Anthropogenic mw

Release 0.0.1

Jan Wiszniowski

Mar 04, 2025

CONTENTS

Moment magnitude estimation of small local earthquakes

Version

Release
0.0.1

The Anthropogenic Mw package is designed for the determination of the moment magnitude (Mw) of small and local earthquakes, where stations are close to hypocenters. Such situations often occur in anthropogenic events, and this algorithm was developed to calculate Mw of mining-induced and reservoir-triggered earthquakes. Hence, the package name is *Anthropogenic Mw*. However, it can also be used for natural events and is recommended for local ones.

The method of Mw computation based on spectral displacement amplitude is elaborated. Mw is computed using a fitting of displacement spectra of seismic waves recorded at stations to the simulated spectrum in the far field with the estimation of the noise. As proposed, it allows for estimating Mw based on a single P or S wave spectra. However, a combined spectrum of two waves together and spectrum simulation in intermediate and near fields was applied to Mw estimation as an innovation. The algorithm automatically estimates the station magnitude of small and close events.

Go to section *Theoretical Background* to get more information on how the code works.

Mw is written in Python and requires a working Python environment and same Python packages to run (see *Installation*).

The *Anthropogenic Mw* package primarily provides several functions and classes for estimating Mw, but it also provides a few command line tools:

- `spectral_Mw`: Compute moment magnitudes from all events in the QuakeMl catalog and add them to the catalog.
- `view_greens_function`: Direct modelling of P, S, and PS waves spectra in far, intermediate, and near fields based on user-defined earthquake source parameters, for various hypocentral distances and plot the results in the frequency domain.

CHAPTER

ONE

THEORETICAL BACKGROUND

1.1 Overview

Anthropogenic Mw inverts the P, S, or PS waves displacement spectral amplitude from station recordings of a single event.

The source parameters are estimated at a station on a recording of the single wave c (P, S) or both waves (PS) by minimisation of

$$\min_{M_0, f_0} L^{(c)} = \| \widehat{U}^{(c)} \left(f | M_0, f_0 \right), \widetilde{U}^{(c)} \left(f \right) \|$$

where $\widehat{U}^{(c))} \left(f | M_0, f_0 \right)$ is the simulated spectrum of the wave and $\widetilde{U}^{(c)} \left(f \right)$ is the displacement spectrum of the recorded wave.

1.2 Signal and noise spectrum

For each station, the code computes P, S, or PS waves displacement spectral amplitude for each component x (e.g., Z, N, E), within a predefined time window.

$$\widetilde{U}_x^{(c)} \left(f \right) = \frac{1}{2\pi f} \left| \int_{t_1^{(c)}}^{t_2^{(c)}} v_x(t) w(t) e^{-2j\pi f t} dt \right|$$

where the exponent (c) means that we are considering either P, S, or both P and S waves, $t_*^{(c)}$ are the start and end times of the P, S, or both P and S waves time window, $v_x(t)$ is the velocity time signal for component x, $w(t)$ is the tapper, and f is the frequency. The spectrum signal is scaled to the time signal according to Parseval's theorem

The same thing is done for a noise time window. However, it is done for a few separate time windows of noise (Fig. 1), which allows us to estimate the mean noise spectrum and the variation of the noise spectrum, which are used to modify the simulated spectrum and to weigh the spectral fitting.

The 3-D signal and noise spectra are the square root of the sum of component squares (e.g., Z, N, E) :

$$\widetilde{U}^{(c)} \left(f \right) = \sqrt{\left(\widetilde{U}_Z^{(c)} \left(f \right) \right)^2 + \left(\widetilde{U}_N^{(c)} \left(f \right) \right)^2 + \left(\widetilde{U}_E^{(c)} \left(f \right) \right)^2}.$$

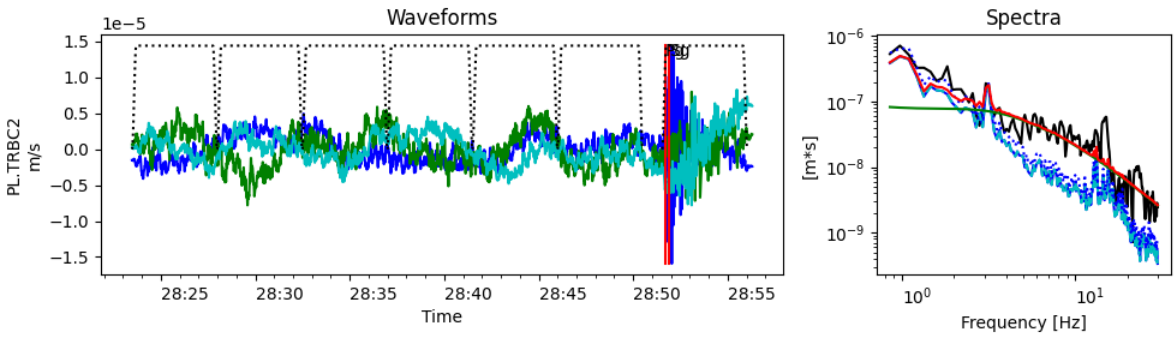

Fig. 1: Example three-component trace plot (displacement), showing noise and S+P wave windows. Pick pointed as read at traces. The colors of traces are: Z - blue, NS - cyan, and EW - green. Windows and tappers are presented as dotted lines. The colors of spectra are: signal - black, pure source spectra at the station - green, source spectra with noise correction at the station - red, noise - blue (dotted lines mark the noise uncertainty)

