooth}(v_{c,w}).$$

The temporal variability score for channel c is the range of the smoothed sequence,

$$r_c = \max_w \tilde{v}_{c,w} - \min_w \tilde{v}_{c,w}.$$

Large values of r_c indicate strong fluctuations in channel variance over time, consistent with intermittent contact issues or unstable electrodes.

Suspicious Score. Both variability features: overall variability v_c and temporal variability r_c are converted into suspiciousness scores using the same two-step procedure: a log-transform followed by z-scoring across channels.

$$z_c^{(\mathrm{var})} = zScore\big(\log v_c\big), \qquad z_c^{(\mathrm{var2})} = zScore\big(\log r_c\big).$$

- **Overall Variability.** High variance indicates noisy or unstable electrodes, while very low variance may reflect flat or poorly conductive channels. Both situations are undesirable. The Module therefore keeps both tails of the distribution, but uses a looser cutoff on the lower side (-2.5) because low-variance channels can occur naturally in quieter scalp regions or near the reference; they should only be flagged when they are clear outliers relative to the group:

$$\text{if} \quad \big(z_c^{(\mathrm{var})} \le -2.5\big) \text{ or } \big(z_c^{(\mathrm{var})} \ge 2\big) \quad \Longrightarrow \quad \mathrm{isSuspicious}(c) = 1$$

- **Temporal Variability.** The temporal-variability measure r_c is computed on robustly z-scored data, so it already removes baseline amplitude differences between channels. For this reason, the Module uses a symmetric tail cutoff to identify channels with unstable or time-varying behavior:

$$\text{if} \quad |z_c^{(\mathrm{var2})}| \ge 2 \quad \Longrightarrow \quad \mathrm{isSuspicious}(c) = 1$$

3.2.4 Combining Feature Scores into Preliminary Channel Flags

Each general check contributes a suspicious flag:

- Neighbor dissimilarity (Section 3.2.1)
- Maximum-amplitude outliers (Section 3.2.2)

- Overall variability outliers (Section 3.2.3)
- Temporal variability outliers (Section 3.2.3)

In addition, the maximum-amplitude feature includes a *direct bad rule* that immediately marks a channel as bad if its raw amplitude exceeds `chMax`.

Preliminary score. The feature outcomes are combined into a single integer score

$$s_c \in \{0, 1, 2\},$$

defined as:

$$s_c = \begin{cases} 2, & \text{if the direct-amplitude rule marks channel } c \text{ as bad,} \\ 1, & \text{otherwise, if at least one feature marks } c \text{ as suspicious,} \\ 0, & \text{if no features mark } c \text{ as suspicious.} \end{cases}$$

Thus, the direct-amplitude rule is the **only** mechanism that forces a channel into the “bad” category at this stage. All other features contribute only to the “suspicious” category.

Notes. In later stages of the pipeline (clustering and eye-channel identification), designated EOG channels are prevented from being automatically labeled as bad. That constraint is applied after spike-similarity clustering and is not part of the general-check scoring defined here.

3.3 Similar "High-Amplitude Transients" Clustering

Rationale. The second stage of `markSusChs()` groups channels based on the similarity of their high-amplitude transient events. In standard EEG workflows, identifying EEG channels with similar "high-amplitude transients" is highly labor-intensive: users must visually search through dozens of channels, find which ones share similar transient patterns, and decide whether the channels are truly bad or simply reflect a global artifact. This manual process is slow, subjective, and error-prone. By automatically clustering channels according to the "high-amplitude transients" patterns, the Module replaces this manual sorting with a reproducible, data-driven procedure. Not all channels with high-amplitude transients should be removed. Global or widespread prescence of these artifacts are often best handled later (e.g., via ICA), whereas channels with *unique* artifact patterns are more likely to reflect true electrode issues. Clustering provides an efficient way to separate these cases and refine suspicious-channel tagging.

Step 1: Robust z-scoring and high-amplitude transient detection. The procedure begins by robustly standardizing each channel using `zScoreRobust()`. This step is essential because raw high-amplitude transients can heavily distort classical variance-based scaling: a few extreme transients inflate the sample standard deviation, masking the very events we aim to detect. Robust z-scoring instead estimates location and scale from statistics that are insensitive to outliers, producing a standardized signal in which high-amplitude transients are expressed relative to the channel's typical behavior rather than dominating its variability.

$$Z = \texttt{zScoreRobust}(xIn).$$

High-amplitude transients are then identified by thresholding the absolute robust z-scores at a high value (default $z_{\text{transient}} = 14$):

$$B_{c,t} = \begin{cases} 1, & |Z_{c,t}| \geq z_{\text{transient}}, \\ 0, & \text{otherwise.} \end{cases}$$

Time points where no channel exhibits a high-amplitude transient are discarded so that clustering focuses only on moments of high-amplitude events.

Step 2: Max-pooling across short windows. To reduce redundancy and capture only the *presence* of high-amplitude transients (not their precise timing), the binary matrix B is max-pooled in short windows of length *(~200ms)*:

$$\text{samples per block} = \texttt{round}(\texttt{win\_sec} \cdot fs).$$

For each channel c and block k:

$$b_{c,k} = \max_{t \in \text{block } k} B_{c,t}.$$

This produces a binary channel-by-block matrix $b_{c,k} \in \{0,1\}^{C \times K}$, where $b_{c,k}$ indicates whether channel c exhibited any high-amplitude transient in block k.

Step 3: Jaccard-like distance between channels. Clustering directly in the time domain would require operating in a very high-dimensional space (one dimension per time point), which is not practical for distance-based clustering. Instead, the pooled binary vectors b_c summarize each channel by the *pattern of high-amplitude transient occurrences across blocks*. Each element of b_c indicates whether channel c exhibits at least one extreme transient within a given block.

Pairwise Jaccard-like distances "D" are computed with `jaccardDistanceChannels()`, which compares the *overlap of transient-active blocks* relative to the total number of blocks in which either channel exhibits activity.

Two channels are considered similar if their high-amplitude transients tend to occur in the same blocks, mirroring how a user would visually judge “similar high-amplitude transients” when scrolling through the traces.

For each channel c, let

$$N_c = \sum_k b_{c,k}$$

denote the number of blocks containing high-amplitude transients. For a pair of channels $c1$ and $c2$, the dissimilarity coefficient is given by:

$$D_{c1c2} = 1 - \frac{\sum_k \{b_{c1,k} = 1 \cap b_{c2,k} = 1\}}{\max(N_{c1}, N_{c2})}$$

Thus, D_{c1c2} is small when two channels share most of their high-amplitude transients blocks (their high-amplitude transients occur in the same places) and large when their high-amplitude transient patterns rarely overlap. The special case $\max(N_c1, N_c2) = 0$ occurs when neither channel has detected high-amplitude transients; in this situation, we set $D_{cd} = 0$ so that high-amplitude transient free channels are treated as maximally similar and can cluster together.

The resulting matrix $D \in [0,1]^{C \times C}$ is symmetric and encodes the pairwise high-amplitude transient pattern distances that are passed to `clusterByThreshold()` for threshold-based clustering.

For reference, the classical Jaccard similarity is [7]:

$$J_{c1c2}^{(\text{Jaccard})} = \frac{\sum_k \{b_{c1,k} \cap b_{c2,k}\}}{\sum_k \{b_{c1,k} \cup b_{c2,k}\}}.$$

Step 4: Clustering via distance threshold. Once the pairwise high-amplitude transient pattern distances D are computed, the Module groups channels using a simple but robust rule: channels belong to the same cluster if their high-amplitude transient patterns are *sufficiently similar*. This is implemented by thresholding the distance matrix and forming an adjacency graph:

$$A_{c1c2} = \begin{cases} D_{c1c2} \le \varepsilon & cluster\ together, \\ D_{c1c2} > \varepsilon & don't\ cluster \end{cases} \qquad c1 \neq c2,$$

where $\varepsilon \sim 0.8$ — controls the similarity tolerance.

This graph-based formulation mirrors how users would manually group channels: two channels are considered “similar” if their high-amplitude transients appear in the same blocks. Importantly, similarity is treated as *transitive*: if channel $c1$ is similar to $c2$, and $c2$ is similar to $c3$, then all three should belong to the same group even if $c1$ and $c3$ are not directly

similar (Figure 6). The Module therefore defines clusters as the *connected components* of the adjacency graph — that is, groups of channels linked by paths of pairwise similarities.

`clusterByThreshold()` returns the following:

- a cluster index for each channel, and
- a summary of cluster sizes.

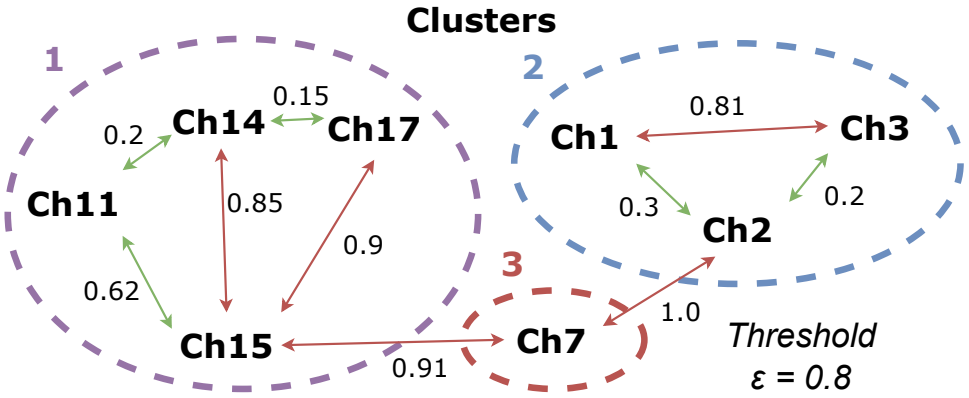


Figure 6: **Clustering via distance threshold.** Edges connect channel pairs whose high-amplitude transient pattern distance is below the threshold ε. Clusters correspond to the connected components of this graph.

3.4 Refining Suspicious and Bad Channels

After computing feature-based suspicious scores (Section 3.2) and forming high-amplitude transient pattern clusters (Section 3.3), `markSusChs()` refines the preliminary per-channel scores $s_c \in \{0,1,2\}$ (*good*, *suspicious*, *bad*). This stage (i) upgrades channels in small high-amplitude transient clusters to suspicious, (ii) protects eye channels from being forced into the bad classification, and (iii) produces the final suspicious and bad-channel lists.

Small high-amplitude transient clusters. Let $\mathscr{C}_k$ denote the set of channels in cluster k returned by `clusterByThreshold()`, and let $n_k = |\mathscr{C}_k|$ be the cluster size. Very small clusters (e.g., singletons, pairs) typically reflect *channel-specific* problems: if only one or two electrodes show a particular high-amplitude transient pattern, it is more likely to be an electrode issue than a global artifact. The toolbox therefore promotes all channels in small clusters to at least suspicious:

$$1 \le n_k < \texttt{minClusterSize} \quad \Longrightarrow \quad s_c = \max(s_c, 1) \quad for \quad c \in \mathscr{C}_k.$$

This rule automatically flags non-global high-amplitude transient patterns that would otherwise require time-consuming manual inspection.

Eye-channel protection. Before applying final bad/suspicious labels, the Module identifies which high-amplitude transient cluster corresponds to the eyes. Let $\mathscr{E}$ denote the set of known eye-channel indices supplied (EOG channels). Using the cluster assignments returned by `clusterByThreshold()`, the Module identifies the cluster containing the most channels in $\mathscr{E}$. This cluster is designated as the *eye cluster*.

Once the eye cluster is identified, all channels belonging to it are protected from automatic "bad" labeling. Eye channels naturally contain large, stereotyped transients (blinks, saccades), which should not cause them to be removed at this bad-channel identification stage. If any eye-channel was marked as bad by the amplitude rule (Section 3.2.2), it is demoted to suspicious:

$$c \in \mathscr{C}_{\text{eye}} \text{ and } s_c = 2 \quad \Longrightarrow \quad s_c \to 1.$$

This prevents eye channels from being classified as bad, while still allowing them to be flagged as suspicious for user review.

add a comment about it is possible that not all channels in E are included in the Eye cluster, (the code thakes the max, but that may lead to a bug if there are two clusters with the same ammoundt of eye channels

3.5 Interpreting General-Check Flags

The feature plots produced by `plotBefore()` summarize the *general-check* flags computed by `markSusChs.m` (Figure 7). These flags are designed to guide the user toward channels that merit closer inspection. Any channel that exceeds a general-check threshold is automatically included in the manual UI review; however, these flags are *not* intended to enforce automatic rejection. Final channel status (*good, suspicious, or bad*) remains a user decision.

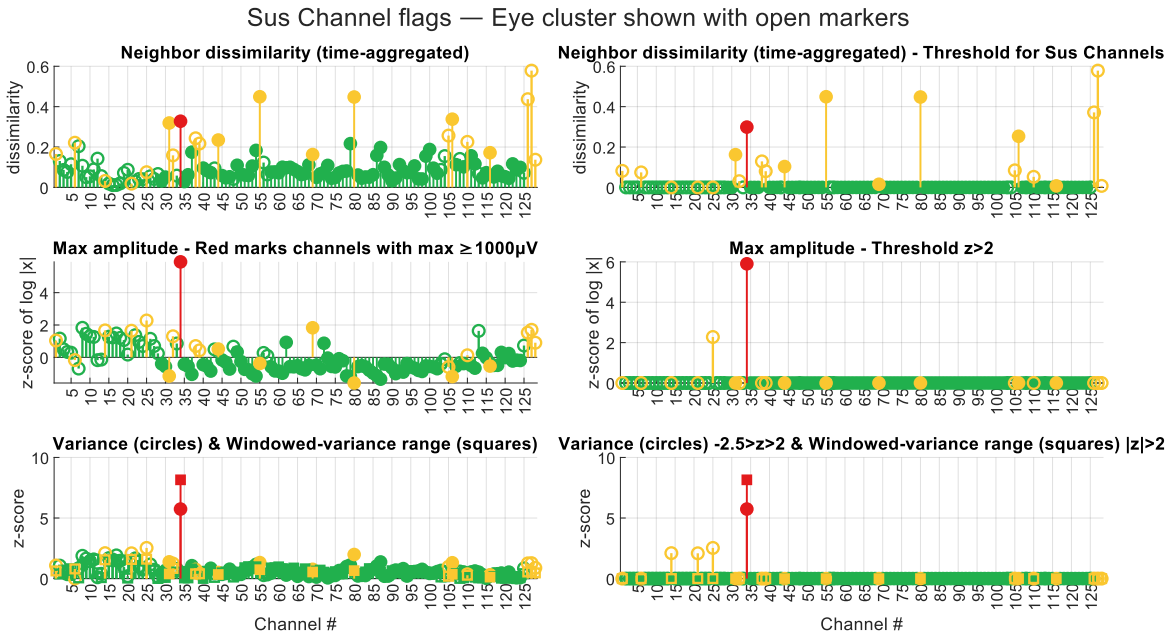


Figure 7: **General checks and thresholds.** This second figure shows: *Left*: Channels' suspicious scores as described in subsection 3.2. *Right*: The threshold-tagged channels that will be shown during UI manual review.

How to use the flag plots. Interpret flag magnitudes *relatively* (i.e., in the context of the full montage), and use them to prioritize attention. A channel with a clearly elevated flag compared to the rest should be reviewed even if the raw trace appears visually plausible.

Neighbor dissimilarity (low agreement with neighbors)

The top row of plots in Figure 7 display neighbor dissimilarity metrics. Neighbor dissimilarity reflects how weakly a channel agrees with its spatial neighbors within short time windows. In general, nearby scalp electrodes are expected to share broad structure because scalp EEG reflects spatially smooth mixtures of underlying sources, with neighboring sensors sampling similar activity with different weights. Consequently, unusually *high* dissimilarity (equivalently, unusually *low* neighbor correlation) suggests that the channel may be dominated by local artifacts (e.g., poor contact, intermittent motion) or is failing to capture the same activity observed in its neighborhood.

- **Interpretation is relative:** expected neighbor dissimilarity depends on montage geometry, electrode spacing, and recording conditions. Consequently, channels are best evaluated by their relative dissimilarity compared to the rest of the montage, with particular attention to clear outliers.

- **Practical rule of thumb:** channels with dissimilarity ≥ 0.4 merit inspection, and channels with dissimilarity ≥ 0.6 merit *careful* inspection. These values are heuristic and may shift with montage geometry and recording conditions.
- **Important:** a channel can appear "clean" by eye yet still have elevated dissimilarity. In such cases, the channel may contain subtle instability or local contamination that is not easily recognized from a single trace but can still degrade spatial analyses.

Note (neighbor dissimilarity). For this feature, values shown in the left panel represent the average dissimilarity across all time windows. In the right panel, windows with dissimilarity below threshold are suppressed prior to aggregation, such that the resulting value is proportional to the area under the dissimilarity curve above threshold over time (see Figure 5).

Maximum absolute amplitude (extreme excursions)

The maximum-absolute-amplitude feature, shown in the middle row of plots in Figure 7, flags channels with extreme transient excursions. These events are particularly important to identify because very large artifacts can dominate covariance structure and, we have found, reduce the effectiveness of spatial decomposition methods (e.g., ICA) used for artifact correction.

- Channels exceeding the configured hard threshold (`chMax`, typically 1000 μV) are tagged as *bad* by default to prevent extreme values from contaminating downstream analyses.
- **Eye-related channels are treated separately:** large deflections in EOG channels are often expected and are useful for ICA-based removal of ocular activity. Therefore, eye-related channels are not automatically rejected solely due to high maximum amplitude.
- If an eye channel exhibits extreme activity that is clearly inconsistent with ocular physiology (e.g., unstable contact, saturation-like jumps, persistent non-ocular noise), it may still warrant being marked as *bad* during manual review.

Variance flags (abnormal variance and instability)

Variance plots are presented in the bottom row of plots of Figure 7. These plots summarize both (i) a global variance measure and (ii) a time-resolved variance measure computed over segments. These features are expressed as z-scores, so interpretation is inherently relative: channels with large-magnitude z-scores exhibit variability that is atypical compared to the rest of the montage.

