

# FaultNVC: Earthquake Fault Plane Extractor through Point-Cloud Normal Vector Clustering for Hypocenter Distributions

Yasunori Sawaki<sup>1</sup>, Yoshihiro Sato<sup>2,3</sup>, and Takahiko Uchide<sup>1</sup>

<sup>1</sup>Research Institute of Earthquake and Volcano Geology, Geological Survey of Japan, AIST, Japan

<sup>2</sup>Faculty of Design and Data Science, Tokyo City University, Japan

<sup>3</sup>Artificial Intelligence Research Center, AIST, Japan

v0.2.0: 06 January 2025

## 1 Introduction

The package `FaultNVC` extracts planar fault alignments in hypocenter distributions through hierarchical clustering of hypocenter positions and point-cloud normal vectors (PCNVs).

Hypocenter alignments typically reflect complex crustal fault structures. Recent developments of hypocenter-based clustering opened the possibility for objective modeling of crustal faults (e.g., [Ouillon et al., 2008](#); [Kamer et al., 2020](#); [Truttmann et al., 2023](#)), however, spatial clustering still struggles with constructing complex fault geometries.

A recent study demonstrated that incorporating PCNVs into hypocenter clustering is efficient for constructing intricate planar features without specifying the number of fault planes (`TwoS-Clust`; [Sato et al., 2022, 2023](#)). PCNV refers to the normal vector of a plane representing the distribution of points in the vicinity of a target point. It is a feature used in fields such as object recognition. The fault plane extraction using `TwoS-Clust` consists of two steps of clustering. First, clustering of PCNVs is performed to extract groups of hypocenters expected to have similar strike and dip.

---

\*[sawaki.yasunori@aist.go.jp](mailto:sawaki.yasunori@aist.go.jp)

Then, spatial clustering based on hypocenter positions is applied to the segmented groups to extract groups of hypocenters that are distant from each other in 3D space. For the latter clustering, we use HDBSCAN (Campello et al., 2013; McInnes et al., 2017), which does not require specifying the number of clusters. HDBSCAN is a method that can identify clusters of various shapes and densities and effectively exclude noise points. However, since TwoS-Clust performs clustering in two steps, the number of parameters increases.

After their two-step clustering method of PCNVs and hypocenter positions, we have developed a single clustering algorithm using HDBSCAN, extracting complex **Fault** planes through PCNV Clustering along with hypocenter positions (FaultNVC; Sawaki et al., 2025). FaultNVC is a Python package built upon a robust class-based architecture and requires minimum user coding.

The instruction below shows how to setup and run FaultNVC for extracting intricate fault planes from hypocenter distributions.

## 2 Installation

### 2.1 Setup

You first need to unzip the downloaded file. The following instructions assume that the unzipped package is placed in \$HOME/FaultNVC directory. We recommend using a conda environment to establish the virtual environment. You need to install FaultNVC using pip locally. In the instructions below, "nvc" is just used as an example environment name.

- Using conda/mamba (recommended)

You may replace mamba with conda in the following instruction

---

```
$ cd $HOME/FaultNVC      # move to the unzipped directory
$ mamba create -n nvc python=3.11 # python 3.10+ required
$ mamba activate nvc     # do not forget to activate the environment

# Install required packages
(nvc) $ mamba env update -f ./environment_dep.yml

# pytest for installation test (skippable)
(nvc) $ mamba install pytest
```

---

Then install FaultNVC

```
# Install FaultNVC locally  
(nvc) $ pip install .
```

```
# Check the installation  
(nvc) $ mamba list FaultNVC
```

---

This is optional, but you may install optional packages

---

```
(nvc) $ mamba env update -f ./environment_opt.yml
```

---

- Using pip on venv (alternative)

```
# Check if an existing python has version 3.10+  
$ python -V
```

```
# Create an environment using venv  
$ mkdir $HOME/venv_src # Any name is OK  
$ cd $HOME/venv_src  
$ python -m venv nvc # Specify python3.10 or python3.11 if needed  
$ source nvc/bin/activate
```

```
# Install FaultNVC locally  
(nvc) $ cd $HOME/FaultNVC # move to the unzipped directory  
(nvc) $ pip install . # install FaultNVC and other packages
```

```
# Check the installation  
(nvc) $ pip show FaultNVC
```

```
# pytest for installation test (skippable)  
(nvc) $ pip install pytest
```

---

## 2.2 Test your installation

You may skip the installation test.