1.3 Source spectral model

Assuming no correlation of the signal and noise Pikoulis *et al.* [2020], the amplitude spectrum of the P or S, or both PS waves displacement spectra in far, intermediate, and near fields for fitting to the signal (Fig. 1) are modelled as

$$\hat{U}^{(c)}\left(f|M_0, f_0\right) = \sqrt{\left[S\left(f|M_0, f_0\right) G^{(c)}\left(f\right)\right]^2 + \left[\hat{N}\left(f\right)\right]^2},$$

where $S\left(f|M_0, f_0\right)$ is the source function, $G^{(c)}\left(f\right)$ is the response term, the product propagation terms in far, intermediate, and near fields, inelastic (internal) attenuation, and site effect, $\hat{N}\left(f\right)$ is the noise estimated at a station including the response of the recorder.

The Boatwright [1978], Boatwright [1980] source frequency function is used

$$S\left(f|M_0, f_0\right) = \frac{1}{2\pi f} M_{\,0} \left[1 + \left(\frac{f}{f_0}\right)^{n\gamma}\right]^{\frac{-1}{\gamma}}$$

where constant values γ and n controls the sharpness of the corners of the spectrum. Brune [1970], Brune [1971] model is a particular case of Boatwright's model for $n = 2$ and $\gamma = 1$. However, there are no contraindications to using other source models.

More detailed information about the response term is described in the *Main module*.

1.4 Inversion method

The parameters determined from the spectral inversion are moment magnitude M_w and scalar moment M_0 and corner frequency.

The inversion is performed by weighted spectra fitting. Two distances were investigated:

1. **The p-norm distance**

$$\|\mathbf{x}, \mathbf{y}\| = \left[\sum_{f=f_{low}}^{f_{high}} \left|x\left(f\right) - y\left(f\right)\right|^p \cdot w\left(f\right)\right]^{\frac{1}{p}}$$

2. **The logarithmic distance**

$$\|\mathbf{x}, \mathbf{y}\| = \left[\sum_{f=f_{low}}^{f_{high}} |\log(x(f)) - \log(y(f))|^p \cdot w(f) \right]^{\frac{1}{p}}$$

The weight coefficients are functions of frequency, signal spectrum, and noise spectrum estimators:

$$w^{(c)}(f) = w\left(f,\ \widetilde{U}^{(c)}(f), \hat{N}(f), \sigma_N(f)\right),$$

where $\sigma_N(f)$ is a standard deviation of the noise spectrum estimator.

Different inversion algorithms can be used, but a simple grid search algorithm is built into the program, which is sufficiently fast and accurate.

CHAPTER

TWO

GETTING STARTED

Note

For the impatient

To run the example, call it `spectral_Mw -c test.xml -s test.msd test.json`

Both Mw programs `spectral_Mw` and `view_green_fun` require the configuration file in the JSON format. The configuration file name is the only calling parameter that is always required. It is described in the *Configuration file* section. Seismic signal can be read from the MiniSEED file or downloaded from servers.

2.1 Spectral Mw calculation

The spectral Mw is calculated by `spectral_Mw`

2.1.1 Use case: external server

You must modify the *Configuration file* file and then run

```
spectral_Mw -q event.xml configuration.json,
```

where event.xml is an example of the catalog file name and configuration.json is an example of the configuration file name.

2.1.1.1 FDSNWS

If you have access to FDSNWS server configure at least:

```
"stream": {
    "source": "fdsnws",
    "host": host_name,
    "net": net_code
},
```

2.1.1.2 ArcLink

If you have access to ArcLink server configure at least:

```
"stream": {
    "source": "arclink",
    "host": host_name,
    "port": "18001",
    "user": "anonymous@",
    "net": net_code
},
```

2.1.2 Use case: files

1. Prepare the the *Configuration file* file
2. **Run `spectral_Mw -s traces.mseed -q event1.xml configuration.json`,**
   where seismic recordings are in miniSEED format (e.g., `traces.mseed`), metadata are in StationXML format (metadata file name is in the configration file, e.g., `station.xml`) and event information is in QuakeML format (e.g., `event.xml`).

2.1.3 Command line arguments

After successfully installing Anthropogenic Mw (see *Mw installation*), you can get help on the command line arguments used by each code by typing from your terminal:

```
spectral_Mw -h
```

2.2 Source spectra visualization

Source spectra are plotted by `view_green_function`. Call

```
view_green_fun configuration.json
```