It is important to note that eye-related channels (EOG) naturally exhibit higher baseline variability due to frequent large-amplitude ocular deflections. Elevated variability in these chan-

nels is therefore often expected and should be interpreted in the context of channel type rather than treated as evidence of poor signal quality.

- **High variability (positive z-scores):** channels with elevated variance relative to the montage often reflect broadband noise, intermittent bursts, or unstable contact. These channels should be prioritized for manual review, especially when the elevation is persistent across time segments. Channels with variability z-scores ≥ 2 are typically considered suspicious and should be reviewed.
- **Low variability (negative z-scores):** unusually low variance can occur for benign reasons (e.g., proximity to a reference electrode or reduced sensitivity), but it can also indicate a failing or "flat" channel. For this reason, low-variance flags use a higher magnitude threshold than high-variance flags. For unusually low variability, a $z \leq -2.5$ threshold is used to reduce false positives while still capturing channels that may be effectively inactive or degraded.

Key takeaway. General-check flags are a prioritization tool. A strong flag does not guarantee that a channel must be rejected, and a weak flag does not guarantee that a channel is good. Use the flag plots to focus attention in the UI review, where the final decision is made.

3.6 Manual Review and Final Channel Selection

The last stage of `markSusChs()` provides a fast manual check of the automatically flagged channels using `reviewBadChsUI()`. The purpose is not full visual cleaning, but a brief verification step to ensure that no electrode is incorrectly removed or overlooked.

Rationale. Automated detection (neighbor dissimilarity, amplitude checks, and variability features) accurately flags most problematic channels, and high-amplitude transient-based clustering groups channels with similar transient patterns. The UI is designed so that this review typically takes under one minute per data record, avoiding the extensive manual labor that traditional bad-channel selection requires.

Interface contents. For each suspicious/bad channel, the interface displays the following:

- the EEG trace,
- the cluster assignment,
- the automated label (*good*, *suspicious*, *bad*).

Example screenshots of the interface are shown in Figures 8–10. Channels in the eye cluster are annotated so the user knows they were protected from auto-bad labeling.

User interaction. The user may leave the automated label unchanged, or override it by clicking on a channel's trace to toggle between the three label options (*good*, *suspicious*, *bad*). Choices are stored in a decision vector:

$$\texttt{userChoice}(c) \in \{0,1,2\},$$

where 0 ($green$) = good, 1 ($yellow$) = suspicious, 2 (red) = bad. If the user makes no selection for a channel, the automated decision is retained.

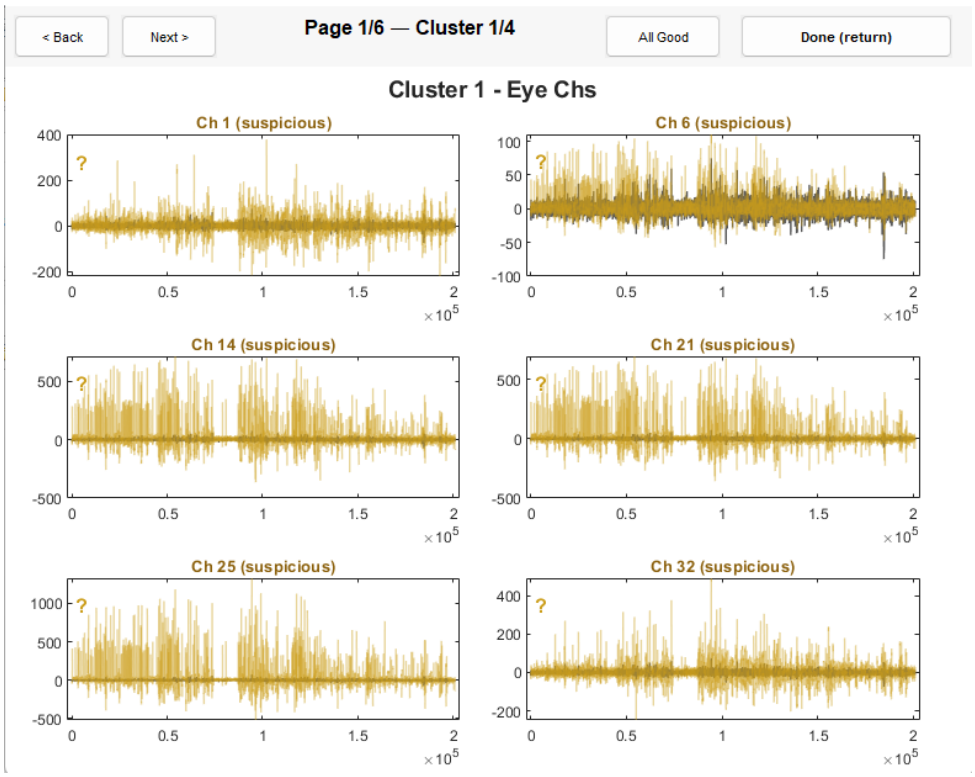


Figure 8: **Manual review — Example 1A.** First panel shown during manual review for a 128-channel EGI recording. Several highlighted channels belong to the eye-movement cluster; these are preserved because ocular activity will be removed later with ICA. Channel 25, however, exceeds 1000 μV, which would degrade ICA whitening and component decomposition (see Section 3.2.2). In the interface, clicking on a channel's trace immediately toggles its label among *good*, *suspicious*, and *bad*, allowing rapid correction of the automated classification.

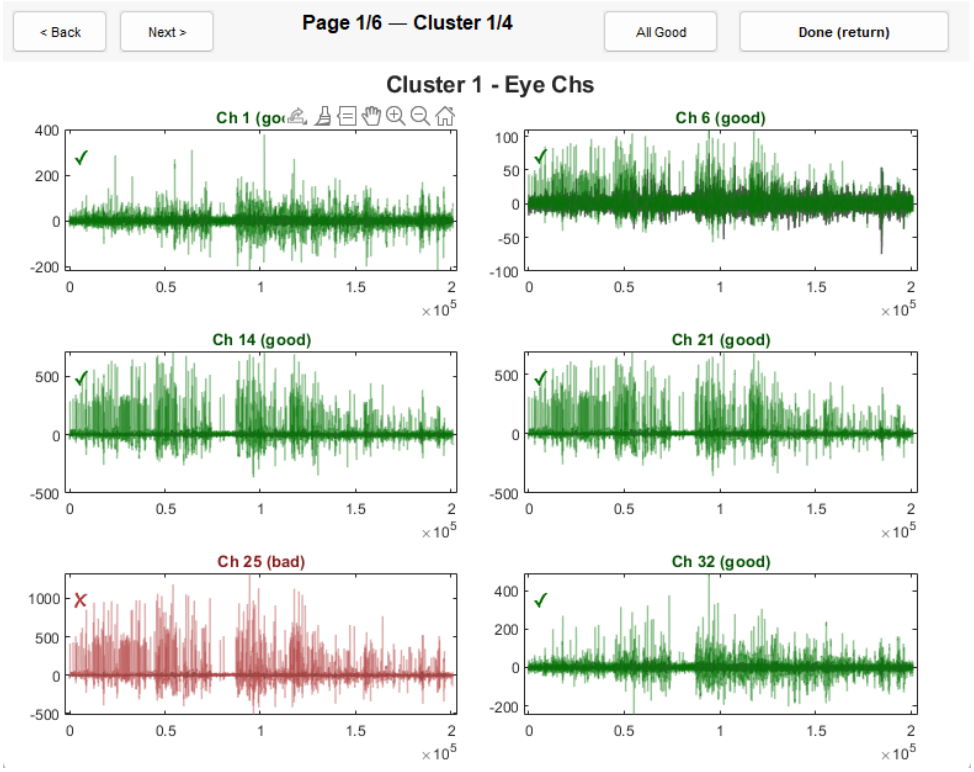


Figure 9: **Manual review — Example 1B (after user overrides).** The plots represent the same data as in Figure 8, but after the user has interacted with the traces. Each channel is now color-coded according to the user's selection (*green* = good, *yellow* = suspicious, *red* = bad). Clicking on a trace cycles through these categories. Only channels marked *red* are ultimately saved as bad channels; the green/yellow colors are provided only for visual guidance during the review process (i.e., there is no need to convert to green any yellow channels a user deems to be good).

Figure 10: **Manual review — Example 2 (isolated high-amplitude transient pattern).** Example of a channel that forms a singleton cluster during the high-amplitude transient similarity analysis. Channel 34 shows large, isolated transients that do not resemble normal EEG morphology and do not co-occur with high-amplitude transients in any other channel. Because its high-amplitude transient pattern is unique, `clusterByThreshold()` assigns it to a one-member cluster. Such "loner" channels are strong candidates for removal.

CHAPTER 4

Example Workflow: Running `markSusChs()`

This section walks through the tutorial script `example.m` in the `BadChannelDetection` Module. This walkthrough highlights important variables and configuration steps. It does not explain every line of code, but instead focuses on the elements that users are expected to adjust when applying the Module to their own datasets.

The script shows how to:

1. specify the data location and load an example EEG file,
2. configure `markSusChs()` parameters and output folders,
3. run the automatic suspicious/bad-channel detection,
4. interact with the manual review UI,
5. optionally perform extra visual inspection and manual overrides,
6. visualize the effect of removing bad channels.

To run the tutorial, open **`example.m`** in MATLAB, ensure that the **`BadChannelDetection`** folder is on your MATLAB path. Example:

```
addpath(genpath('SENSI-EEG-Preproc-bad-ch-main'))
```

- Where, "SENSI-EEG-Preproc-bad-ch-main" represents the folder containing the code base.
- Run the script section-by-section (Steps 1–3 are required; Steps 4–7 are optional).
- If you do not change the folder paths (explained below), make sure you are located in the code base root folder when running the **`example.m`** script.

The example will download the sample files (e.g., `data1.mat`, `data2.mat`, `data3.mat`) to a folder of your choice. Each file will contain, at minimum:

- `xAll_129`: filtered EEG data
- `fs`: sampling rate (Hz)
- `EOG`: indices of EOG/auxiliary channels. Otherwise, define them in the script (use EOG struct in example data as reference).

4.1 Step 1: Setup

Paths, Downloading Files, Filenames, and Basic Options

The first section of `example.m` sets the data location, downloads the sample files, chooses which example file to load, and specifies where to save figures.

```
%% 1. SETUP: PATHS, FILENAMES, AND BASIC OPTIONS

% Folder containing example data#.mat files
dataFolder = 'ExampleData\'; % <-- Optional. Set folder path
getExampleData_badChannel(dataFolder);  % download example.mat files

% Choose one example file:
%   'data1.mat' : Example dataset 1
%   'data2.mat' : Example dataset 2
%   'data3.mat' : Example dataset 3
fileName = 'data1.mat'; % <-- EDIT THIS

% Folder where tutorial figures are saved
INFO.preprocFigDir = 'Figures\'; % <-- Optional. Set folder path

% Toggle saving of figures (1 = save, 0 = only show on screen)
saveFigs = 1; % EDIT (optional)

% Remove reference channel (assumes the 129th channel is reference)
xAll_128 = xAll_129(1:128,:);
```

Key points:

- **`dataFolder`** and **`INFO.preprocFigDir`** may be updated to prevent path errors.
- **`fileName`** selects which dataset is used in the tutorial.
- The script assumes an EGI-style 129-channel recording and removes the 129th (zero-valued reference) channel, producing `xAll_128` with 128 scalp channels.

Outputs

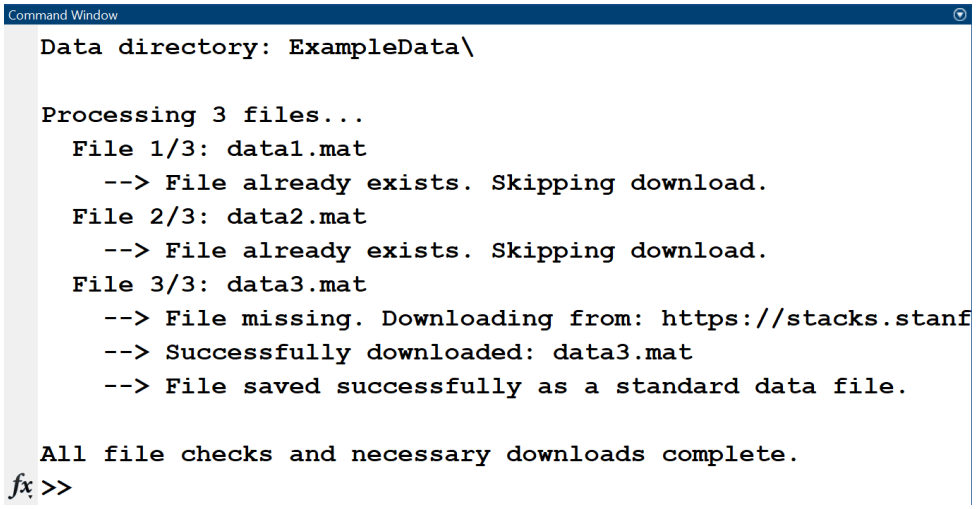


Figure 11: **Command line — downloading files**. Section 1 will ensure the example files are contained in the **"dataFolder"** directory. If they are not present, it will download them to the defined directory.

If there is an error while downloading the data, you may do the following:

1. Comment out `"getExampleData_badChannel(dataFolder);"`
2. Download the sample data[a]
3. Place sample data in the **ExampleData/** folder, or update the **dataFolder** variable with your desired data folder.

[a] https://purl.stanford.edu/dg856vy8753

4.2 Step 2: Configure INFO.badChs Options

The second section defines all parameters used by markSusChs() via the INFO.badChs structure:

```
%% 2. CONFIGURE markSusChs OPTIONS (INFO.badChs)

% Initialize bad-channel list (empty for first run)
INFO.badCh = [];

% WINDOWING PARAMETERS -> used by makeWindowsFromX()
INFO.badChs.winSec = 10;                   % window length (s)
INFO.badChs.hopSec = INFO.badChs.winSec;  % hop size (no overlap)

% CLUSTERING PARAMETERS
INFO.badChs.win_sec = 0.200;  % block size for max-pooled spikes (s)
INFO.badChs.eps     = 0.8;    % distance threshold for clustering

% UI PLOT SETTINGS
INFO.badChs.minClusterUI = 7;  % show clusters up to this size in UI
INFO.badChs.ref = 36; % Ref Ch for comparison (not EEG reference).

% THRESHOLDS
INFO.badChs.chMax = 1000;          % cutoff (µV) for bad channel
INFO.badChs.neighborDissThresh = 0.3; % Suspicious Threshold
                                        (correlation <= 0.7)
```

These values are reasonable starting points for 128-channel EGI data. Users can tune them based on their own montage and noise characteristics.

Guidelines for Choosing Parameter Values. The parameters in this section control how markSusChs() computes features, detects high-amplitude transients, forms clusters, and displays channels in the UI. Below are practical guidelines for adjusting them to your own dataset:

- **Windowing parameters (winSec, hopSec).** These determine the temporal resolution of neighbor-dissimilarity and time-varying variance features. Shorter windows increase sensitivity to brief artifacts; longer windows provide more stable estimates. Default = 10s.
- **Clustering parameters (win_sec, eps).**

  - **win_sec** controls the duration of blocks used for high-amplitude transient pooling. Values in the range 100–300 ms work well for detecting short transients. Default = 0.200s.
  - **eps** is the similarity threshold for grouping channels. Larger values (e.g., 0.8–0.9) produce fewer, broader clusters; smaller values (e.g., 0.5–0.7) enforce stricter matching between channels. Default = 0.8.

- **UI display options (`minClusterUI`, `ref`).** These settings affect only the presentation in the manual-review interface. Adjust these for clarity rather than analysis.
  - **minClusterUI**: Clusters with ≤ `minClusterUI` channels are added for manual review in the UI. Smaller clusters are more likely to contain problematic channels and therefore warrant closer inspection. Larger clusters usually reflect global or widespread patterns and are less likely to contain isolated bad electrodes. This value should be adjusted based on montage size; for EGI 128-channel systems, a default of 7 works well.
  - **ref** is the comparison channel used for visual overlays (visual aid only)

- **Suspiciousness thresholds.**
  - **chMax** defines the hard cutoff for automatically labeling a channel as bad due to extreme amplitude excursions (e.g., ±1000 µV).
  - **neighborDissThresh** sets the cutoff for marking a channel suspicious based on sustained neighbor dissimilarity (see Figure 5). Lower values make this check more sensitive. Default = 0.3 (0.7 correlation).

4.3 Step 3: Run `markSusChs()` and Inspect Clusters

The next section runs the main bad-channel detection function and prints cluster memberships to the Command Window:

```
%% 3. RUN markSusChs AND INTERACTIVE REVIEW

[susMask, badChList, plotStruct] = markSusChs(xAll_128, fs, ...
    INFO, EOG, saveFigs, INFO.preprocFigDir, thisFnOut);
```

This section performs all functions for bad-channel detection:

- Computes all suspicious scores (neighbor dissimilarity, amplitude, variability),
- Performs high-amplitude transient detection, block binarization, and high-amplitude transient pattern distance computation,
- Applies the clustering threshold `INFO.badChs.eps` to form high-amplitude transient pattern clusters,
- Identifies the eye cluster and protects eye channels from automatic bad labeling,
- Initializes the suspicious and bad masks,
- Launches the interactive review UI.