---

```
# Run test files  
(nvc) $ (cd tests && pytest)
```

---

Check if all tests are passed. If any test is failed, retry installation.

## 2.3 Requirements

FaultNVC requires Python 3.10+ and following packages:

- [HDBSCAN](#) (hdbscan; [McInnes et al., 2017](#))  
(different from the class integrated in scikit-learn: [sklearn.cluster.HDBSCAN](#))
- NumPy (numpy)
- Numba (numba)
- Matplotlib (matplotlib)
- scikit-learn (sklearn)
- pandas (pandas)
- pyproj (pyproj)
- joblib (joblib)

Following packages are optional:

- [Cartopy](#) (cartopy)
- Plotly (plotly)
- [Colorcet](#) (colorcet)
- tqdm (tqdm)

## 3 How to Use

### 3.1 Preface

The class FaultHDBSCAN has numerous methods, performing all the necessary processes for this analysis. The input data is three-dimensional point-cloud data (north, east, and depth), such as hypocenter distributions. The general steps are:

1. Initialize FaultHDBSCAN
2. Load point-cloud data and compute PCNVs
3. Run HDBSCAN (clustering) under various parameter sets

Parameter and argument settings for FaultHDBSCAN are described in Section [4.2.1](#).

## 3.2 Compute point-cloud normal vectors

As the first step, PCNVs are estimated using KNN-PCA. Important parameters described in [Sawaki et al. \(2025\)](#) are following:

- `max_neighbors_normal_vector` ( $k^{\text{NV}}$ ): Maximum number of nearest neighbors for KNN-PCA to estimate PCNVs
- `max_dist_neighbors` ( $r^{\text{NV}}$ ): Maximum allowable distance for neighbor search in the same dimension of distance as the input point cloud. This prevents including distant points for a point of interest with low density

Here we use a sample dataset `sample1` in `$HOME/FaultNVC/examples/sample1` the following procedure.

---

```
import numpy as np
import pandas as pd

from fnvc import FaultHDBSCAN

# Sample dataset (random values)
df_input = pd.read_csv(
    '$HOME/FaultNVC/examples/sample1/points.txt',
    sep=r'\s+', header=None, names=('E', 'N', 'D'),
)

# Parameters for computing point-cloud normal vectors
max_neighbors_normal_vector = 150
min_neighbors_normal_vector = 20
max_dist_neighbors = 2.0
min_planarity = 1.25

# Initialize FaultHDBSCAN
clusterer = FaultHDBSCAN(
    max_neighbors_normal_vector=max_neighbors_normal_vector,
    min_neighbors_normal_vector=min_neighbors_normal_vector,
    max_dist_neighbors=max_dist_neighbors,
    min_planarity=min_planarity,
)
```

---

The method `.calc_normal_vector()` calculates PCNVs by KNN-PCA.

---

```

# Load point-cloud data and compute point-cloud normal vectors
clusterer.calc_normal_vector(
    df_input,
    id_points=df_input.index.values,
    kw_points=dict(N='N', E='E', D='D'), # convert colnames to (N,E,D)
    standardize=True, # standardize positions
    max_workers=2, # parallel computing
)

```

---

The method `.plot_events()` plots normal vectors or apparent strike.