CHAPTER

THREE

CONFIGURATION FILE

The configuration is kept in the Python dictionary, where keys are case-sensitive strings and values depend on parameters. They can be strings, float values, integer values, boolean values, sub-dictionaries, or lists. Parameters definition can be required or optional. The required definitions must be in the configuration, whereas optional need not. When definition of optional parameters is missing the default value is taken. The information whether parameter is required or optional is with the parameter description. In the case of optional parameter the default value is described. The configuration file (example name: `config.json`) is a file in JavaScript Object Notation (JSON) Here is the example file:

```
{
  "station_parameters" : {
    "any" : {
      "phase_parameters": {
        "P" : {
          "Q_0" : 800.0,
          "Q_theta" : 0.0,
          "Q_corner" : 0.0,
          "kappa" : 0.0,
          "distance_exponent" : 1.0,
          "low_frequency" : 7.50000000000000010e-001,
          "high_frequency" : 3.00000000000000000e+001,
          "noise_bias" : 0.00000000000000000e+000,
          "noise_std_bias" : 0.00000000000000000e+000,
          "noise_freq_bias" : 0.00000000000000000e+000
        },
        "S" : {
          "Q_0": 400.0,
          "Q_theta": 0.0,
          "Q_corner": 0.0,
          "kappa": 0.0,
          "distance_exponent": 1.0,
          "low_frequency": 7.50000000000000010e-001,
          "high_frequency": 3.00000000000000000e+001,
          "noise_bias": 0.00000000000000000e+000,
          "noise_std_bias": 0.00000000000000000e+000,
          "noise_freq_bias": 0.00000000000000000e+000
        }
      },
      "far_radial_average_radiation" : 0.52,
      "far_transversal_average_radiation": 0.63,
      "mw_correction" : 0.00000000000000000e+000,
```

(continues on next page)

(continued from previous page)

```
      "consider_intermediate_field": false,
      "consider_near_field": false
    }
  },
  "default_rho": 2700.0,
  "default_vp": 5200.0,
  "default_vs": 3000.0,
  "method": "separate_phases",
  "metric": "lin",
  "source_model": "Brune",
  "taper" : {
    "type" : "cosine_taper",
    "percentage" : 10
  },
  "stream": {
    "source": "arclink",
    "host": "tytan.igf.edu.pl",
    "port": "18001",
    "user": "anonymous@igf.edu.pl",
    "timeout": 300,
    "net": "VN",
    "cache" : "cache_Mw"
  },
  "optimization": {
    "method": "grid_search",
    "mw": [0.0, 5.01, 0.05],
    "log_f0": [-0.6, 1.51, 0.05]
  },
  "inventory": {
    "file_name": "VN_Stations.xml",
    "file_format": "STATIONXML"
  },
  "remove_response": {
    "prefilter": [0.01, 0.2, 45, 50]
  },
  "plot": {
    "draw_the_noise": false,
    "draw_source_spectrum_without_the_noise": true,
    "own_frequencies": [0.1, 30, 0.1],
    "view": "DISP",
    "how_to_show": "many figures inline"
  }
}
```

Below is the description of parameters. Not all described bellow parameters are in the example, not all parameters are required and some of them are optional. If optional parameters are not defined, default values are assumed.

station_parameters
: (dict) describe all parameters required to estimate the Mw at the station. The value is the list of stations, where keys are station names and vales are sub-dictionaries of parameters for the current station. The magic station's name is `any`. When the station configuration is missing or the specific station parameter is missing, it will be taken from the `any` station. (required)

velocity_model

(str) (optional, default value is "constant"). Now, only "constant" is available. Other velocity models have not been implemented yet.

default_rho
(float) Default value of density [kg/m^3] (required),

default_vp
(float) Default value of the P wave velocity [m/s] (required),

default_vs
(float) Default value of the S wave velocity [m/s] (required),

method
(str) (required) the moment magnitude spectral estimation method. Two options are available:

- "separate_phases" - estimation of Mw base on P wave and/or S wave, and the common magnitude is calculated based on $Mw^{(P)}$ and $Mw^{(S)}$
- "multiphase" - estimate Mw base signal covering both P and S wave.

metric
(str) The metric distances used for spectra fitting. Two metrics are available: p_norm (lin) and log (See *Inversion method*), (optional, default value is "p_norm"),

p_value
(float) The power of the values distances in the metric. (See *Inversion method*) (optional, default value is 2.0)

source_model
(str) (optional, default value is "Brune")

Boatwright gamma
(float) (optional, default value is 1.0)

Boatwright n
(float) (optional, default value is 2.0)

taper
(dict) the signal *Taper parameters* for Fourier transform (required)

stream
(dict) *Stream parameters* describing how to get streams for magnitude estimation (required)

optimization
(dict) *Optimization parameters* defining the minimization method of signal and source spectra difference (required)

inventory
(dict) (required) The dictionary of parameters defining how to get the inventory of all stations (see *Inventory parameters*)

remove_response
(dict) Parameters required for stream preprocessing (required). They are described in the *Remove response parameters* section.

plot
(dict) define the results *Plot parameters* (optional, if the position is not defined no plot is performed)

output_file
(str) The name of the output file in which the catalog with estimated magnitudes will be saved (optional, default value is "output.xml")

module
(str) (optional, default value is "MinimizeInGrid")

method
(str) (optional, default value is "minimize")

catalog_file
(str) The catalog file name (required unless the name is the call parameter).

3.1 Not predefined parameters

Not predefined parameters are created during computation and should not be predefined.

Plotter
(PlotMw) It keeps the plotter class object. Do not define it.