Outputs and User Interaction

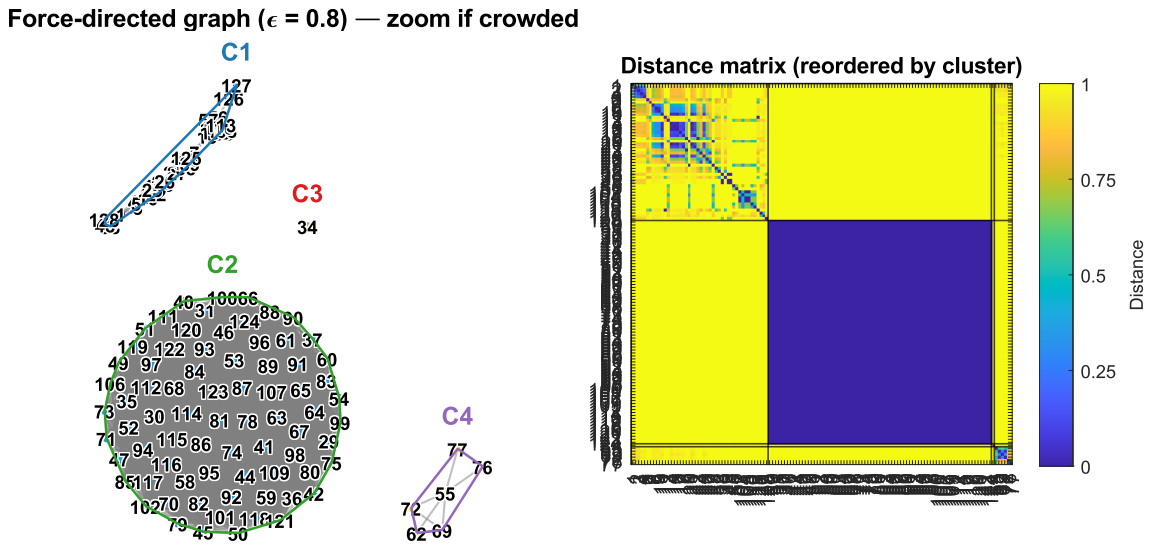


Figure 12: **Channels cluster structure.** The first Module figure provides an overview of the clustering structure and the separation between channel groups. *Left*: A force-directed graph that reveals relationships and structure of the **high-amplitude transients clusters**. *Right*: A distance matrix reorganized by clusters.

- These plots show the **high-amplitude transient artifacts relations between channels**.
- Clusters with low number of channels like C3 are strong bad-channel candidates because their high-amplitude transients structure is distinct from other channels.
- Cluster with less than **"minClusterUI"** are automatically tagged as *suspicious* and passed to UI manual review. For example, if `minClusterUI` = 7, the C4 cluster containing 6 channels would be marked as suspicious and included in the UI review, even if the general feature-based checks do not flag those channels.
- Channels in cluster C2 (the large blue square on the right) are unlikely to be bad or suspicious. This cluster appears blue because its channels either exhibit no detected high-amplitude transients or share identical transient occurrences, resulting in perfect similarity (distance = 0). In contrast, cluster C1 (the upper-left square and second-largest cluster) contains channels that are not perfectly related, as indicated by nonzero pairwise distances.

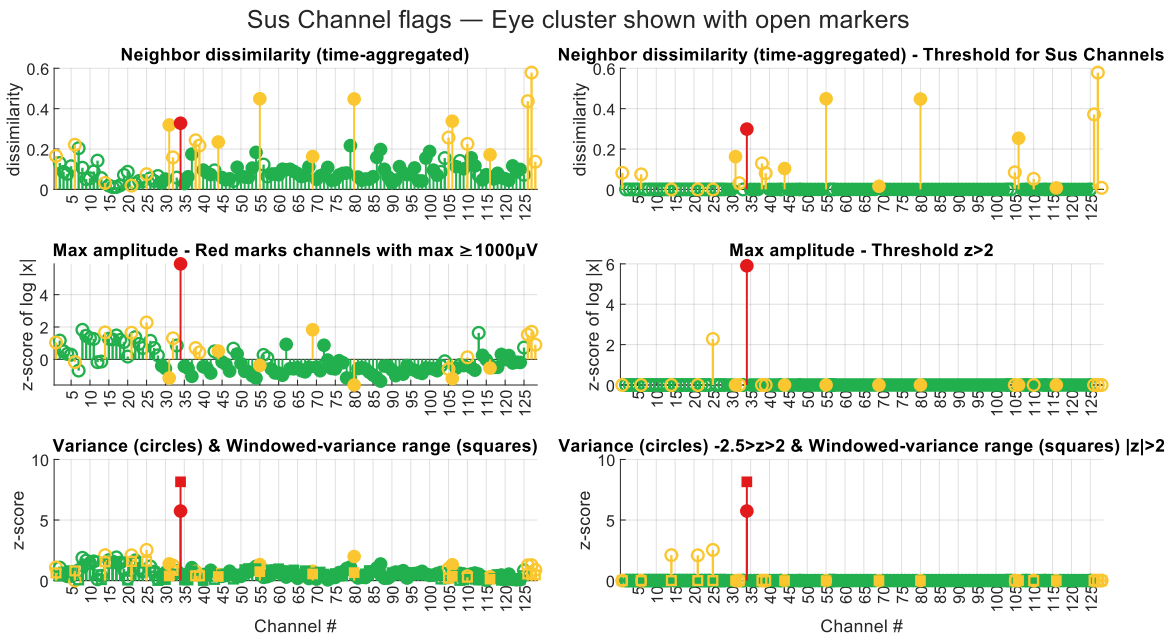


Figure 13: **General checks and thresholds.** This second figure from the Module. *Left*: Channels' suspicious scores as described in subsection 3.2. *Right*: The threshold-tagged channels that will be shown during UI manual review. Channels are color-coded as *good* (green), *suspicious* (yellow), or *bad* (red).

For guidelines on how to evaluate these plots, see subsection 3.5. For plot on the right in Figure 13, channels flagged in red or yellow are automatically included in the manual UI review. Channel 34 is marked as bad and exhibits strong flags across multiple metrics, consistent with poor signal quality. Channels 55 and 80 warrant particular attention due to unusually high neighbor dissimilarity (dissimilarity ≥ 0.4). In this dataset, channels 126 and 127 correspond to cheek electrodes; because their neighboring electrodes are less well defined than those in the central scalp layout, elevated neighbor dissimilarity is expected and does not necessarily indicate poor channel quality.

- Yellow stems show suspicious channels passed to UI review.
- Red stems show direct bad-channel tagging ($\max(V_{\text{channel}}) > 1000\mu V$)
- Eye channels (open stems) are never tagged as bad channels

Eye cluster may not be perfect. Look at clusters in UI review.

The next step is the UI review. At this stage, EEG channels are grouped into clusters based on their high-amplitude transient-pattern similarity. The goal of the interface is to allow the

user to visually inspect *suspicious* channels and decide whether they should be promoted to *bad* and removed or retained for downstream analyses. Channels are color-coded for clarity: yellow = suspicious, red = bad, and green = good.

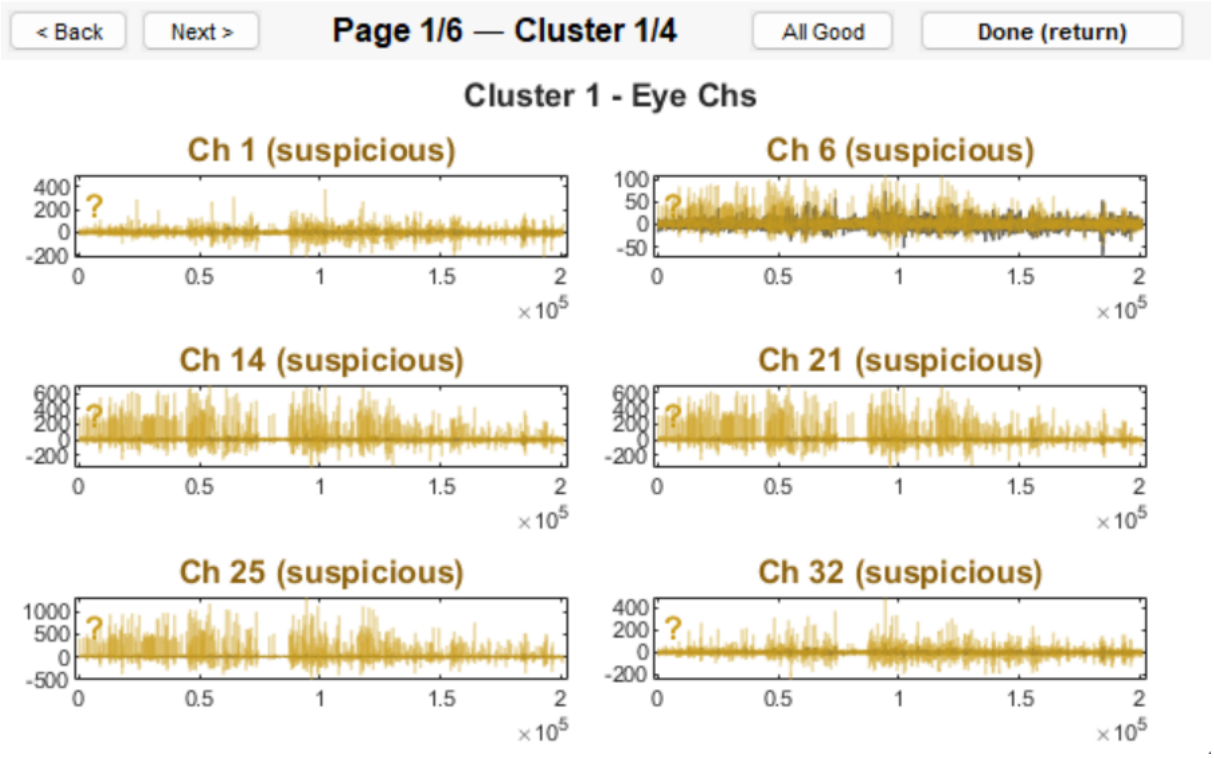
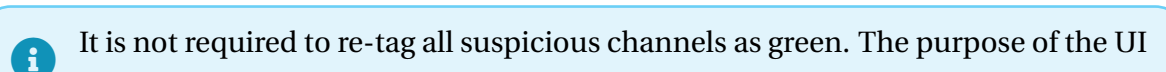


Figure 14: **UI controls — Example 1, page 1**

UI Controls

- **<Back** and **Next>** buttons — navigate between panels. Each panel shows up to 6 plots (can be changed in settings); larger clusters span multiple panels.
- **Page / Cluster indicators** — show your current position in the review.
- **Tag Cycle** on individual panels — click on a channel's plot to cycle its status (Suspicious → Good → Bad).
- **All Good** button — quickly mark all channels in the current panel as good.
- **Done (return)** button — finalize decisions and return `badChList`. This is the only valid way to exit the UI.

⚠ Do **not** close the UI window manually. Doing so bypasses the callback logic and will produce an error. **Always use Done (return) to finalize the review.**



It is not required to re-tag all suspicious channels as green. The purpose of the UI is to identify and confirm **bad** channels. Green tags are mainly for user reference; the plots before and after UI review provide a record of your decisions.

All UI figures generated during manual review are saved to the folder specified by `INFO.preprocFigDir`. The following pages illustrate the remaining UI panels for this example dataset, along with brief notes describing the decisions made during review.

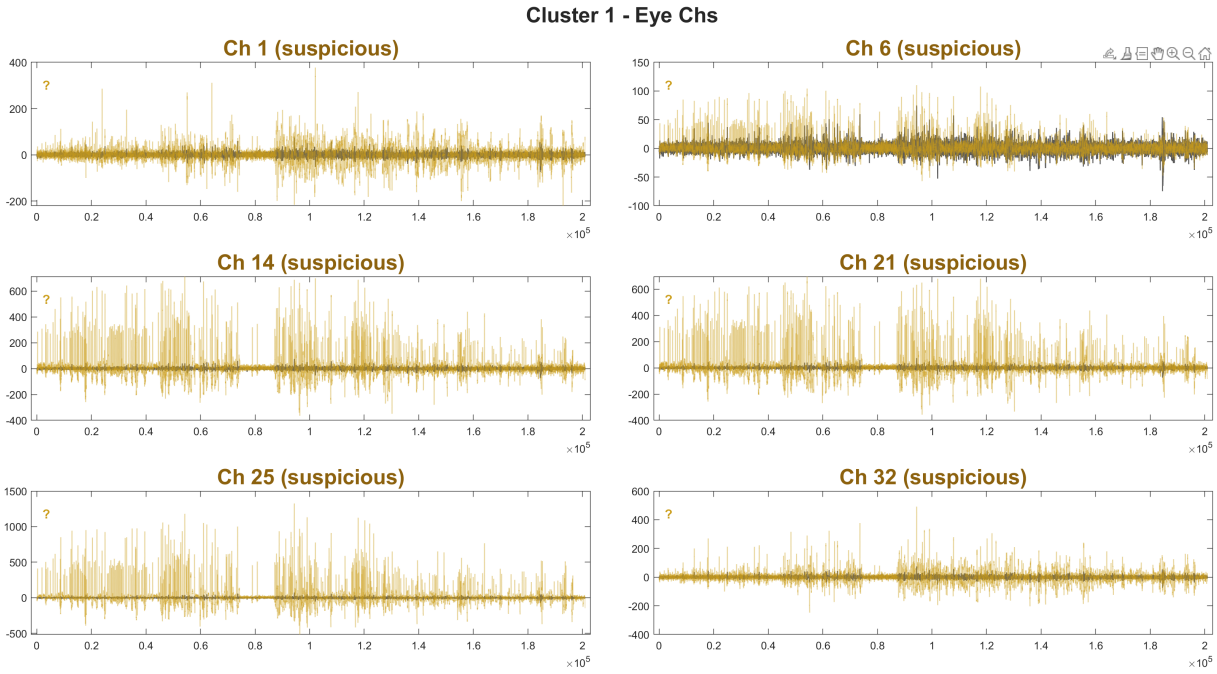


Figure 15: **Example 1, page 1 (review).** Channels in this cluster show strong eye-related activity, which will later be removed using ICA. Channel 25 (bottom left) exceeds the 1000 μV amplitude limit; however, this reflects pronounced eye artifacts that ICA will subsequently remove from all channels. Therefore, we do not mark any of these channels as bad. If an eye-channel displayed a large artifact that was *not* eye-related, then removal of that specific channel would be considered.

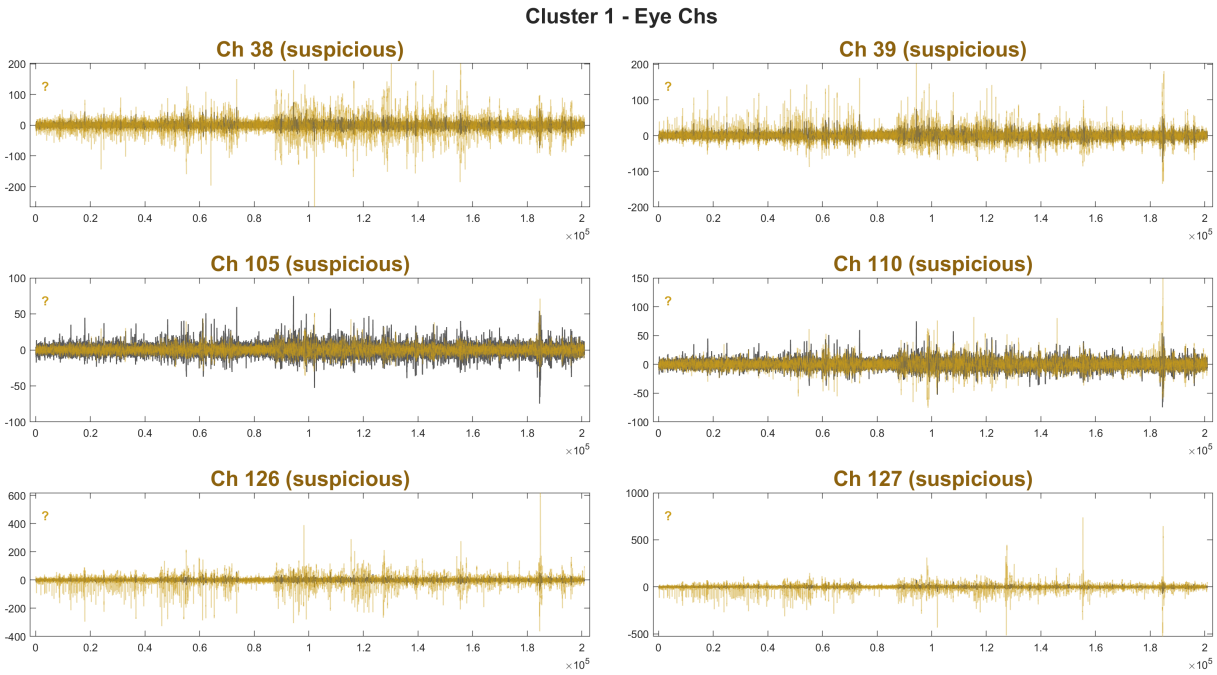


Figure 16: **Example 1, page 2.** No major isolated transients are present. The observed high-amplitude transients are shared across multiple channels and repeat over time, indicating global artifacts that e.g., ICA can remove. All channels were therefore kept.

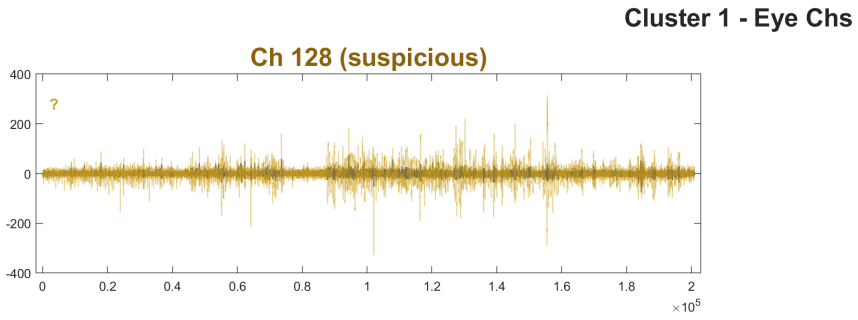


Figure 17: **Example 1, page 3.** No problematic channels detected. The suspicious transient also appeared in earlier channels and is consistent across many electrodes, suggesting a global artifact suitable for removal using ICA.