---

```

# Normal vector
fig = clusterer.plot_events(
    includes_normal=True,
    color_normal=True,
    mode='2d', # "2d", "3d", and "cartopy"
    cbar_rect=[0.15,0.15,0.2,0.2]
)

```

---



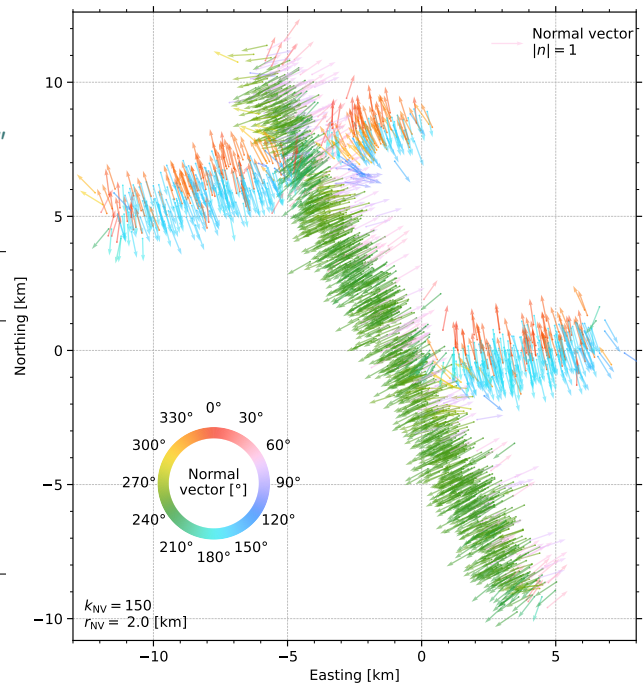
---

```

# Apparent strike
fig = clusterer.plot_events(
    includes_normal=False,
    color_normal=False,
    cbar_rect=[0.15,0.15,0.2,0.2]
)

```

---



### 3.3 Run HDBSCAN

The method `.fit_predict()` executes clustering and extracts fault planes under the specified parameter set:

- `min_samples` ( $m_{pts}$ ) – Number of nearest neighbors to calculate a core distance
- `min_cluster_size` ( $m_{clSize}$ ) – Minimum size to be considered a cluster

- `cluster_selection_epsilon` ( $\epsilon$ ) – Maximum distance to merge two clusters in the condensed dendrogram

Candidate values for  $m_{\text{pts}}$  and  $m_{\text{clSize}}$  can be selected through trial and error to ensure that the resulting clusters reflect primary plane structures or include minor clusters, setting  $\epsilon$  to 0.0. It is advisable to start with a large value for  $m_{\text{pts}}$  and  $m_{\text{clSize}}$  and then search for a smaller value that allows the desired outcome. However, choosing `min_samples` and `min_cluster_size` might not be straightforward. You may simplify the clustering by setting  $m_{\text{clSize}} = m_{\text{pts}}$  (Campello et al., 2013; McInnes et al., 2017), depending on the case.

Once you fixed  $m_{\text{pts}}$  and  $m_{\text{clSize}}$ , you may increase  $\epsilon$  to merge minor clusters. It is not likely that  $\epsilon$  exceeds 1.0 because the distance between two feature vectors ( $q_k, q_l$ ) is defined as

$$d^{\text{FV}}(q_k, q_l) = \sqrt{\|\tilde{X}_k - \tilde{X}_l\|^2 + 2(1 - |n_k \cdot n_l|)}, \quad (1)$$

where  $\tilde{X}$  is the standardized position of  $X$ :

$$\tilde{X} = \frac{X}{\sqrt{V[N] + V[E] + V[D]}}, \quad (2)$$

and  $n$  is a PCNV. A proper range of  $\epsilon$  depends on the case, but plotting a condensed dendrogram by `.plot_dendrogram()` may help.

Each call to `.fit_predict()` is assigned a unique key starting from 0. The method manages a dictionary of parameter sets and results, keyed by these unique identifiers.