3.2 Station parameters

Station parameters consts of

phase_parameters:
(dict) It describes all parameters required to estimate the Mw at the phase. There could phases `P`, `S`, and `any`. The parameter is taken from the `any` phase if the parameter is missing in `P` or `S` phases sub-dictionaries. (see *Phase parameters*)

far_radial_average_radiation:
(float) The average radiation in the far field radial direction. It is the same as P wave average radiation in the far field (optional, default value is 0.52)

far_transversal_average_radiation:
(float) The average radiation in the far field transversal directions. It is the same as S wave average radiation in the far field (optional, default value is 0.63)

mw_correction:
(float) Empirical Mw correction at the particular station (optional, default value is 1.0). It consists of a correction value added to the estimated value before the saving. It was shown in [Wiejacz and Wiszniowski, 2006] the correction is required in some cases.

consider_intermediate_field:
(bool) If it is true, the source spectrum calculation considers the intermediate field (optional, default value is false)

intermediate_p_radial_average_radiation:
(float) The average P wave radiation in the intermediate field radial direction (optional, default value is 4.0 * far_radial_average_radiation)

intermediate_p_transversal_average_radiation:
(float) The average P wave radiation in the far field transversal directions (optional, default value is -2.0 * far_transversal_average_radiation)

intermediate_s_radial_average_radiation:
(float) The average S wave radiation in the intermediate field radial direction (optional, default value is -3.0 * far_radial_average_radiation)

intermediate_s_transversal_average_radiation:
(float) The average S wave radiation in the far field transversal directions (optional, default value is 3.0 * far_transversal_average_radiation)

consider_near_field:
(bool) If it is true, the source spectrum calculation for common P and S waves considers the near field (optional, default value is false)

near_radial_average_radiation:
(float) The average radiation in the far field transversal directions(optional, default value is 9.0 * far_radial_average_radiation)

near_transversal_average_radiation:
(float) The average S wave radiation in the far field transversal directions (optional, default value is -6.0 * far_transversal_average_radiation)

weight:
(float) The station weight for computation of the event magnitude part from the current station magnitude (optional, default value is 1.0)

3.2.1 Phase parameters

Phase parameters (`phase_parameters`) consts of

The parameters below define the internal (inelastic) dumping by the formula

$$A(f) = exp\left(\frac{-\pi T f}{Q(f)}\right),$$

where $Q(f)$ is is given by

$$Q(f) = Q_0\left(\frac{f_q + f}{f_q}\right)^{\vartheta},$$

or not considering cornel frequency

$$Q(f) = Q_0(f)^{\vartheta},$$

where:

Q_0
(float) define the Q_0 value (required)

Q_theta
(float) define the Q_θ value (optional, default value is 0.0)

Q_corner
(float) define the f_q value (optional, default value is 0.0, which means the second formula, not considering cornel frequency, is used)

The parameters below define the site near surface amplification and dumping according the formula

$$R(f) = R_c \exp(-\pi\kappa f)$$

where:

kappa
(float) define the dumping κ value (optional, default value is 0.0)

surface_correction
(float) define the free surface amplification R_c value (optional, default value is 1.0)

These parameters define the site amplification and dumping by the formula for the current phase when the signal is calculated in the far field. In the more fields. The signal correction is accordingly:

- P phase - the radial component of the signal
- S phase - the transversal component of the signal

distance_exponent
(float) 1.0, **Not used in current solution**

The frequency limits are defined for phases, but in the case of the PS-wave method the P wave values are considered.

low_frequency
(float) lowest frequency of spectra fitting (optional, default value is 0.5)

high_frequency
(float) highest frequency of spectra fitting (optional, default value is 20.0)

The parameters below modify (bias) the noise correction. We notice that slight noise bias protects occasional estimation of wrong high magnitude and low cornel frequency in the case of small signals and low-frequency noise The noise is biased by the formula:

$$\widehat{N}(f) = \overline{N}(f)\left(1 + b_N f^{b_f}\right) + b_\sigma \sigma_N$$

where:

noise_bias
(float) define the b_N value (optional, default value is 0.0)

noise_std_bias
(float) define the b_σ value (optional, default value is 0.0)

noise_freq_bias
(float) (optional, default value is 0.0) define the $b_f f$ value

weights
(dict) the definition of fitting weights assessment (optional, default value is None)

window
(dict) The signal window period for spectra computation (optional, if missing, the window is set based on S - P time). It contains two coefficients b_1 and b_2 defined by keys "b1" and "b2". The window time is calculated according to $\tau = r^{b_1}/10^{b_2}$

length
The minimal S-wave window (optional, default = 2 s)

P-S
The part of P-S time taken as P wave window (optional, default value is 0.9)

3.2.1.1 Phase weights parameters

use_threshold
(float) the binary weights threshold T_h

$$w = \begin{Bmatrix} 1 : \Delta \leq T_h \\ 0 : \Delta < T_h \end{Bmatrix}$$

If the parameter use_threshold is missing or None, weights take real values

$$w = \begin{Bmatrix} 0 & : & \Delta \leq 0 \\ \Delta & : & 0 < \Delta < 1 \\ 1 & : & \Delta \geq 1 \end{Bmatrix}$$

(optional, default is None)

use_logarithm
(bool) the Δ calculation method. If the "use_logarithm" parameter exists and is true

$$\Delta = \log(U) - \log(N)$$

else

$$\Delta = \frac{U - N}{U}$$

where U is the seismic signal and N is the noise term computed based on the mean value and standard deviation of noise (optional, default value is false)

use_std
(float) the coefficient of the part of the standard deviation of noise in the nose term computation. When the use_std is set to s value

$$N = \bar{N} + s\delta N$$

else

$$N = \bar{N}$$

where $\widehat{N}$ is the mean value of noise, δN is the standard deviation of noise, and s is the "use_std" parameter (optional, default is None)

use_frequency
(float) The option, whether use frequencies in as fitting weighting (optional, default is None). The parameter defines the value f_w in the formula

$$w = w\,|f - f_m|^{f_w}$$

use_main_frequency
The option, whether use main frequency in the middle of testing band for fitting weighting (float) (optional, default is 0). The parameter defines the value f_m

The phase P parameters are treated as both phase parameters in the case of multiphase spectral Mw estimation.