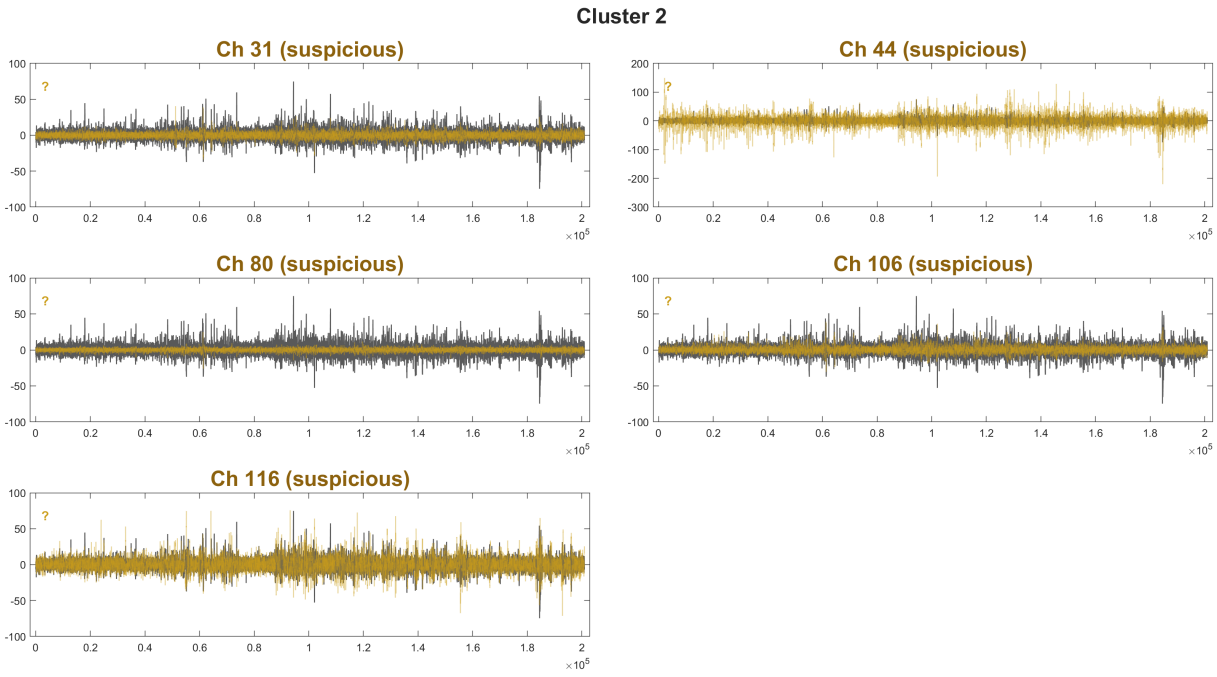


Figure 18: **Example 1, page 4.** Cluster 2. No concerning behavior. The visibly salient late-session artifact seen here is small and also appears in Cluster 1, reinforcing that it is a global transient likely to be handled by ICA.

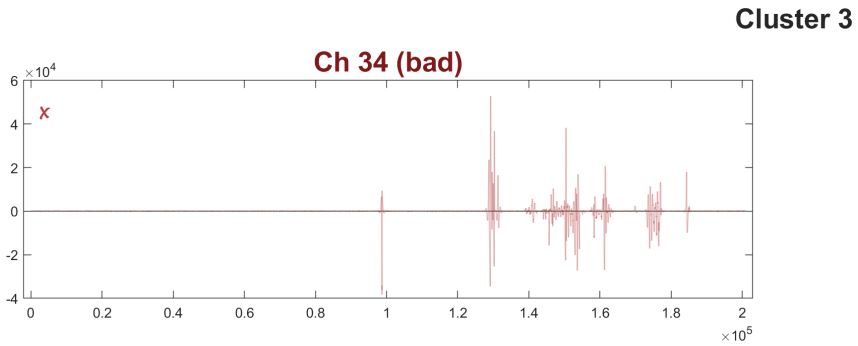


Figure 19: **Example 1, page 5 (singleton cluster).** Cluster 3. This single-channel cluster contains a clearly abnormal signal with high-amplitude, isolated artifacts. Channel 34 was correctly flagged as bad and should be excluded from further analysis.

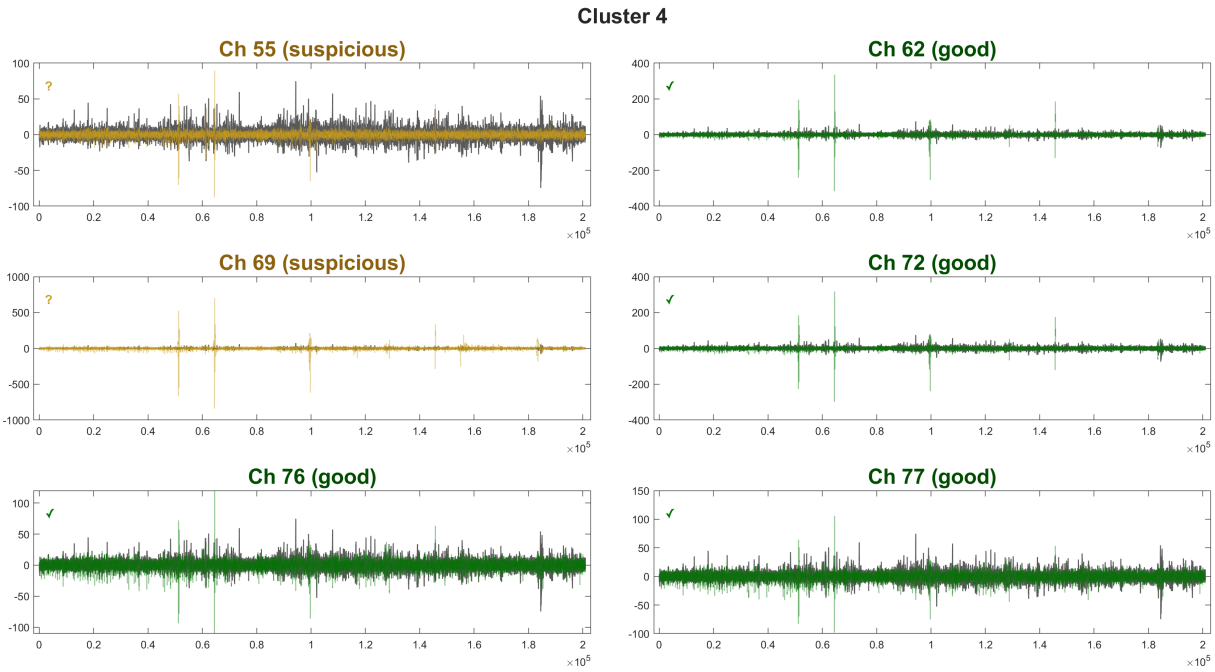


Figure 20: **Example 1, page 6.** Cluster 4. Similar to Page 2, the high-amplitude transients are widespread and repeat consistently across channels, indicating global artifacts rather than electrode-specific issues. Since none exceed the amplitude threshold, all channels were kept.

The final step is to press **Done (return)** in the UI. At this point, `markSusChs()` finalizes the review, stores the selected bad channels in **badChList**, and saves all relevant figures to the output directory.

Step 4: Extra manual inspection (Optional — Not part of `markSusChs.m`)

This step does not alter the bad-channel list automatically; it simply helps inform decisions. Users may add channels to the bad list (via **manual.addBadCh**) or protect specific channels from removal (via **manual.neverRemove**) before creating the final list in Step 5–6.

```
%% 4. OPTIONAL: EXTRA MANUAL INSPECTION (NO CHANGE TO badChList)

% Manually Inspect Channels
manual.inspectCh      = [ ];   % e.g., [12 37 88]; leave [ ] to skip

% Manually add to and/or remove from bad-channel list
manual.addBadCh    = [ ];   % user-added bad channels (unordered)
manual.neverRemove = [ ];   % channels that must be kept
```

Important variables in step 4:

- **manual.inspectCh** – Channels that you may want to inspect.
- **manual.addBadCh** – Channels to force into the bad list in Step 5–6.
- **manual.neverRemove** – Channels that must be kept (they will be removed from the bad list in Step 5–6).

Step 5-6: Override and Before vs. After visualization

Step 5 (Override) constructs a new EEG matrix with the identified bad channels removed. This step applies the final bad-channel decisions from the automated detection and any prior manual inspection.

Step 6 provides a quick sanity check using a two-panel plot (Figure 21): the top panel shows the raw EEG (all channels), while the bottom panel displays the EEG after removing the identified bad channels. This visualization helps verify that bad-channel removal improves overall signal quality without unintentionally discarding large portions of usable data.

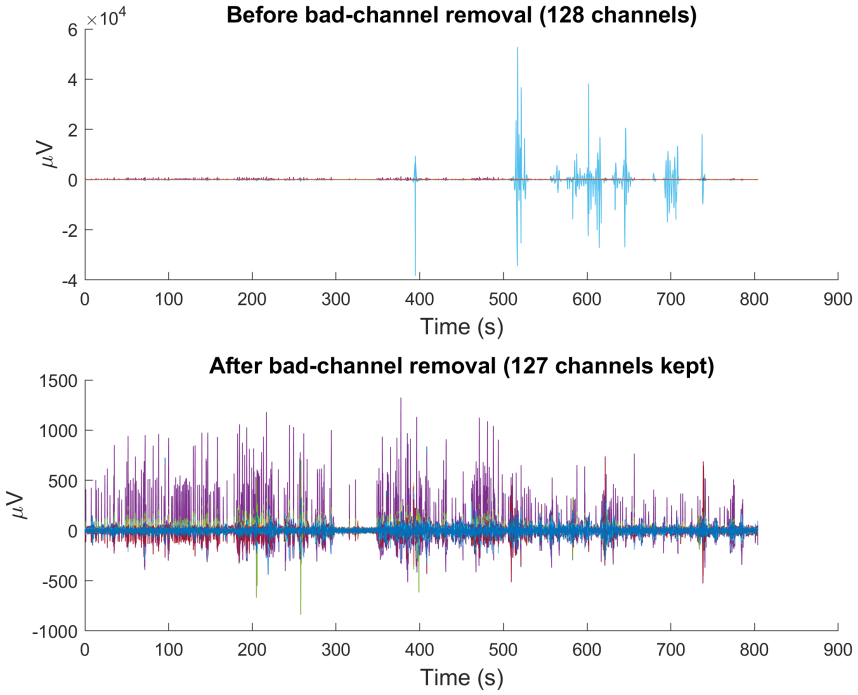


Figure 21: **EEG stacked plot — before vs. after bad-channel removal.** Stacked traces of all 128 channels before (top) and after (bottom) removing bad channels. This overview plot helps assess whether the cleaned dataset shows smoother, more consistent activity and whether visibly corrupted channels were successfully removed. *Top*: the data are dominated by a bad channel, which has increased the range of the y-axis and obscured the activity of the other channels. *Bottom*: after removing the bad channel, the remaining channels exhibit typical EEG activity, including EOG in some channels.

CHAPTER 5

Additional Illustrative Examples

In this section we present two additional runs of `markSusChs()` using `data2.mat` and `data3.mat`. These examples are intended to illustrate how the same bad-channel detection framework behaves with other data records (e.g., cleaner versus noisier sessions), and how UI decisions depend on the underlying artifact patterns.

No new code is introduced here: each example uses the tutorial script `example.m`, changing only the `fileName` variable to `data2.mat` or `data3.mat` and re-running the workflow described in Section 4. We therefore focus on the *figures* and on the *manual decisions* made during UI review.

5.1 Example 2:

Data summary

Data used

- Dataset: `data2.mat`
- Channels: **129** scalp channels (EGI-style)
- Sampling rate: **250 Hz**
- Notes: **moderate noise. No manual interaction required.**

Cluster structure and suspicious scores

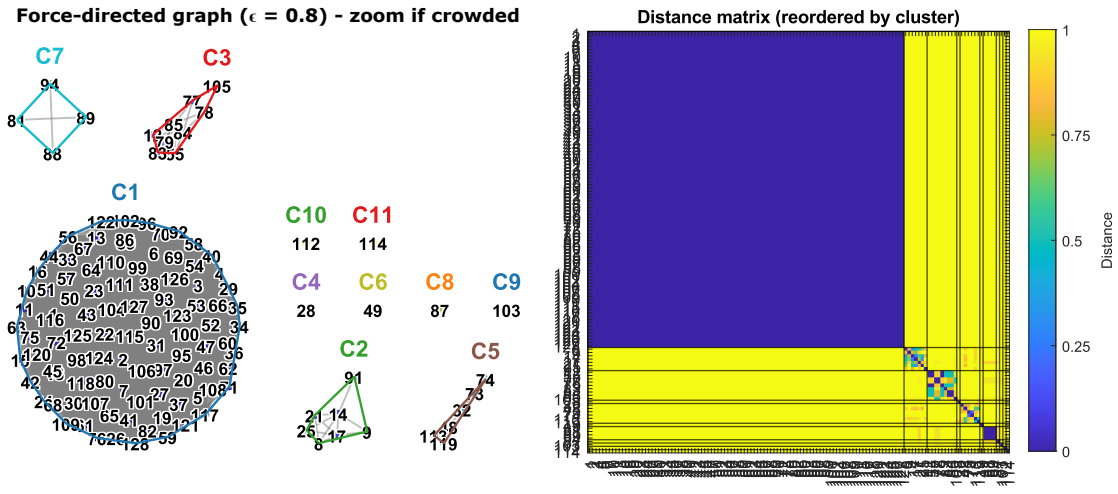


Figure 22: **Example 2 — high-amplitude transient pattern clusters and distance matrix.** *Left:* Force-directed graph showing high-amplitude transient pattern clusters. *Right:* Distance matrix reordered by cluster index.

Cluster overview

- **Cluster C1 (96 channels).** This cluster is represented by the large blue square on right plot of Figure 22 and is the cluster of channels without high transients (technically speaking, it is possible that they all have the same transients). Channels in this cluster are less likely to be bad or suspicious.
- **Clusters C10, C11, C4, C6, C8, C9.** Six singletons to review.
- **Clusters C7, C3, C2, C5.** Four small clusters to review.

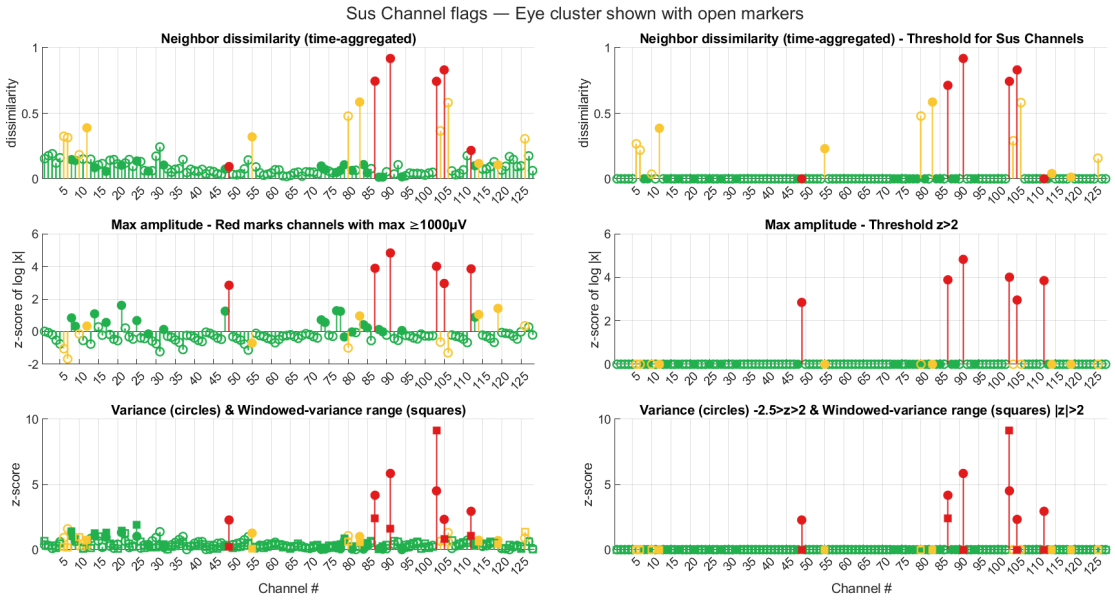


Figure 23: **Example 2 — suspicious scores and thresholds.** *Left*: Feature scores per channel. *Right*: Channels exceeding suspicious / bad thresholds. There are multiple suspicious and bad channels, and this recording needs detailed inspection. Remember to reference these flags when assessing a channel and deciding whether to tag it as bad.

UI Panels and Manual Decisions

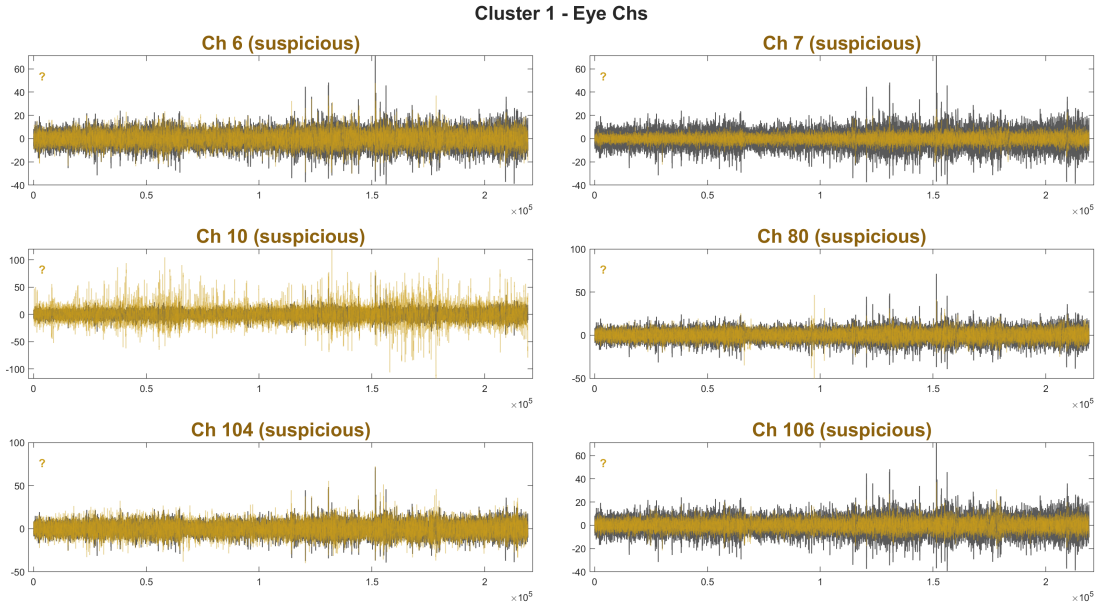


Figure 24: **Page 1, cluster 1 — eye channels.** These channels are fine. Channel 10 is slightly noisier than the rest, but not an atypical waveform for an EOG channel and lacks flags in Figure 23. Channel 106 has suspicious Neighbor dissimilarity (>0.4), but the time waveform looks acceptable.