---

```
# key 0
clusterer.fit_predict(
    min_cluster_size=5,
    min_samples=5,
    cluster_selection_epsilon=0.000,
)

# key 1
clusterer.fit_predict(
    min_cluster_size=5,
    min_samples=5,
    cluster_selection_epsilon=0.300,
)

# key 2
clusterer.fit_predict(
```

```

min_cluster_size=3,
min_samples=2,
cluster_selection_epsilon=0.000,
)

```

The method `.show_computed_params()` returns keys with the associated parameter sets:

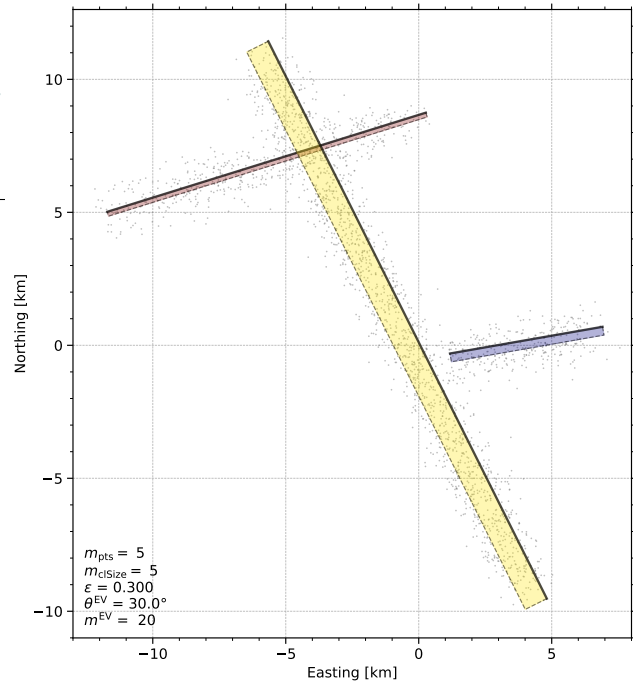
	min_cluster_size	min_samples	cluster_selection_epsilon
0	5	5	0.000
1	5	5	0.300
2	3	2	0.000

The method `.plot_planes()` creates a figure of extracted fault planes. Specify key for the associated parameter set:

```

# max_angle_diff: maximum median angle offset of
# point-cloud normal vectors and fault normal
# min_points: minimum number of points to plot a plane
fig = clusterer.plot_faults(
    key=1,
    mode='2d', # "2d", "3d", and "plotly"
    max_angle_diff=30.0, # default to 90.
    min_points=20, # default to 5
)

```





### 3.4 Output results

- Output the results as DataFrame

---

```
results_clustering, output_events = clusterer.output_planes(key=0)
```

---

- `results_clustering`: Dataframe for extracted fault parameters
- `output_events`: Dataframe for hypocenter cluster numbers

- Save the results into CSV files

---

```
filename = "path/to/samplefile0"  
clusterer.save_model(filename=filename, key=0)  
# path/to/samplefile0_planes.csv -> results_clustering  
# path/to/samplefile0_events.csv -> output_events
```

---

### 3.5 A script for sample1

You can test the procedure above by running a python script as follows:

---

```
(nvc) $ python $HOME/FaultNVC/examples/sample1/sample1.py --min_samples 2 --mode 2d  
↪ --max_angle_diff 30.0 --min_points 10
```

---

Optional arguments can be found by `$python sample1.py -h`.

## 4 API overview

### 4.1 Modules

- `fncv` – A top module, which is importable as `import fncv`
- `fncv.clustering` – Stores class `FaultHDBSCAN` and functions for clustering
- `fncv.synthetic` – Stores functions for synthetic tests
- `fncv.utils` – Stores helpful functions

### 4.2 Classes with main methods

#### 4.2.1 `fncv.clustering.FaultHDBSCAN` – Main class

Class `FaultHDBSCAN`, a subclass of `fncv.clustering.BaseClustering`, has numerous methods, performing all the necessary processes for this analysis. Note that we use the coordinate system of [North, East, Down] positive. Thus, the input data of [X,Y,Z] is transformed into [Y,X,Z] (+Z & +D for downward).