3.3 Taper parameters

type
(str) The taper type (required, only available type is now "cosine_taper")

percentage
(float) Percentage of cosine taper (optional, default value is 10)

half_cosine
(bool) If it is true, the taper is a half cosine function, otherwise, it is a quarter cosine function (optional, default value is true)

3.4 Stream parameters

source
(str) The web server source type (required, available options "arclink", "fdsnws")

host
(str) Host name (required)

port
(int) Server port number, (optional)

user
(int) User name, (required for arclink)

timeout
The waiting time for the server response (optional)

net
(str) The network code (required if *stations* parameter is missing)

loc
(str) The location filter (optional)

chan
(str) Channels filter (optional)

stations
(list(str)) list of station names. When stations names are in the form "NN.SSSS" where "NN" is the network code and "SSSS" is the station code. The "net" parameter can be omitted. If stations names are in the form "SSSS", the "net" parameter must be defined. It is possible to define in the list individual channels in the form "NN.SSSS.LL.CCC" where "LL" is a location code (can be empty) and "CCC" is the channel code.

cache
(str) the cache directory (optional, if missing data are not cached)

3.5 Optimization parameters

module
(str) The module name containing the optimization function (optional, default is "MinimizeInGrid"),

method
(str) The optimization function name (optional, default is "grid_search"),

mw
(list) This parameter is required for the grid search optimization. It is the list of three values describing the grid whose magnitude was checked. They are: initial value, upper limit, and the step e.g. [0.0, 5.01, 0.05],

log_f0
(list) This is the parameter required for the grid search optimization. It is the list of three values describing the grid in which logarithms were checked. They are: the initial value, upper limit, and the step, e.g. [-0.6, 1.51, 0.05]

3.6 Inventory parameters

The *Inventory parameters* describe how to read station inventories.

file_name
The file name of the inventory file (optional, default value is "inventory.xml"). When the file doesn't exist, the program tries to download the inventory to the file from the server defined in *Stream parameters*,

file_format
The inventory format (optional, default value is "STATIONXML"). It is not required when the inventory file exists

3.7 Remove response parameters

The remove response parameters are compatible with the ObsPy response removing :water_level: (int) (optional, default value is 128) :prefilter: (list(4*float)) The prefilter coefficients. :output: (str) (optional, default value is "VEL")

3.8 Plot parameters

do_not_draw:
(bool) When this parameter exists and is true, results are not plotted. This parameter allows turn off plotting without erasing plot configuration (optional, default value is false)

draw_the_signal_spectrum:
(bool) When this parameter exists and is false, the signal spectrum is not plotted. Usually, this parameter is missing and useless (optional, default value is true)

draw_the_noise:
(bool) When this parameter exists and is true, the noise spectrum is plotted (optional, default value is false)

draw_the_noise_uncertainty:
(bool) When this parameter exists and is true, The dotted linie describing the high noise spectrum uncertainty levels is plotted (optional, default value is false)

draw_the_noise_correction:
(bool) When this parameter exists and is true, the noise correction value, which needn't be the same as the noise, is plotted (optional, default value is false)

draw_source_spectrum_without_the_noise:
(bool) When this parameter exists and is true, the source spectra corrected by Green function but by noise is plotted in green (optional, default value is false)

draw_source_spectrum_with_the_noise:
(bool) When this parameter is missing or is true, the final source spectrum is plotted (optional, default value is true)

own_frequencies:
(list(float, float, int)) Used to define spectra frequency for viewing the source function simulation. consists of low frequency, high frequency, and the number of frequency points, e.g. [0.1, 30, 25] (required in the *test_greens_function*)

view:
(str) Defines the result plotting method (optional, allowed values "DISP" - displacement or "VEL" = velocity, default value is "VEL")

how_to_show:
(str) Defines the result plotting method. Three methods can be used

- "single figure" - all stations are plotted in one figure. Each row is one station,
- "many figures" - each station is plotted in separate figures - signals at the top

CHAPTER

FOUR

INPUT/OUTPUT DATA

General input data are * ObsPy catalogs * ObsPy streams * ObsPy inventory

All data can be read from files or downloaded from seismic servers. The downloaded depends on the server type, but preferred are MiniSEED for waveforms and StationXML for the inventory and QuakeML for the catalogue.

Two types of servers are defined * FDSN web server * ArcLink server The FDSN web server communication is supported by ObsPy library. The ArcLink server is obsolete in the ObsPy, therefore it required an additional library from the older ObsPy version. Communication with the server is performed in the cache mode (see Downloading data from servers)

4.1 Downloading data from servers

Downloading data from servers is performed in the cache mode (see Downloading data from servers). It means that the downloaded are stored in the disk cache, and next downloading takes data from the file in the cache. It allows faster downloading in repeated calculations or taking the data in the cache to the computer, which cannot access the server. The downloaded data are preprocessed before storing on the cache disk. It speedup rereading, but in the case of changing preprocessing, data must be reloaded. The cache consists of downloaded stream or inventory files and the file containing a description of what was downloaded.