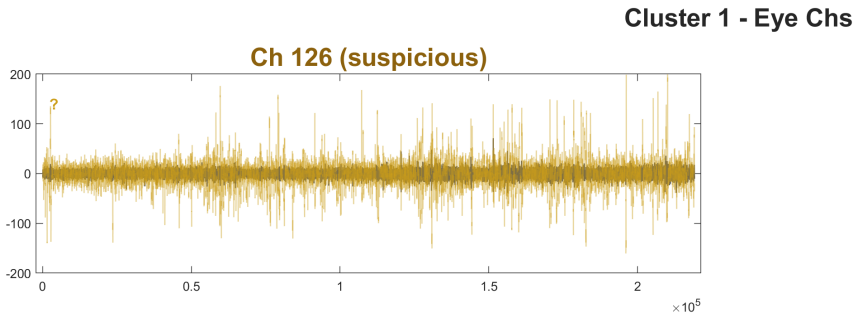


Figure 25: **Page 2, cluster 1 — eye channels.** This channel is fine. It is similar to channel 10 but with higher amplitude. No strong flags.

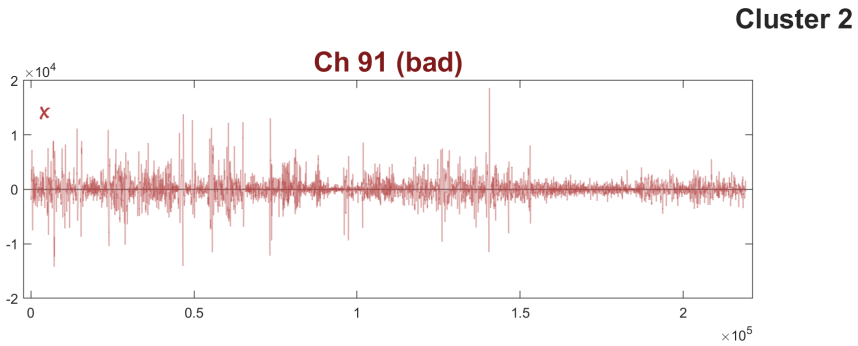


Figure 26: **Page 3, cluster 2 (singleton).** This channel was successfully auto-tagged as bad. The shown data are not characteristic of EEG activity but rather reflect an artifact (perhaps poor connection between the electrode and the scalp.) This channel had the highest neighbor dissimilarity in Figure 23.

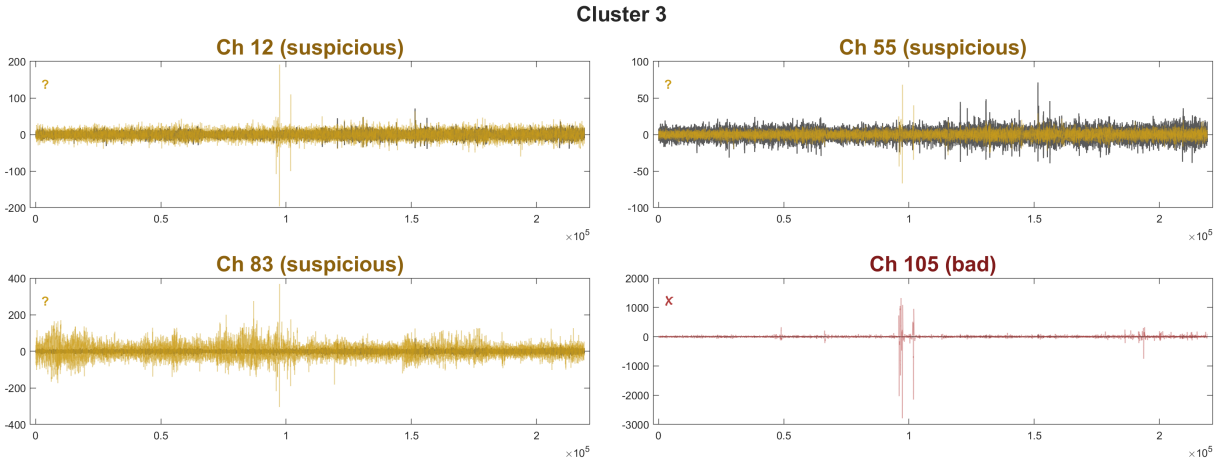


Figure 27: **Page 4, cluster 3.** Channels 12 and 55 are fine. Channel 83 is a bit noisier, with a neighbor dissimilarity score of ≈ 0.71 (high). Due to this, we will remove this channel by clicking on the plot until it toggles to red. Channel 105 has two transients located in the same position as other channels in this cluster, and it is most likely the source of this cluster's high-amplitude transients. However, the magnitude is too large and it would affect ICA performance. Therefore, we will remove channel 105 as originally tagged.

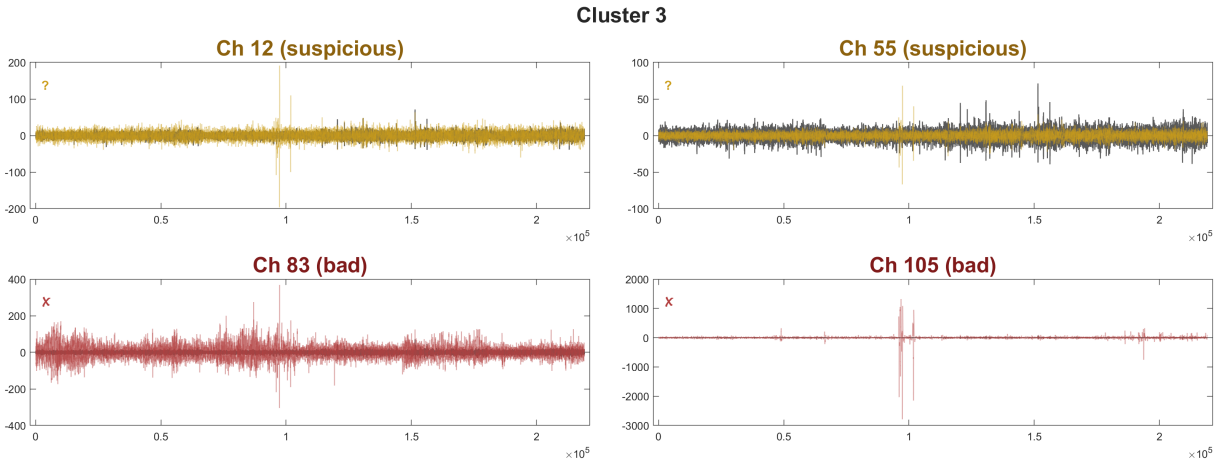


Figure 28: **Page 4, cluster 3 (after review).** This is how page 4 (formerly Figure 27) should look after user review and intervention. Channel 105 was already tagged as bad channel, and we added channel 83 for removal.

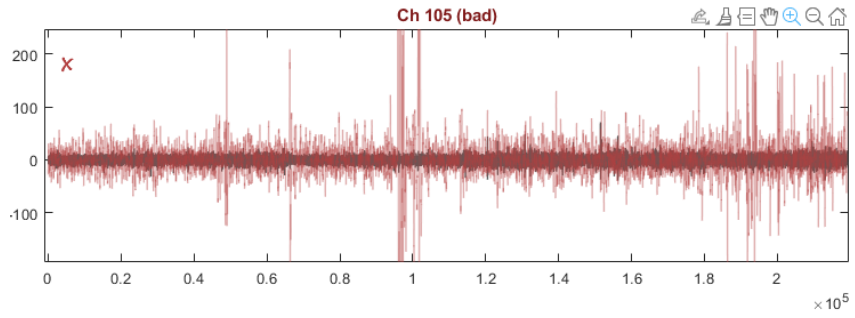


Figure 29: **Channel 105 (for context) — zoomed in.** We already selected this channel for removal and will not change the decision. After zooming in, we can see artifacts of channel 105 are not isolated to a certain time point, but are present continuously. This channel also had a high neighbor dissimilarity score. Therefore, this channel cannot be salvaged by e.g., later interpolating only the large transients.

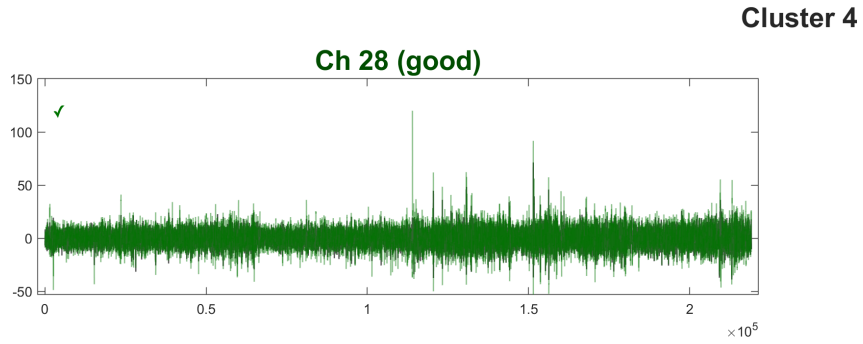


Figure 30: **Page 5, cluster 4 (singleton).** This channel is fine. It has "small" transients that are shown in other channels, with values so low that they will likely not impact ICA. It is likely that these artifacts can be removed by other means without needing to reject the entire channel.

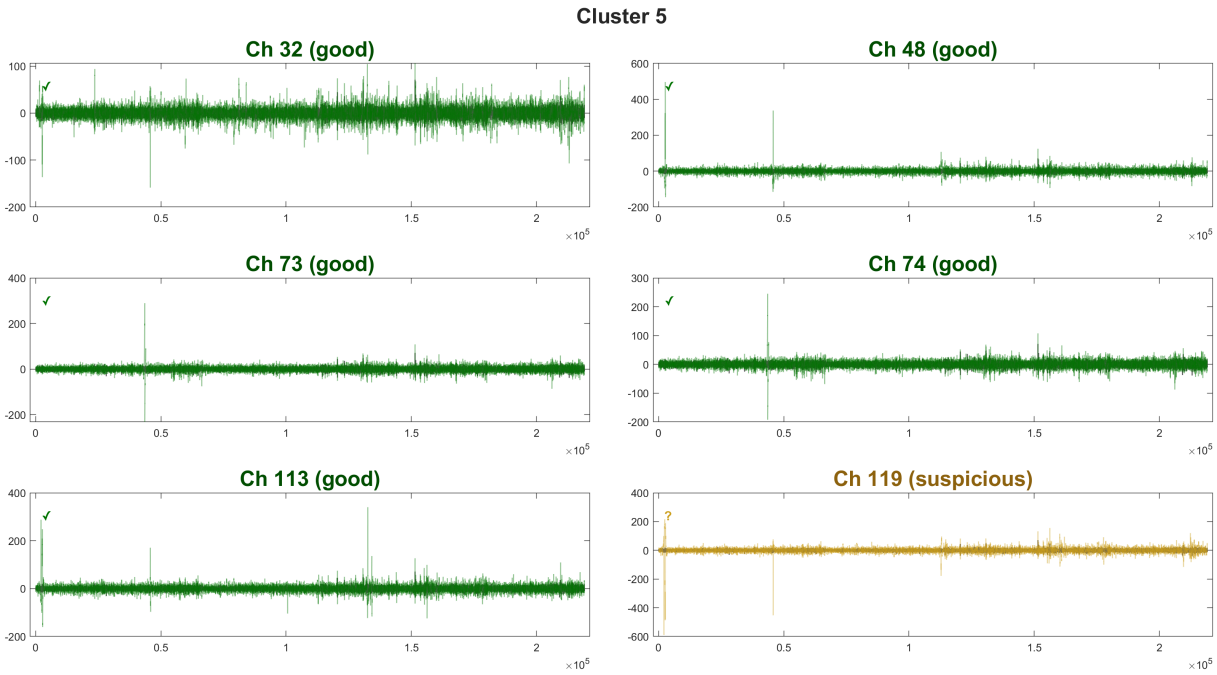


Figure 31: **Page 6, cluster 5.** These channels are fine. These artifacts are likely to be removed after ICA. None of these channels have strong flags, other than 119 being dissimilar with its neighbor in a small time frame.

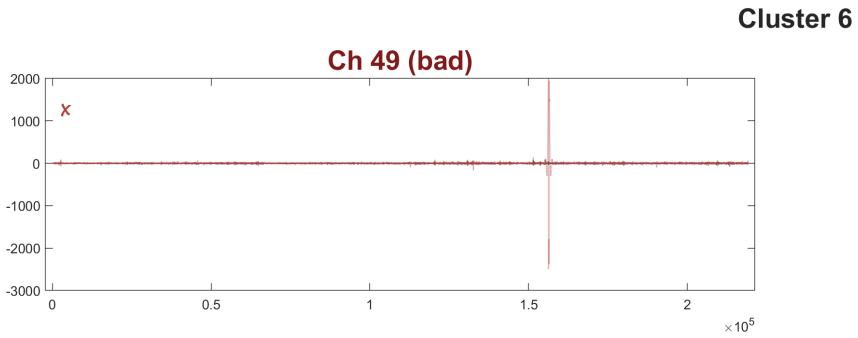


Figure 32: **Page 7, cluster 6 (singleton).** This channel was successfully tagged as bad. The transient around sample 1.56×10^5 is too large and will likely affect ICA. One thing to keep it mind is that the neighbor dissimilarity is low, therefore, the artifact is likely to be present only in a small time frame. We investigate this channel further in the next figure.

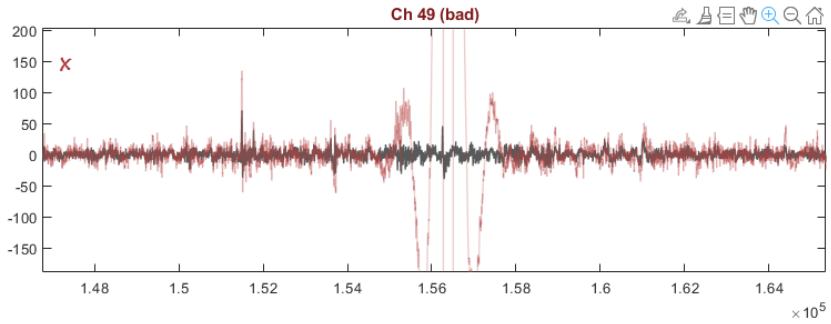


Figure 33: **Channel 49 (for context) — zoomed in.** We already selected channel 49 for removal and will not change the decision. After zooming in, we can see that the artifacts are isolated between samples 1.54×10^5 and 1.58×10^5. Channel 49's EEG activity follows the visual reference channel (gray plot in the background) before and after the artifact. Therefore, this channel could alternatively be salvaged by replacing the large transients with spatially interpolated values from its neighbors.

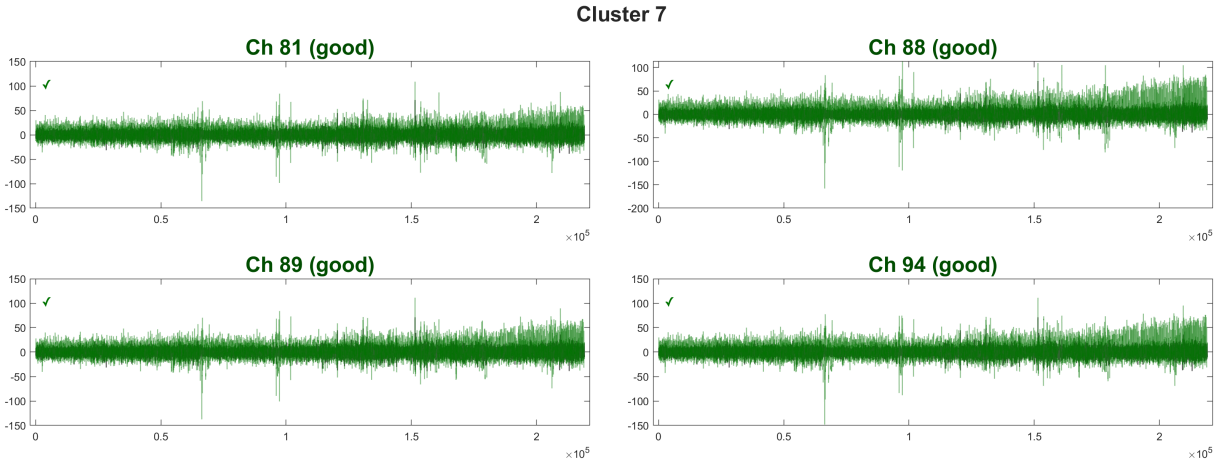


Figure 34: **Page 8, cluster 7.** These channels are fine. There are some unusual segments with increasing variance over time, which is likely the reason these channels were grouped together. These plots did not get tagged as suspicious because they were not flagged by the general checks; however, the cluster size is small and therefore the UI shows this cluster.