**Import** `FaultHDBSCAN` can be importable from the top module:

---

```
from fncv import FaultHDBSCAN
```

---

**Arguments** Class arguments only for KNN-PCA

- `max_neighbors_normal_vector` ( $k^{NV}$ ) – The maximum number of nearest neighbors for KNN-PCA to estimate PCNVs
- `max_dist_neighbors` ( $r^{NV}$ ) – The maximum distance to find nearest neighbors. This prevents including distant points for a point of interest with low density

**Methods** Calculating PCNVs, clustering, plotting, and saving fault parameters...

- `.calc_normal_vector(all_points, id_points=None, kw_points=None)`  
 Calculate PCNVs formed by the top two principle-component vectors. Each dataset composes of  $k^{NV}$  points retrieved using the nearest neighbor method with KDtree algorithm
  - `all_points` – ndarray or DataFrame of all data points. 1D array of [X, Y, Z] for each row
  - `id_points` – List of event IDs
  - `kw_points` – Dict for component names (N,E,D)

Returns `.normal_vectors_kneighbors` (ndarray): PCNV for each hypocenter

- `.fit_predict(min_cluster_size=5, min_samples=None, cluster_selection_epsilon=0.0)`  
 executes clustering and extracts fault planes. Upon invocation, the method generates a unique key starting from 0 and uses it to index the associated parameter set and result
  - `min_samples` ( $m_{pts}$ ) – Number of nearest neighbors to calculate a core distance. Default to None, setting to  $m_{clSize}$
  - `min_cluster_size` ( $m_{clSize}$ ) – Minimum size to be considered a cluster. Default to 5
  - `cluster_selection_epsilon` ( $\epsilon$ ) – Maximum distance to merge two clusters in the condensed dendrogram. Default to 0.0
- `.show_computed_params()` prints parameter sets for the corresponding keys in standard output. Returns None
- `.output_planes(key, min_points=5)` outputs clustering results of the extracted planes and events
  - `key` – Index of the associated parameter set and result
  - `min_points` – Minimum number of points in an output cluster. Default to 5

Returns `results_clustering` and `output_events` in `pandas.DataFrame`. The former is for the extracted fault parameters, and the latter is for hypocenter cluster numbers

- `.save_model(filename, key, min_points=5)` saves the fault model and event clusters in CSV format
  - `filename` – Path to the output file (e.g., `path/to/sample1`)
  - `key` – Index of the associated parameter set and result
  - `min_points` – Minimum number of points in an output cluster. Default to 5
- `.plot_events(mode="2d", includes_normal=True, color_normal=True, s=2, lw=1.5, size_vector=1.0, cmap=None, cbar_rect=None, alpha_event=0.7, fig=None, ax=None)` plots events and normal vectors
  - `mode` – Type of projection. Select one from "2d" (Cartesian map view), "3d" (bird's eye view), and "cartopy" (geographic map view). Default to "2d"
  - `includes_normal` – Whether to include normal vectors. Default to True
  - `color_normal` – Whether colors are based on normal vectors (True) or based on apparent strike (False). If set to False, normal vectors are converted to apparent strike. Default to True
  - `s` – Size of scatter plot
  - `lw` – Linewidth
  - `cmap` – `cmap` of colorbar. Specify `str` or `matplotlib.colors.Colormap`. If not specified, `DEFAULT_CYCLIC_CMAP` will be used
  - `cbar_rect` – Bbox for colorbar
  - `alpha_event` – Opacity of scatter plot. Default to 0.7
  - `fig` – `matplotlib.figure.Figure`. If not specified, new Figure object will be created.
  - `ax` – `matplotlib.axes.Axes`. If not specified, new Axes object in `fig` will be created

Returns `fig (matplotlib.figure.Figure)`

- `.plot_planes(key, mode="2d", max_angle_diff=30.0, min_points=5, size=1.0, kw_planes_mpl, cmap="jet", cone_scale=1.0, plotly_layout=None, fig=None, ax=None, **view_kwargs)` plots extracted fault planes
  - `key` – Index of the associated parameter set and result