Here is the example downloading file:


```
{
    "smi:igf.edu.pl/LGCD_CIBIS_800586_PL.KWLC": {
        "source": {
            "source": "arclink",
            "host": "tytan.igf.edu.pl",
            "port": "18001",
            "user": "anonymous@igf.edu.pl",
            "timeout": 300,
            "net": "PL",
            "cache": "cache_Mw"
        },
        "stations": [
            "PL.KWLC"
        ],
        "begin_time": "2015-08-19T01:06:10.915125",
        "end_time": "2015-08-19T01:06:17.349625",
        "file_name": "cache_Mw/96559972-cee2-4da9-a30e-d997dd2736b2.msd",
        "preprocess_name": "Mw_preprocessing_1",
        "processing": {
```


(continues on next page)

(continued from previous page)


```
            "PL.KWLC..EHZ": [
                "ObsPy 1.4.1: trim(endtime=UTCDateTime(2015, 8, 19, 1, 6, 17,␣
↪344734)::fill_value=None::nearest_sample=True::pad=False::starttime=UTCDateTime(2015,␣
↪8, 19, 1, 6, 10, 914734))",
                "ObsPy 1.4.1: remove_response(fig=None::inventory=<obspy.core.inventory.
↪inventory.Inventory object at 0x000001C9B8CEF2B0>::output='VEL'::plot=False::pre_
↪filt=[0.01, 0.05, 45, 50]::taper=True::taper_fraction=0.05::water_level=128::zero_
↪mean=True)"
            ],
            "PL.KWLC..EHN": [
                "ObsPy 1.4.1: trim(endtime=UTCDateTime(2015, 8, 19, 1, 6, 17,␣
↪344734)::fill_value=None::nearest_sample=True::pad=False::starttime=UTCDateTime(2015,␣
↪8, 19, 1, 6, 10, 914734))",
                "ObsPy 1.4.1: remove_response(fig=None::inventory=<obspy.core.inventory.
↪inventory.Inventory object at 0x000001C9B8CEF2B0>::output='VEL'::plot=False::pre_
↪filt=[0.01, 0.05, 45, 50]::taper=True::taper_fraction=0.05::water_level=128::zero_
↪mean=True)"
            ],
            "PL.KWLC..EHE": [
                "ObsPy 1.4.1: trim(endtime=UTCDateTime(2015, 8, 19, 1, 6, 17,␣
↪344734)::fill_value=None::nearest_sample=True::pad=False::starttime=UTCDateTime(2015,␣
↪8, 19, 1, 6, 10, 914734))",
                "ObsPy 1.4.1: remove_response(fig=None::inventory=<obspy.core.inventory.
↪inventory.Inventory object at 0x000001C9B8CEF2B0>::output='VEL'::plot=False::pre_
↪filt=[0.01, 0.05, 45, 50]::taper=True::taper_fraction=0.05::water_level=128::zero_
↪mean=True)"
            ]
        }
    },
    "smi:igf.edu.pl/LGCD_CIBIS_800586_PL.MOSK": {
        "source": {
            "source": "arclink",
            "host": "tytan.igf.edu.pl",
            "port": "18001",
            "user": "anonymous@igf.edu.pl",
            "timeout": 300,
            "net": "PL",
            "cache": "cache_Mw"
        },
        "stations": [
            "PL.MOSK"
        ],
        "begin_time": "2015-08-19T01:06:13.726250",
        "end_time": "2015-08-19T01:06:16.183250",
        "file_name": "cache_Mw/4ca4c2fd-9131-46b7-83a2-d4874d09102a.msd",
        "preprocess_name": "Mw_preprocessing_1",
        "processing": {
            "PL.MOSK..EHN": [
                "ObsPy 1.4.1: trim(endtime=UTCDateTime(2015, 8, 19, 1, 6, 16,␣
↪186531)::fill_value=None::nearest_sample=True::pad=False::starttime=UTCDateTime(2015,␣
↪8, 19, 1, 6, 13, 726531))",
                "ObsPy 1.4.1: remove_response(fig=None::inventory=<obspy.core.inventory.
```


(continues on next page)

(continued from previous page)

```
↪inventory.Inventory object at 0x000001C9B8CEF2B0>::output='VEL'::plot=False::pre_
↪filt=[0.01, 0.05, 45, 50]::taper=True::taper_fraction=0.05::water_level=128::zero_
↪mean=True)"
            ],
            "PL.MOSK..EHZ": [
                "ObsPy 1.4.1: trim(endtime=UTCDateTime(2015, 8, 19, 1, 6, 16,␣
↪186531)::fill_value=None::nearest_sample=True::pad=False::starttime=UTCDateTime(2015,␣
↪8, 19, 1, 6, 13, 726531))",
                "ObsPy 1.4.1: remove_response(fig=None::inventory=<obspy.core.inventory.
↪inventory.Inventory object at 0x000001C9B8CEF2B0>::output='VEL'::plot=False::pre_
↪filt=[0.01, 0.05, 45, 50]::taper=True::taper_fraction=0.05::water_level=128::zero_
↪mean=True)"
            ],
            "PL.MOSK..EHE": [
                "ObsPy 1.4.1: trim(endtime=UTCDateTime(2015, 8, 19, 1, 6, 16,␣
↪186531)::fill_value=None::nearest_sample=True::pad=False::starttime=UTCDateTime(2015,␣
↪8, 19, 1, 6, 13, 726531))",
                "ObsPy 1.4.1: remove_response(fig=None::inventory=<obspy.core.inventory.
↪inventory.Inventory object at 0x000001C9B8CEF2B0>::output='VEL'::plot=False::pre_
↪filt=[0.01, 0.05, 45, 50]::taper=True::taper_fraction=0.05::water_level=128::zero_
↪mean=True)"
            ]
        }
    }
}
```