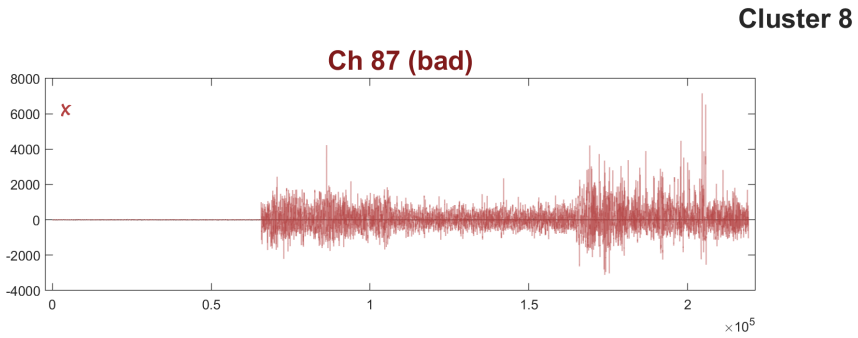


Figure 35: **Page 9, cluster 8 (singleton).** This channel was successfully tagged as bad. The amplitude and morphology of the data suggest that this channel may have lost connection to the scalp partway through the recording.

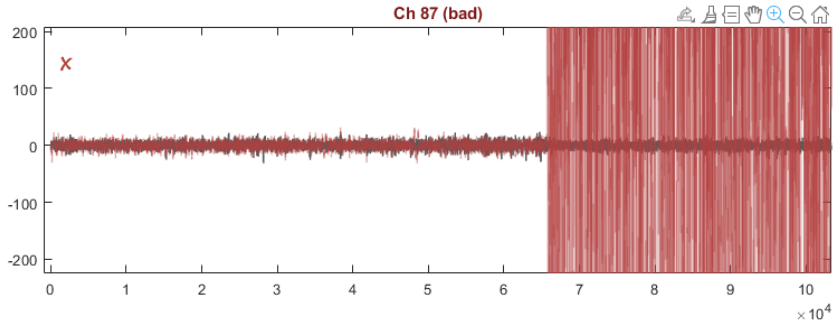


Figure 36: **Channel 87 (for context) — zoomed in.** We already selected channel for removal and will not change the decision. After zooming in, we can see artifacts of channel 87 starts around after sample 6.6×10^4 and continuous thereafter. Therefore, this channel cannot be salvaged by replacing the large transients with zeroes.

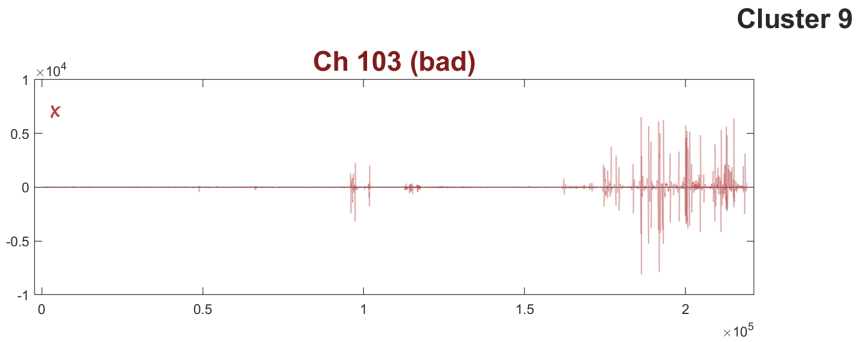


Figure 37: **Page 10, cluster 9 (singleton).** This channel was successfully tagged as bad. This channel is marked by intermittent, high-amplitude transients and should be removed.

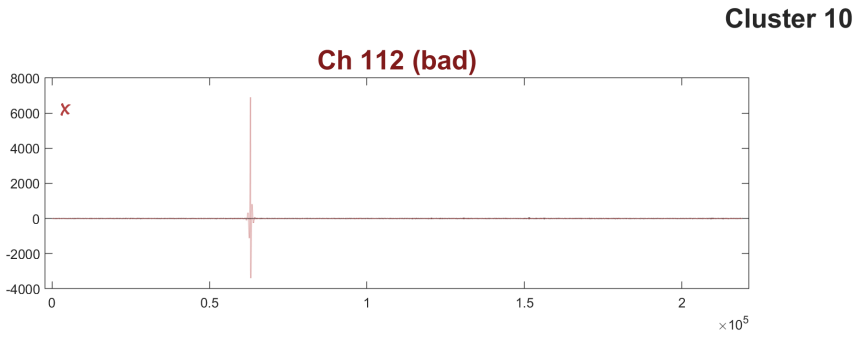


Figure 38: **Page 11, cluster 10 (singleton).** This channel was successfully tagged as bad. The transient around sample 1.56×10^5 is too large and will affect ICA. Like Channel 49, channel 112 has low neighbor dissimilarity and temporal variance equivalent to baseline. Therefore, the artifact is likely to be present only in a small time frame. In the future, this channel is a candidate for neighbor interpolation over the duration of the transient.

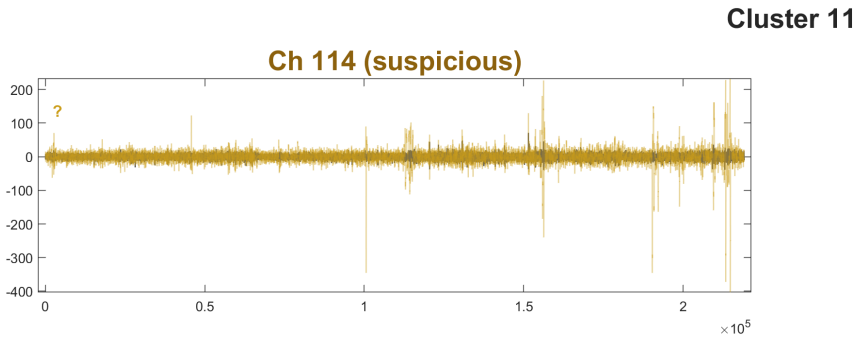


Figure 39: **Page 12, cluster 11 (singleton).** This channel is more challenging to characterize: two of the strong high-amplitude transients here are present in channel 91 and 49, both bad channels that were tagged for removal. The transients found in channel 114 do not appear to repeat in other channels. This channel does not look too bad, but not good either. However, looking at Figure 23, we can see that the suspicious scores are low. The neighbor dissimilarity, max amplitude, and variance are close to the baseline values. This channel will probably need some inspection after ICA and may require neighbor interpolation during the transient segments.

Decisions:

- None, all bad channels were correctly tagged.
- **Bad Channels (to be removed from dataset):** [91 105 49 87 103 112]
- Keep an eye on channel 114.

Override and Before vs. After visualization.

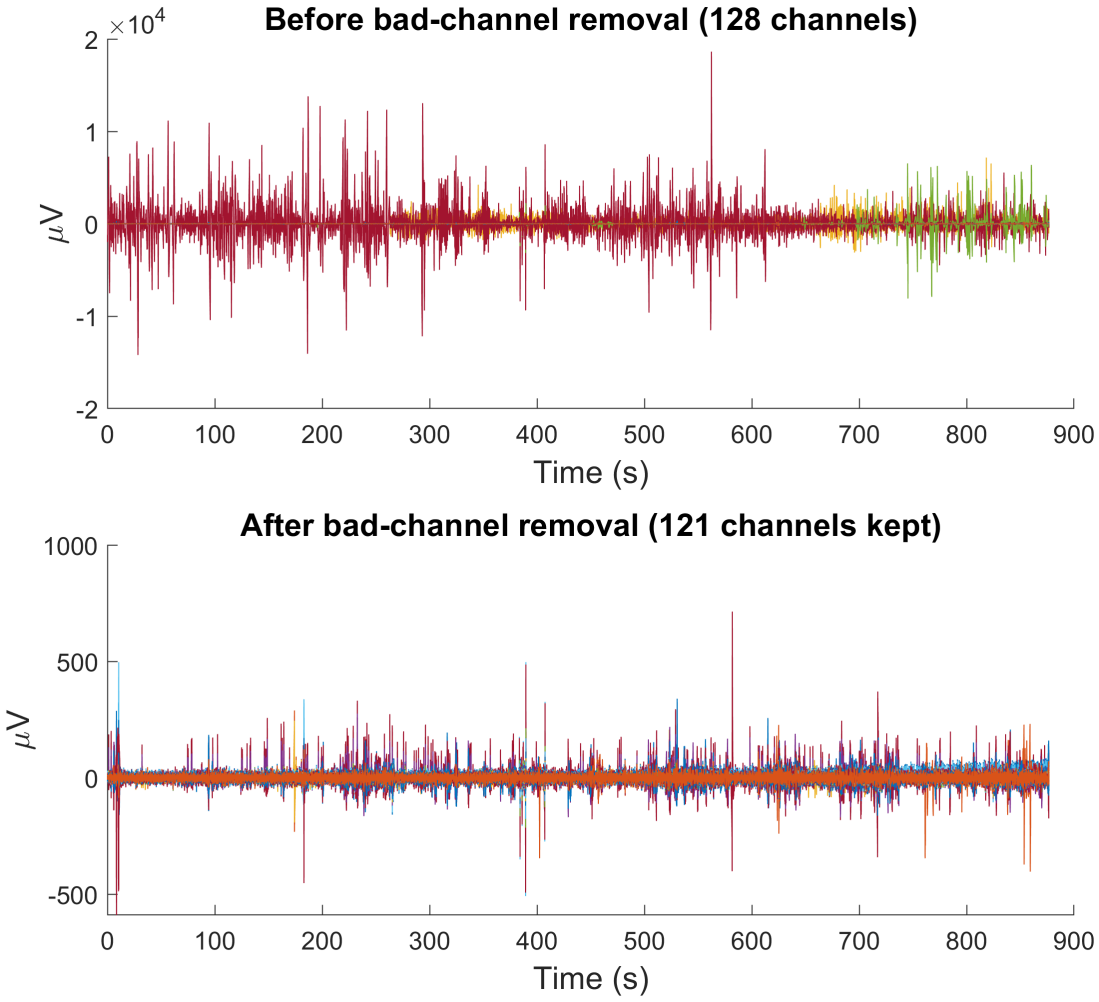


Figure 40: **EEG stacked plot — before vs. after bad-channel removal.** Stacked traces of all 128 channels before (top) and after (bottom) removing bad channels. This overview plot helps assess whether the cleaned dataset shows smoother, more consistent activity and whether visibly corrupted channels were successfully removed.

Summary

This recording exhibited moderate artifacts, including several channels with large, repeated transients and others with long stretches of non-physiological activity. Most channels within the main cluster were preserved, including eye channels and channels whose large high-amplitude transients were shared across multiple electrodes and therefore suitable for ICA removal. In contrast, singleton clusters and a few small clusters revealed channels with unique, high-magnitude artifacts that would distort ICA, and these were removed as tagged. Overall, `markSusChs()` correctly isolated a small set of genuinely bad channels while avoiding over-pruning channels that participate in global artifact patterns.

5.2 Example 3:

Data summary

Data used

- Dataset: `data3.mat`
- Channels: **129** scalp channels (EGI-style)
- Sampling rate: **250 Hz**
- Notes: **low noise. Manual interaction required.**

Cluster structure and suspicious scores

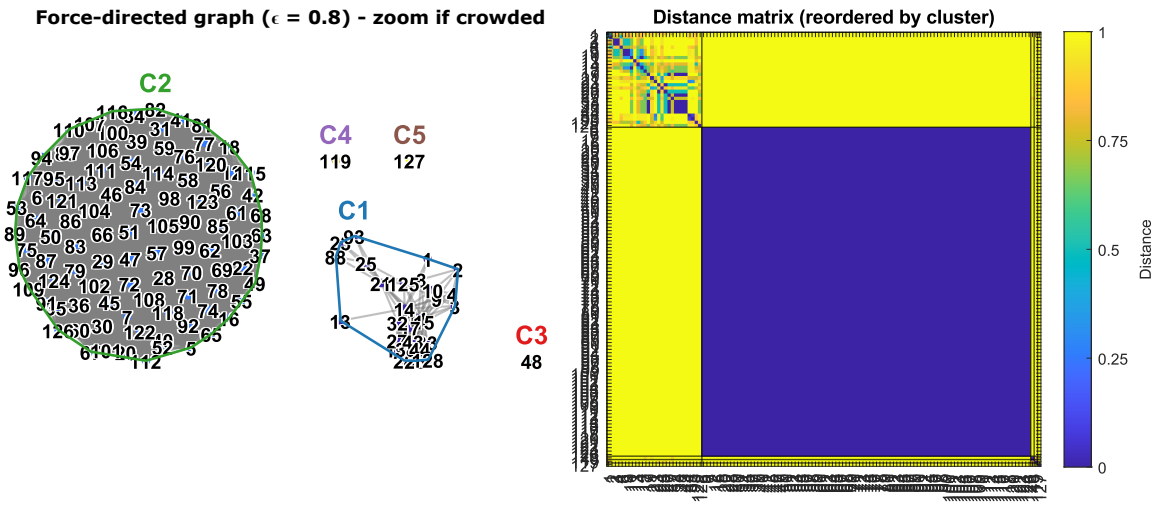


Figure 41: **Example 3 — high-amplitude transient pattern clusters and distance matrix.** *Left:* Force-directed graph showing high-amplitude transient pattern clusters. *Right:* Distance matrix reordered by cluster index.

Cluster overview

- **Cluster C2 (98 channels).** This cluster is represented by the large blue square on right plot of Figure 41 and is the cluster of channels without high transients (technically speaking, it is possible that they all have the same transients). Channels in this cluster are less likely to be bad or suspicious.
- **Clusters C4, C5, C3.** Three singletons to review.
- **Cluster C1.** One moderately sized cluster to review.

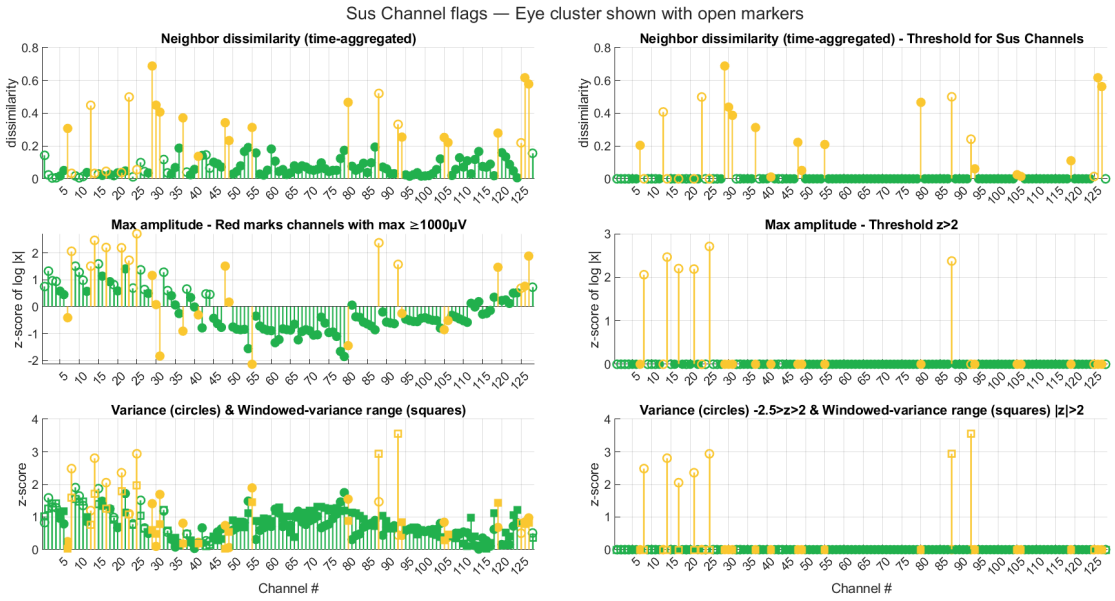


Figure 42: **Example 3 — suspicious scores and thresholds.** *Left*: Feature scores per channel. *Right*: Channels exceeding suspicious / bad thresholds. There are multiple suspicious channels, calling for detailed inspection of this recording. Remember to reference these flags when assessing a channel and deciding whether to tag it as bad.

Observations:

- Channel 88 exhibits high dissimilarity, high amplitude, and high variance — strongly suspicious.
- Channel 29 shows markedly elevated dissimilarity.
- Channels 126 and 127 are cheek electrodes; elevated metrics here are expected and do not necessarily imply bad channels.
- Eye channels receive right-of-way in the decision process. For example, channel 125 shows high amplitude and variance, but if it corresponds to eye activity, it should *not* be marked as a bad channel automatically.
- Channels 80, 23, 13, and 30 warrant closer inspection.

UI Panels and Manual Decisions

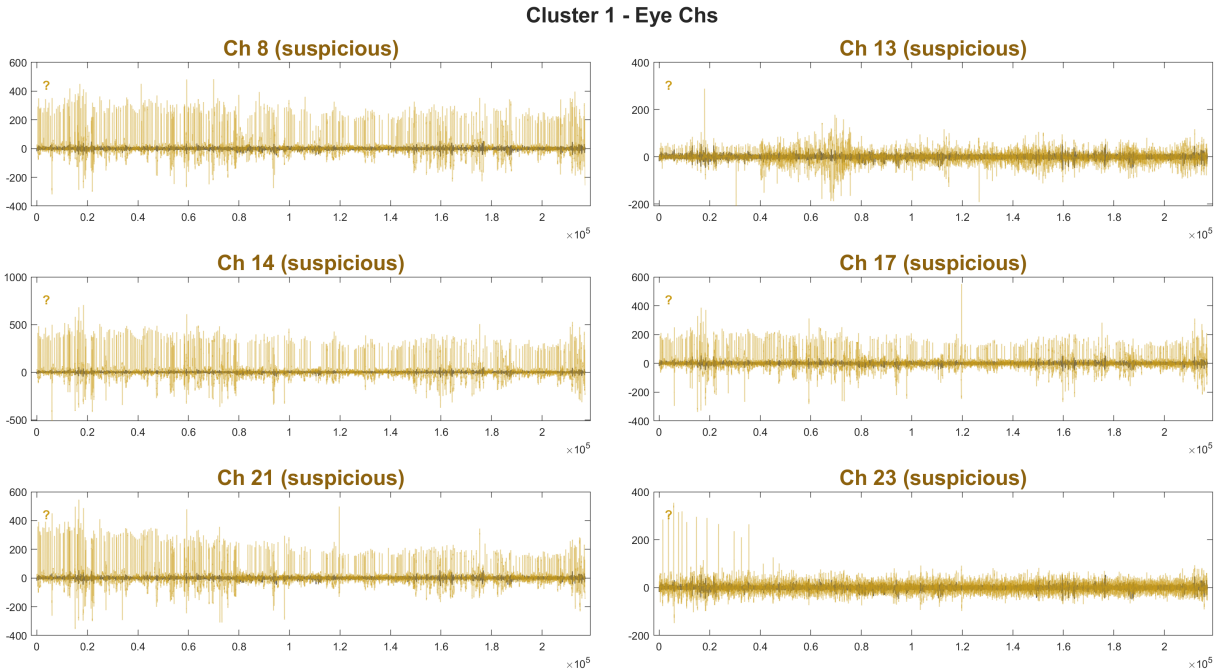


Figure 43: **Page 1, cluster 1 — eye channels.** All these channels are eye-related channels except number 13. For 129-channel EGI, 13 is closer to Cz. It may have had some transient similarities that made it fall into this cluster. Channel 13 has an irregular signal and high dissimilarity score (Figure 42). Therefore, we will click on channel 13's plot until it turns red to flag it for removal. Channel 23 is another suspicious channel because of its high neighbor dissimilarity. Channel 23 is not an eye electrode, but it is close to them. Those high transients may be related to eye activity and it appears in following some EOG activity. Channel 23 is a bit noisy, but it follows the reference channel outside quite well (zoom in MATLAB example). We will keep channel 23.