- `mode` – Type of projection. Select one from "2d" (Cartesian map view), "3d" (bird's eye view), and "plotly" (bird's eye view on plotly). Default to "2d"
- `max_angle_diff` – Maximum median angle difference between fault normal and point-cloud normal vectors. Specify 90.0 to plot all the clusters. Default to 30.0
- `min_points` – Minimum number of points in an output cluster. Default to 5
- `size` – Size of scatter plot
- `kw_planes_mpl` – Dictionary in `mpl` plot used for plotting planes. Default to `{"c"="k", "lw"=1.0, "ls"="-", "alpha"=0.5}`
- `cmap` – `cmap` of colorbar. Specify `str` or `matplotlib.colors.Colormap`. If not specified, `DEFAULT_CYCLIC_CMAP` will be used
- `cone_scale` – Size of normal vectors
- `plotly_layout` – Optional layout for plotly plot
- `fig` – `matplotlib.figure.Figure` or `plotly.graph_objects.Figure (mode="plotly")`. If not specified, new `Figure` object will be created
- `ax` – `matplotlib.axes.Axes`. If not specified, new `Axes` object in `fig` will be created except when `mode="plotly"`

Returns `fig (matplotlib.figure.Figure or plotly.graph_objects.Figure)`

- `.plot_dendrogram(key, ax=None, recursionlimit=None, **kwargs)` plots a dendrogram for the specified parameter set
  - `key` – Index of the associated parameter set and result
  - `ax` – `matplotlib.axes.Axes`. If not specified, new `Axes` object will be created
  - `recursionlimit` Set the recursion limit by `sys.setrecursionlimit()`. Note that this works for the whole process

Returns `ax (matplotlib.axes.Axes)`

## 4.3 Functions

### 4.3.1 `fncv.utils.compute_each_normal_vector(locations, returns_EVR=False)`

Computes the normal vector (PC3 eigenvector) of the input point cloud. Normal vector is defined as upward direction, but we use the coordinate system of [North, East, Down] positive. So, Z component of the normal vector must be negative

- `locations` – ndarray for input 3D point cloud data (NED)
- `returns_EVR` – Whether to return variance ratios. Default to `False`

Returns ndarray of the upward normal vector

#### 4.3.2 `fncv.utils.compute_fault_params(points)`

Computes various fault params required for output files

- `points` – ndarray for input 3D point cloud data (NED)

Returns tuple of `box_arr`, `fault_length`, `fault_width`, `normal_vector`, `strike_vector`, `dip_vector`, `variance`, `planarity`, and `pc3_variance_ratio`

#### 4.3.3 `fncv.utils.init_transformer(eps_g_local)`

Initializes `pyproj.Transformer` from a local EPSG code. This will help convert geographic/-Cartesian coordinates into the other

- `eps_g_local` – str for EPSG code (e.g., "eps\_g:6675")

Returns two `pyproj.Transformer` objects: the former converts geographic to Cartesian and the latter converts Cartesian to geographic

#### 4.3.4 `fncv.utils.fix_aspectratio(fig, only_visible=True)`

Fixes the aspect ratio for plotly plot. Note that plotly must be importable on your environment.

- `fig` – `plotly.graph_objects.Figure`
- `only_visible` – Determines the axes scale only from visible data points when set to `True`. When set to `False`, determines it from all data points.

#### 4.3.5 `fncv.synthetic.synthesize_cluster(n_points, normal_vector_NED, planar_shape="uniform", scale=0.1, strike_lim=[-2,2], dip_lim=[-1,1], returns_as_END=True)`

Synthesize point cloud data around a given planar shape. The origin of data points is `[0,0,0]`

- `n_points` – The number of points to be synthesized

- `normal_vector_NED` – ndarray of the upward normal vector
- `planar_shape` – Shape of the plane on which data points are distributed. Choose one from "uniform" or "circular". Default to "uniform"
- `scale` – Standard deviation of the Gaussian noise added to the data points. Default to 0.1
- `strike_lim` – Range for the strike direction. Default to [-2,2]
- `dip_lim` – Range for the dip direction. Default to [-1,1]
- `returns_as_END` – Whether to return output in END. Default to True