The file in JSON format contains a dictionary, where the dictionary key is the downloading event ID and the dictionary value is the directory describing the downloading parameters:

source
: (str) The server parameters,

station
: (list(str)) The list of station names. It must be the list even if one station is defined. **WARNING! The list contains all requested stations, not only downloaded.**,

begin_time
: (UTCDateTime) The request begin time,

end_time
: (UTCDateTime) The request end time,

file_name
: (str) The name of the stream file,

preprocess_name
: (str) The name of the preprocessing method,

processing
: (dict(str,list(str))) The preprocessing description - copy of the trace class processing instance.

4.2 File formats

All the stream and inventory formats supported by ObsPy can be read from files or they are downloaded from seismic servers. The downloaded depends on the server type.

Preferred are MiniSEED for waveforms and StationXML for inventory

ObsPy catalog are read from the QuakeML files

4.3 Output file

The output file is the seismic catalogue supplemented with new moment magnitudes in the QuakeML format

CHAPTER

FIVE

MW INSTALLATION

Magnitude Mw requires at least Python 3.7. All the required dependencies will be downloaded and installed during the setup process.

5.1 Installing the latest release

To keep Anthropogenic Mw updated, run:

```
pip install --upgrade amw
```

from within your environment.

5.1.1 Using pip and PyPI

The latest release of Anthropogenic Mw is available on the Python Package Index.

You can install it easily through `pip`:

```
pip install amw
```

To upgrade from a previously installed version:

```
pip install --upgrade amw
```

5.2 Installing a developer snapshot

If you need a recent feature that is not in the latest release (see the "unreleased" section in *Changelogs*), you want to use the more recent development snapshot from the Anthropogenic Mw GitHub repository.

5.2.1 Using pip

The easiest way to install the most recent development snapshot is to download and install it through `pip`, using its builtin `git` client:

```
pip install git+https://github.com/JanWiszniowski/amw.git
```

Run this command again, from time to time, to keep Anthropogenic Mw updated with the development version.

5.2.2 Cloning the Anthropogenic Mw and Mw GitHub repository

If you want to take a look at the source code (and possibly modify it), clone the project using `git`:

```
git clone https://github.com/JanWiszniowski/amw.git
```

or using SSH:

```
git clone git@github.com:SeismicSource/amw.git
```

(avoid using the "Download ZIP" option from the green "Code" button, since version number is lost).

Then, go into the `amw` main directory and install the code in "editable mode" by running:

```
pip install -e .
```

You can keep your local Anthropogenic Mw repository updated by running `git pull` from time to time. Thanks to `pip`'s "editable mode", you don't need to reinstall Anthropogenic Mw after each update.

CHAPTER

SIX

GETTING HELP

6.1 I need help

Join the Anthropogenic Mw Discussions and feel free to ask!

6.2 I found a bug

Please open an Issue.

CHAPTER

SEVEN

MW CONTRIBUTING

Anthropogenic Mw estimation development happens on GitHub.

This is a new solution. So, I'm very open to contributions: if you have new ideas, please open an Issue. Don't hesitate to send me pull requests with new features and/or bugfixes!

CHAPTER

EIGHT

HOW TO CITE

No citation jet

CHAPTER

NINE

ANTHROPOGENIC MW API

Mw has a modular structure. Each module corresponds to a specific function or class of functions.

9.1 Main module

The main module, `spectral_mw.py`, contains functions that perform all the operations needed to count Mw magnitudes. These include finding peaks, taking waveforms, counting spectra, etc. Mw main module functions are presented below, following the alphabetic order. The module `spectral_mw.py` can be called directly with the configuration file in JsonFormat and the catalogue file in the QuakeML XML format as parameters

E.g: `python3 spectral_mw.py config.json catalog.xml`

9.1.1 spectral_mw

9.1.1.1 Spectral magnitude estimation for all event in the catalog

class amw.mw.spectral_mw.**MwStreamPreprocessing**(*configuration*, *inventory*)

amw.mw.spectral_mw.**catalog_moment_magnitudes**(*catalog*, *configuration*)

Function estimate the mw magnitude for all events having origin and picks according the configuration. As a result the function modifies the catalog and adds mw magnitude to events

Parameters

- **catalog** (*ObsPy Catalog*) – The seismic catalog of event the magnitude will be estimated. The estimated magnitudes will be added to the catalog magnitude.
- **configuration** (*dict*) – The configuration container of all parameters dictionary required for the program to work.

Returns

None

amw.mw.spectral_mw.**station_moment_magnitude**(*station_name*, *picks*, *event*, *origin*, *inventory*, *configuration*)

Function estimate the mw magnitude for one events having origin and picks according the configuration.