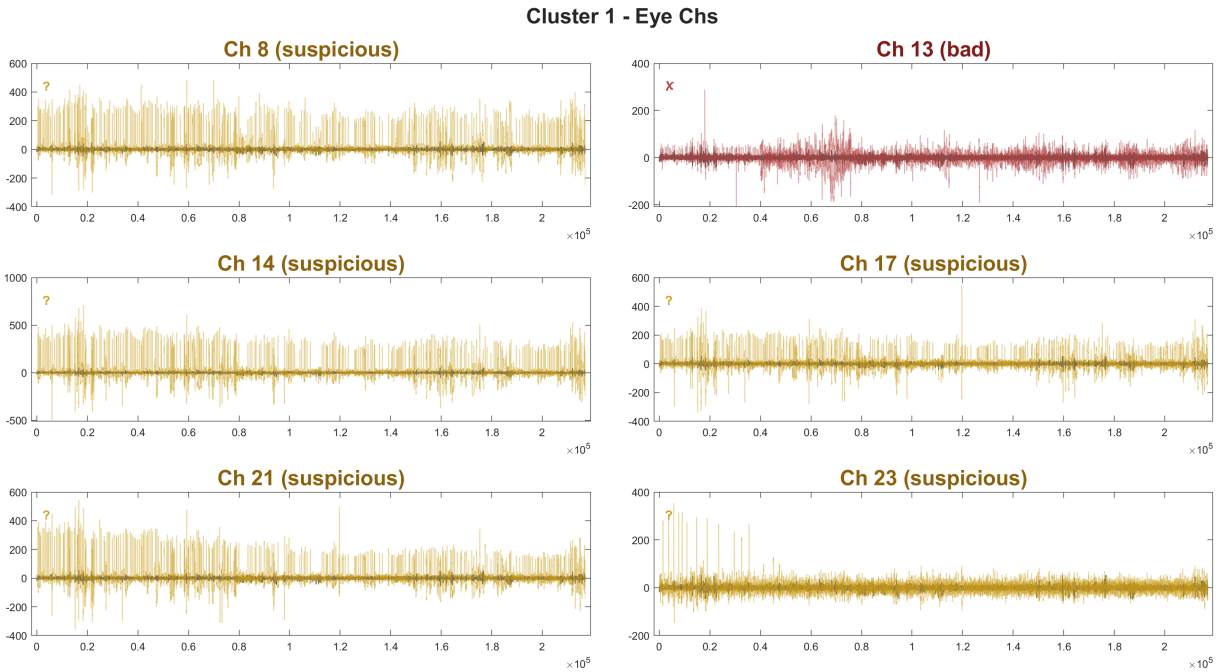


Figure 44: **Page 1, cluster 1 — eye channels (after review).** This is how page 1 (formerly Figure 43) should look before proceeding. We have flagged channel 13 for removal.

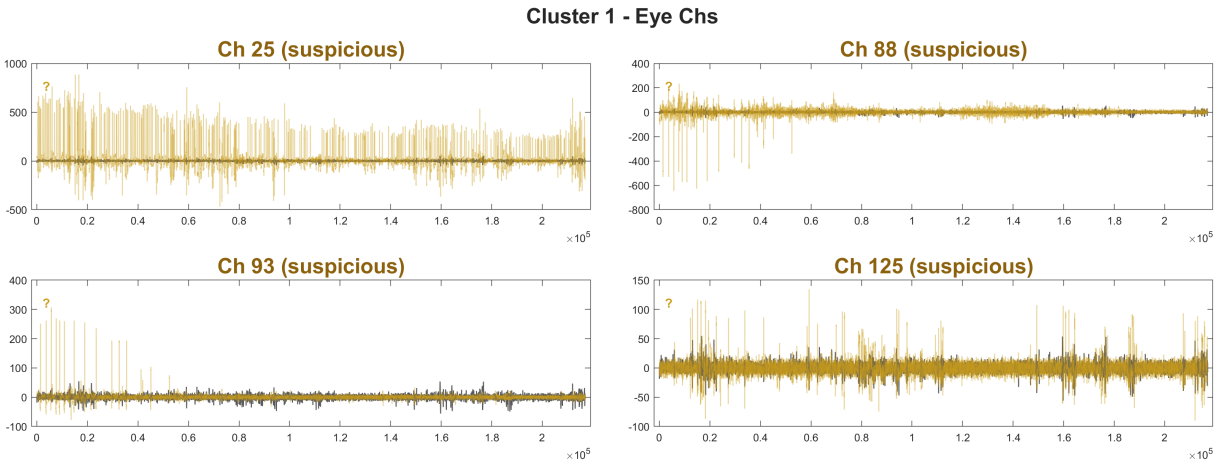


Figure 45: **Page 2, cluster 1 — eye channels.** Channels 88, 93, and 23 share similar transient patters, and they could be passed to be fixed with ICA. However, channel 88 has voltage drift (zoom in for better view), high neighbor dissimilarity, high amplitudes, and high temporal variance. Therefore, we will flag it for removal.

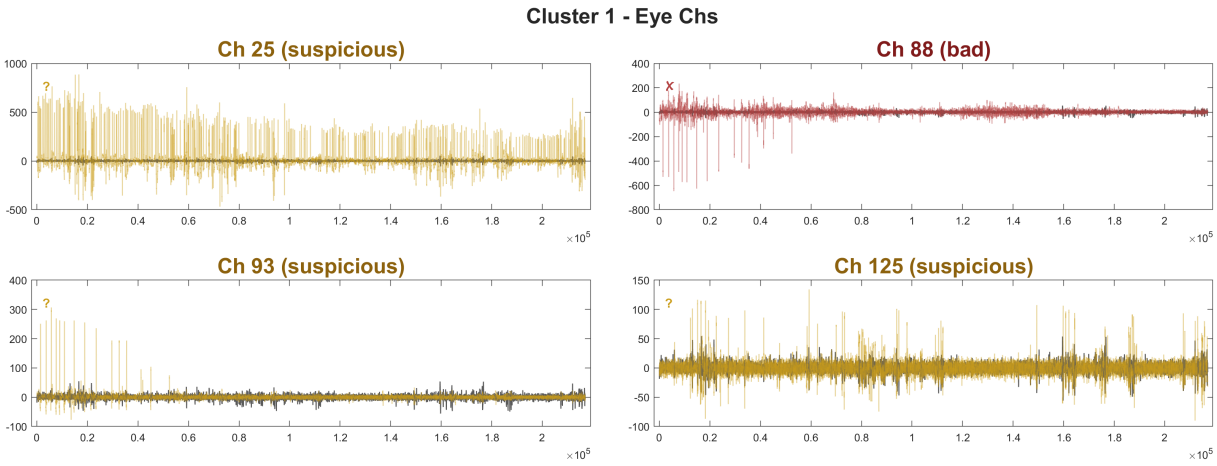


Figure 46: **Page 2, cluster 1 — eye channels (after review).** This is how page 2 (formerly Figure 45) should look before proceeding. We have flagged channel 88 for removal.

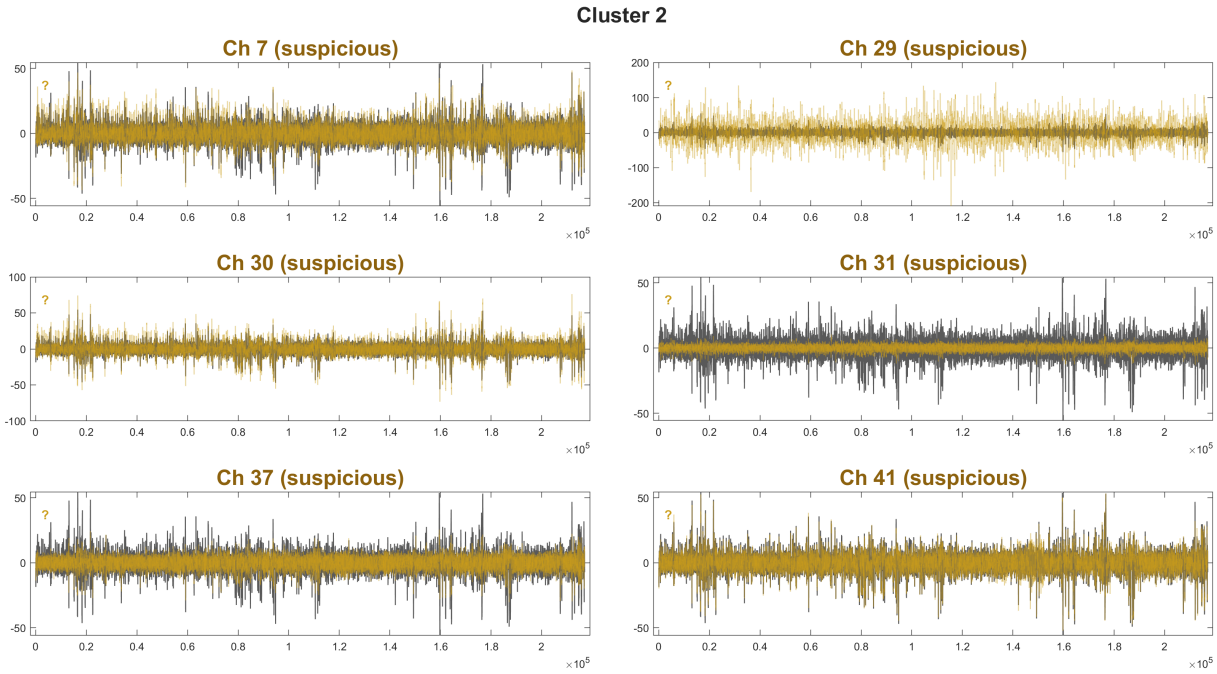


Figure 47: **Page 3, cluster 2.** Channel 29 has high neighbor dissimilarity and is noisier than other channels in this cluster throughout. Moreover, while it is not easy to see at first, channel 29 also has lower "zero-crossings rate" (crossing zero less often than the rest of the channels). In contrast, the other channels have lower-amplitude artifacts that do not warrant removal. Therefore, we will reject only channel 29 from this set.

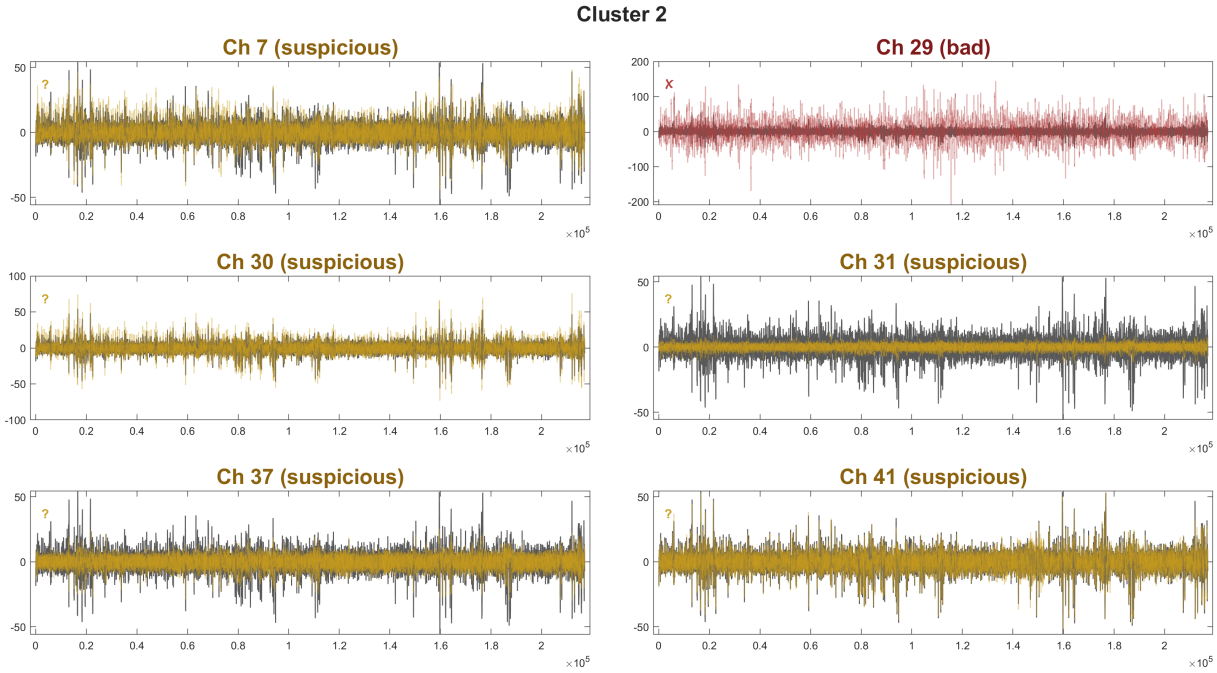


Figure 48: **Page 3, cluster 2 (after review).** This is how page 2 (formerly Figure 47) should look before proceeding. We have flagged channel 29 for removal.

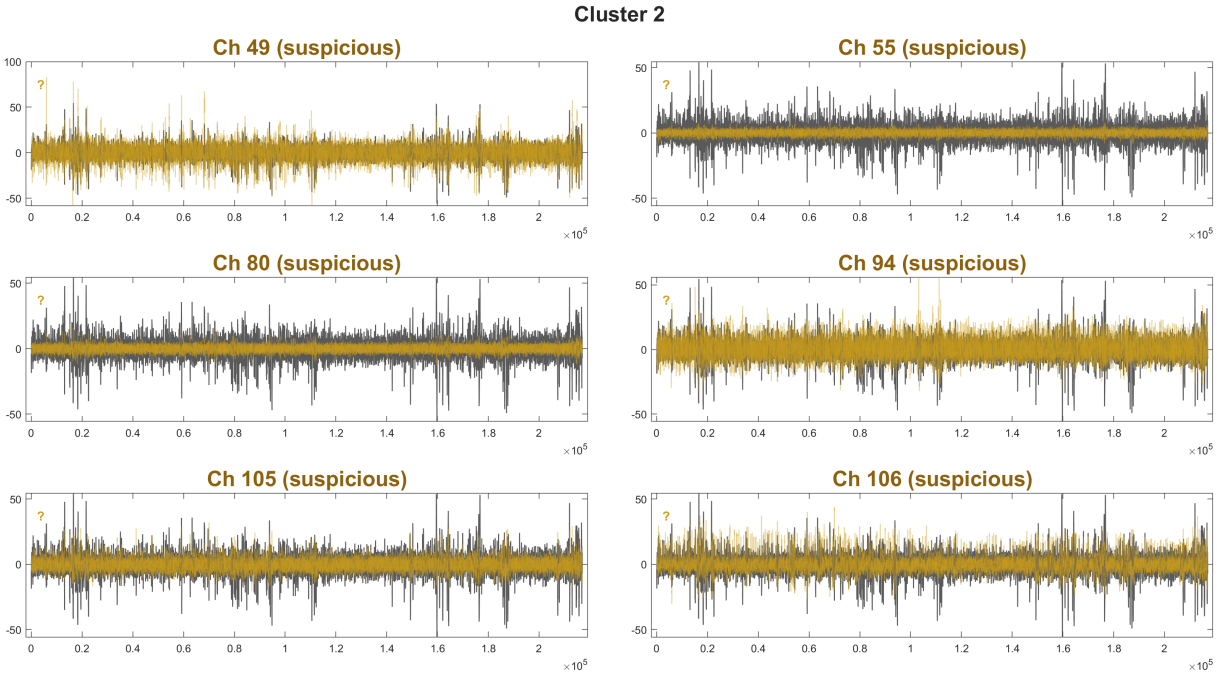


Figure 49: **Page 4, cluster 2.** These channels are fine, and we would not change any channel to bad. Channel 80 has high neighbor dissimilarity, but the visualized data look acceptable and there are no other flags. Therefore, we will keep it.

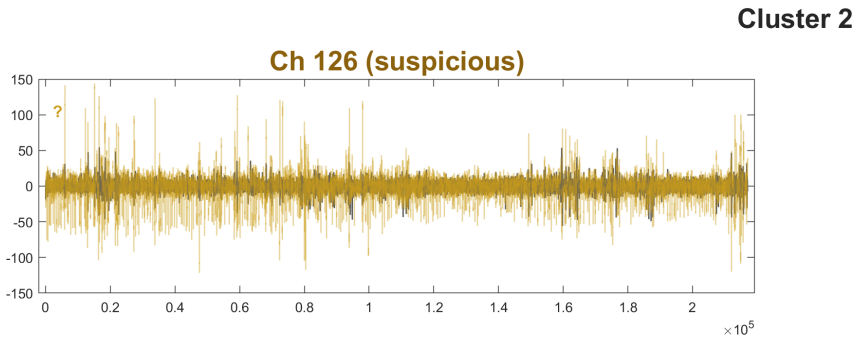


Figure 50: **Page 5, cluster 2.** This channel is fine. Channel 126 is a cheek electrode, and its higher neighbor dissimilarity is expected due to the large spatial separation from its assigned neighboring channels.