Returns ndarray for output 3D point cloud data in END (if `returns_as_END=True`)

## Note

- This package has been tested on Ubuntu 22.04
- The authors do not assume any responsibility for any issues or consequences that may arise from the use of this package

## Contributors


### Core team

- Yasunori Sawaki  (Ph.D.) – Principle developer (`sawaki.yasunori@aist.go.jp`)
- Yoshihiro Sato  (Ph.D.) – Developed the two-step clustering method (`yosato@tcu.ac.jp`)
- Takahiko Uchide  (Ph.D.) – Project administrator (`t.uchide@aist.go.jp`)

### Additional contributors

- K. Sagae, A. Mpuang, T. Shiina, and H. Horikawa – Code review

## Terms of use

- Please cite this Open-File Report and our preprint (or a paper when published) for this method (Sawaki et al., 2025) if you use the package
- The latest version of this package, FaultNVC, can be accessible from the principal developer, Y. Sawaki , upon request
- Any bug reports and suggestions are welcome!

## References

- Campello, R. J. G. B., Moulavi, D., & Sander, J., 2013. *Density-Based Clustering Based on Hierarchical Density Estimates*, pp. 160–172, Springer, doi: 10.1007/978-3-642-37456-2\_14.
- Kamer, Y., Ouillon, G., & Sornette, D., 2020. Fault network reconstruction using agglomerative clustering: Applications to southern Californian seismicity, *Natural Hazards and Earth System Sciences*, **20**, 3611–3625, doi: 10.5194/nhess-20-3611-2020.
- McInnes, L., Healy, J., & Astels, S., 2017. hdbscan: Hierarchical density based clustering, *The Journal of Open Source Software*, **2**, 205, doi: 10.21105/joss.00205.
- Ouillon, G., Ducorbier, C., & Sornette, D., 2008. Automatic reconstruction of fault networks from seismicity catalogs: Three-dimensional optimal anisotropic dynamic clustering, *Journal of Geophysical Research*, **113**, B01306, doi: 10.1029/2007JB005032.
- Sato, Y., Horikawa, H., Uchide, T., Fukayama, S., & Ogata, J., 2022. Fault Plane Estimation from 3D Hypocenter Distribution by Two-step Clustering Considering Local Shape, in *The 2022 Seismological Society of Japan Fall Meeting*, pp. S21P–07.
- Sato, Y., Horikawa, H., Uchide, T., Fukayama, S., & Ogata, J., 2023. Estimation of fault planes by two-step clustering on hypocenter distribution, in *Japan Geoscience Union Meeting 2023*, pp. SCG55–P12.
- Sawaki, Y., Shiina, T., Sagae, K., Sato, Y., Horikawa, H., Miyakawa, A., Imanishi, K., & Uchide, T., 2025. Fault Geometries of the 2024 Mw 7.5 Noto Peninsula Earthquake from Hypocenter-Based Hierarchical Clustering of Point-Cloud Normal Vectors [Preprint], *Earth and Space Science Open Archive*.



Truttmann, S., Diehl, T., & Herwegh, M., 2023. Hypocenter-Based 3D Imaging of Active Faults: Method and Applications in the Southwestern Swiss Alps, *Journal of Geophysical Research: Solid Earth*, **128**, e2023JB026352, doi: 10.1029/2023JB026352.

## Acknowledgments

- Comments from Ryota Hino and Satoshi Ide improved the method.
- This study was supported by [MEXT Project for Seismology toward Research Innovation with Data of Earthquake \(STAR-E\)](#) Grant Number JPJ010217.

## License

This package is licensed under the BSD 3-Clause License (see LICENSE file for full license details). Copyright (c) 2024, National Institute of Advanced Industrial Science and Technology (AIST). All rights reserved.