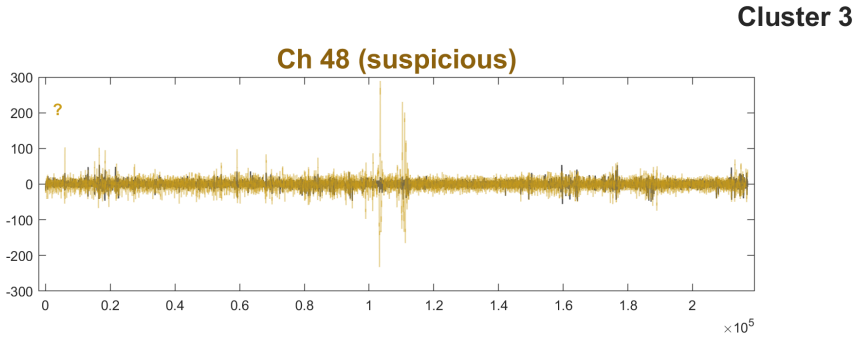


Figure 51: **Page 6, cluster 3 (singleton).** This channel has some transients, but they are not too large. Perhaps we can remove them after ICA if they persist.

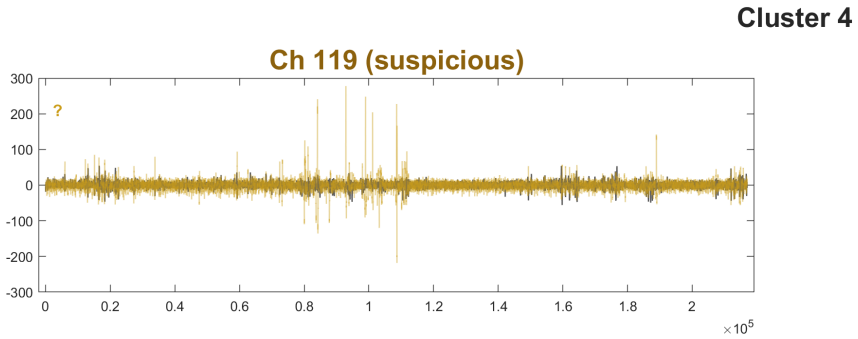


Figure 52: **Page 7, cluster 4 (singleton).** This channel also has some transients, but they are not too large. We will attempt to address the transients locally later in the broader cleaning pipeline rather than reject the entire channel.

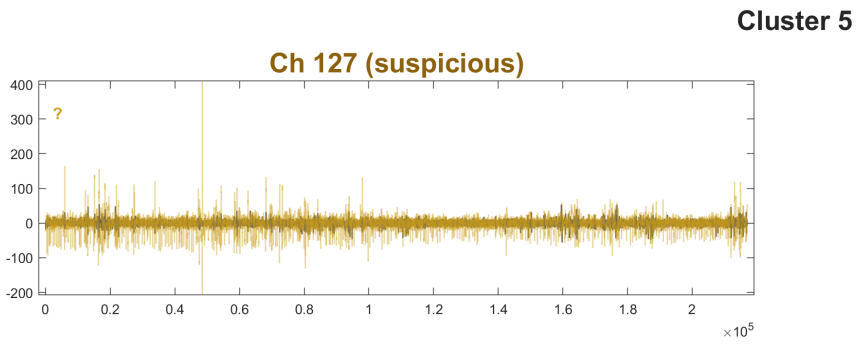


Figure 53: **Page 8, cluster 5 (singleton).** This channel is fine. Channel 127 is a cheek electrode, and its higher neighbor dissimilarity is expected due to the large spatial separation from its assigned neighboring channels.

Decisions:

- Manually tagged bad channels.
- **Bad Channels (to be removed from dataset):** [13 88 29]

Override and Before vs. After visualization.

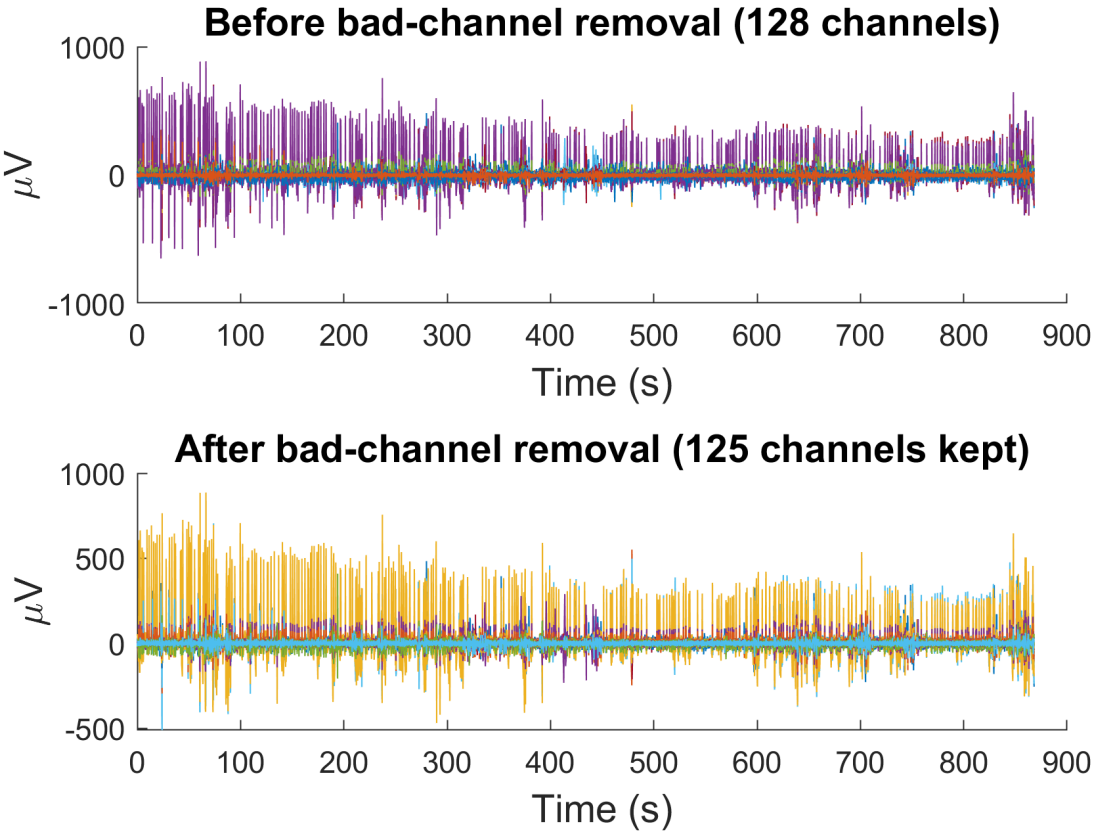


Figure 54: **EEG stacked plot — before vs. after bad-channel removal.** Stacked traces of all 128 channels before (top) and after (bottom) removing bad channels. This overview plot helps assess whether the cleaned dataset shows smoother, more consistent activity and whether visibly corrupted channels were successfully removed.

Summary

This dataset exhibited relatively low noise, but several channels required manual inspection due to subtle artifacts. Few channels showed irregular drift, reduced zero-crossing rates, or transient patterns inconsistent with neighboring activity. These issues were identified through UI review rather than clustering alone, leading to the removal of a small set of channels with clear non-physiological behavior. Overall, `markSusChs()` provided a reliable starting point, and guided manual interaction ensured that only genuinely problematic channels were removed.

CHAPTER 6

Appendix: Helper functions & Neighbor Electrode Maps

6.1 Helper Functions

This section contains brief descriptions of the helper functions, which are all contained in the `HelperFunctions` folder. Helper functions are documented in alphabetical order. For each helper function, we provide the syntax, a brief narrative description of its inputs and outputs, and its usage by other user-called and/or helper functions.

`clusterByThreshold()`

Syntax

```
[idx, info] = clusterByThreshold(D, eps)
```

Description. Clusters channels by connecting all pairs with $D \leq \varepsilon$ (symmetric adjacency) and extracting connected components. Returns cluster IDs and diagnostic information such as adjacency matrix, epsilon value, number of clusters, and their sizes.

Usage. Called by `markSusChs.m` after computing the Jaccard distance matrix. Transforms the Jaccard distances into discrete channel clusters, which are then labeled (e.g., eye cluster).

`jaccardDistanceChannels()`

Syntax

```
[D, S, O, n] = jaccardDistanceChannels(X, opts)
```

Description. Computes pairwise "Jaccard-like" similarity and distance between EEG channels based on binarized event matrices. Accepts logical input or thresholds numeric data (either with a scalar threshold or a per-channel vector). Handles zero-event edge cases: if both channels have no events, similarity is set to 1 ($S = 1, D = 0$); if only one channel has no events, similarity is 0 ($S = 0, D = 1$). Returns the distance matrix `D`, similarity matrix `S`, number of overlapping events `O`, and per-channel event counts `n`.

Usage. Called by `markSusChs.m` after computing high-z event matrices from windowed features. The resulting Jaccard distance matrix quantifies how often channels share similar artifact patterns.

`makeWindowsFromX()`

Syntax

```
[xOut, win] = makeWindowsFromX(xIn, fs, winSec, hopSec)
```

Description. Partitions a continuous EEG signal $[C \times T]$ into overlapping or non-overlapping temporal windows. Each window has duration `winSec` and hop size `hopSec`, both defined in seconds. Returns the windowed data array `xOut` of size $[C \times L \times W]$, where L is the number of samples per window and W the total number of windows. Also returns the structure `win`, containing metadata such as window start and end indices.

Usage. Called by `markSusChs.m` to segment each channel into time windows before computing neighbor correlations and windowed z-scores. This enables time-resolved evaluation of artifacts and supports later smoothing in `smoothFeature()`.

`neighborCorrFromWindows()`

Syntax

```
[R, Rcell] = neighborCorrFromWindows(Xwin, forceNNeighbors, mode, neighborDir)
```

Description. Computes the correlation between each channel and its spatial neighbors within each time window, providing a measure of local signal similarity over time.

Usage. Called by `markSusChs.m` after windowing the EEG data with `makeWindowsFromX()`. The resulting correlation matrix `R` is converted to neighbor dissimilarity ($1 - |\text{corr}|$) and later smoothed by `smoothFeature()` to derive features for bad-channel detection.

plotBefore()

Syntax

```
fig = plotBefore(plotStruct)
```

Description. Generates a multi-panel figure summarizing pre-cleaning channel features such as amplitude, variability, and neighbor dissimilarity, highlighting suspicious or clustered channels.

Usage. Called by `markSusChs.m` to visualize channel metrics before manual review, providing an overview of automatically detected outliers and artifact clusters.

plotBeforeAfter()

Syntax

```
fig = plotBeforeAfter(xIn, xGood, plotStruct)
```

Description. Creates a side-by-side comparison of EEG data and channel features before and after bad-channel removal, showing improvements in amplitude, variability, and neighbor dissimilarity.

Usage. Called by `markSusChs.m` after the review step to document the effect of excluding bad channels and to confirm that signal quality improved following cleaning.

plotClustersGraphAndHeatmapV2()

Syntax

```
fig = plotClustersGraphAndHeatmapV2(D, eps, idx, clusterNames)
```

Description. Plots the structure of channel clusters as both a graph and a heatmap, using the Jaccard distance matrix and cluster assignments to visualize how channels group together.

Usage. Called by `markSusChs.m` after clustering with `clusterByThreshold()`. Used to inspect and verify the spatial or temporal coherence of artifact-related channel clusters before manual review in `reviewBadChsUI()`.

reviewBadChsUI()

Syntax

```
[maskOut, badList] = reviewBadChsUI(X, t, maskIn, groups, titles, ...
    INFO, gridCols, ref, alpha, saveFigs)
```

Description. Provides an interactive graphical interface for reviewing channel quality and confirming or modifying automatic classifications. Channels are displayed in grouped panels (e.g., spatial or cluster-based). Each plot shows the raw time series, optional reference traces. Users can toggle channel states (good, suspicious, or bad) using keyboard shortcuts or mouse clicks.

Usage. Called by `markSusChs.m` after automatic feature computation and clustering. Used as the final human-validation step to confirm or adjust which channels should be rejected.

`smoothFeature()`

Syntax

```
Xs = smoothFeature(X, smoothWin)
```

Description. Smooths a feature matrix $[C \times W]$ across time windows using a moving median and limits extreme values through per-channel winsorization. Each channel is smoothed independently along the window dimension using a median filter.

Usage. Called by `markSusChs.m` to temporally stabilize windowed features such as neighbor dissimilarity and windowed variance before computing summary metrics. This reduces the influence of high-amplitude transients or noise bursts on bad-channel classification.

`zScoreRobust()`

Syntax

```
Z = zScoreRobust(X)
```

Description. Computes robust z-scores for each channel using the median and median absolute deviation (MAD). This provides a scale-invariant measure of deviation that is less sensitive to outliers or non-Gaussian artifacts in EEG data.

For each channel x_i, the robust z-score is given by

$$z_i = \frac{x_i - \text{median}(x_i)}{1.4826 \times \text{MAD}(x_i)},$$

Usage. Called by `markSusChs.m` to obtain robustly standardized channel amplitudes prior to thresholding and clustering. These z-scores form the basis for detecting large transient deviations and constructing binary event matrices for the Jaccard-like distance calculation.

6.2 Neighbor Electrode Maps

The `EGIMontages` folder contains a set of `.mat` files defining neighbor relationships between electrodes for different EGI HydroCel montages (e.g., 64, 124, 128 channels). Each file, named `neighboringElectrodes#.mat`, stores a single cell array `neighbors`, where `neighbors{c}` lists the indices of electrodes considered spatial neighbors of channel c.

The neighbor definitions correspond to the standard Electrical Geodesics, Inc. (EEG) HydroCel Geodesic Sensor Net layouts [9, 6], or the BioSemi electrode coordinates and cap specifications [3]. Relevant manufacturer documentation is available online.[10]

Different EEG systems or custom montages can be supported by providing a corresponding `neighboringElectrodes#.mat` file that defines appropriate neighbor relationships.

`neighboringElectrodes#.mat`

Syntax

```
S = load('neighboringElectrodes128.mat'); % loads cell array: S.neighbors
```

Usage. Implicitly loaded by `neighborCorrFromWindows()` via:

```
neiFn = sprintf('neighboringElectrodes%d.mat', Cused);
S = load(neiFn); neighbors = S.neighbors;
```

where `Cused` is the number of channels used (e.g., 64, 124, 128). The neighbor lists drive the per-window neighbor correlations, which are converted to dissimilarity features for bad-channel scoring.

- These neighbor maps correspond to EGI HydroCel or BioSemi nets.
- If you use a different montage or channel count, provide the matching `neighboringElectrodes#.mat`.
- For more information on included montages or custom maps, refer to subsection 2.5.

[10] Electrical Geodesics, Inc. (2007). *HydroCel Geodesic Sensor Net Technical Manual*, Appendix B; BioSemi *Headcap Specifications and Electrode Layouts*: `https://www.biosemi.com/headcap2.htm`.

CHAPTER 7

References

[1] Anthony J. Bell and Terrence J. Sejnowski. “An information-maximization approach to blind separation and blind deconvolution”. In: *Neural Computation* 7.6 (1995), pp. 1129–1159.

[2] Nima Bigdely-Shamlo, Tim Mullen, Christian Kothe, Kyung-Min Su, and Kay A. Robbins. “The PREP pipeline: standardized preprocessing for large-scale EEG analysis”. In: *Frontiers in Neuroinformatics* 9 (2015), p. 16. DOI: 10.3389/fninf.2015.00016.

[3] BioSemi B.V. *BioSemi Headcap and Electrode Layout Specifications*. https://www.biosemi.com/headcap2.htm. Technical documentation for Electrode Layouts.

[4] Radoslaw Martin Cichy and Dimitrios Pantazis. “Multivariate pattern analysis of MEG and EEG: A comparison of representational structure in time and space”. In: *NeuroImage* 158 (2017), pp. 441–454. DOI: https://doi.org/10.1016/j.neuroimage.2017.07.023.

[5] Jacek P. Dmochowski, Paul Sajda, Joao Dias, and Lucas C. Parra. “Correlated Components of Ongoing EEG Point to Emotionally Laden Attention — A Possible Marker of Engagement?” In: *Frontiers in Human Neuroscience* 6 (2012). DOI: 10.3389/fnhum.2012.00112.

[6] Electrical Geodesics, Inc. *Geodesic Sensor Net Technical Manual*. Technical manual. 2007.

[7] Paul Jaccard. “Étude comparative de la distribution florale dans une portion des Alpes et du Jura”. In: (1901).

[8] Markus Junghöfer, Thomas Elbert, Don M. Tucker, and Brigitte Rockstroh. “Statistical control of artifacts in dense array EEG/MEG studies”. In: *Psychophysiology* 37.4 (July 2000), pp. 523–532. DOI: 10.1111/1469-8986.3740523.

[9] Phan Luu and Thomas Ferree. “Determination of the Geodesic Sensor Nets’ Average Electrode Positions and Their 10–10 International Equivalents”. In: *Technical Note* (Jan. 2000).

[10] Amilcar J. Malave and Blair Kaneshiro. *Bad-Channel Detection Module: A MATLAB framework for semi-automated EEG bad-channel detection and review*. Version 1.1. 2025.

[11] Amilcar J. Malave and Blair Kaneshiro. “Example EEG Data for the SENSI EEG PREPROC Bad-Channel Detection Module”. In: *Stanford Digital Repository*. Available at: https://purl.stanford.edu/dg856vy8753; accessed 2025-12-17. 2025.