Parameters

- **station_name** – The station name in the form 'NN.SSSS', where NN is the network code and SSSS is the station_name code.
- **station_name** – str

- **picks** (*ObsPy Pick*) – The list of two picks: P and S. The P wave_name is first the S wave_name is second. If the wave_name is missing there should be None value. At list one wave_name is required, but wo picks P and S are recommended. At least one wave_name must be given. If the P or S wave_name is missing, the function tries to determine it based on the earthquake time at the focus and the remaining wave_name time.
- **event** (*ObsPy Event*) – The event object
- **origin** (*ObsPy Origin*) – The event origin.
- **inventory** (*ObsPy Inventory*) – The inventory Object. It must contain the station_name inventory
- **configuration** (*dict*) – The configuration container of all parameters dictionary required for the program to work.

Returns
The station magnitude object or None, if the magnitude can not be estimated.

Return type
ObsPy.StationMagnitude

9.1.2 test_greens_function

9.1.2.1 The plot of the source spectra

Program plot the source spectra including the source term and theoretical Green function

9.2 Support modules

9.2.1 The general spectral magnitude estimation

amw.mw.estimation.**estimate_mw**(*signal*, *begin_signal*, *picks*, *origin*, *station_inventory*, *configuration*)

Function estimate_mw estimates either single phase or cumulated phases spectral moment magnitude.

Parameters

- **signal** (*ObsPy Stream*) – The signal is the 3D seismic displacement stream, which must cover both the P wave, the S wave, and the noise before the P onset.
- **signal** – The signal is the 3D seismic displacement stream, which must cover both the P wave, the S wave, and the noise before the P onset.
- **begin_signal** (*ObsPy UTCDateTime*) – The first phase time (usually it is P wave) required to select noise before seismic waves onset
- **picks** (*list(ObsPy Pick)*) – A list of picks of waves is used for moment magnitude estimation. It can consist of a single pick P or S, then the magnitude estimation method from the single wave is used, or two picks P and S for the magnitude estimation based on both waves.
- **origin** (*ObsPy Origin*) – The event origin.
- **station_inventory** (*ObsPy Inventory*) – The inventory of the station that the signal was picked on
- **configuration** (*dict*) – The configuration container of all parameters dictionary required for the program to work.

Returns
mw : Estimated moment magnitude f0 : Source function corner frequency m0 : Scalar moment time_window : The assessed time window of P and S waves

Return type
tuple

Uses classes :
DefaultParameters MwFunctionParameters

Uses functions :
get_source_par : get_margin : get_station_name : get_phase_window get_spectrum get_noise_spectrum get_minimization_method minimization_method

9.2.2 Functions and classes for preparation of mw estimation procedure parameters

class amw.mw.parameters.**LogMetric**(*p*, *weights=None*)

The logarithmic metric class. It computes the distance

$$\|\mathbf{x}, \mathbf{y}\| = \left[\sum_{f=f_{low}}^{f_{high}} |\log(x(f)) - \log(y(f))|^p \cdot w(f) \right]^{\frac{1}{p}}$$

Parameters

- **p** – The power of the p-norm metric class
- **weights** (*np.array(float)*) – Weights for spectra frequency comparison. The size must be the same as spectra. Optional parameter. If missing no weight are applied.

class amw.mw.parameters.**MwFunctionParameters**(*picks*, *station_name*, *signal_spec*, *noise_spec*, *noise_sd*, *freq*, *source_parameters*, *station_inventory*, *configuration*)

The class keeps all parameters used to estimate the mw and its object is used as optimised function.

Parameters

- **picks** (*list(ObsPy.Pick)*) – List of two picks P and S. The P pick must be firsts .
- **station_name** (*str*) – The station name in the form required to find the configuration for the station
- **signal_spec** (*numpy.array(float)*) – The signal spectrum
- **noise_spec** (*numpy.array(float)*) – The noise mean spectrum
- **noise_sd** (*numpy.array(float)*) – The standard deviation of noise spectrum
- **freq** (*numpy.array(float)*) – The frequencies that the spectra are compared
- **source_parameters** (*SourceParameters*) – The seismic source parameters required for of mw estimation procedure
- **station_inventory** (*ObsPy Inventory*) – The inventory of the station that the signal was picked and mw is estimated
- **configuration** (*dict*) – The configuration container of all parameters dictionary required for the program to work.

Warning! signal_spectrum, noise_spectrum, noise_sd, frequencies must have the same size

class amw.mw.parameters.**PNormMetric**(*p: float*, *weights: array | None = None*)

The p-norm metric class. It computes the distance

$$\|\mathbf{x}, \mathbf{y}\| = \left[\sum_{f=f_{low}}^{f_{high}} |x(f) - y(f)|^p \cdot w(f) \right]^{\frac{1}{p}}$$

Parameters

- **p** – The power of the p-norm metric class
- **weights** (*np.array(float)*) – Weights for spectra frequency comparison. The size must be the same as spectra. Optional parameter. If missing no weight are applied.

amw.mw.parameters.**get_correction**(*phase_name*, *station_parameters*, *frequencies*, *travel_time*)

Calculate the internal dumping, near surface amplification and frequency dumping in frequency domain

Parameters

- **phase_name** (*str*) – The name of the phase ('P or 'S'). In the case of 'P' the internal dumping, etc. are calculating for radial component of the signal, which is the P wave in far field. In the case of 'S' the internal dumping, etc. is calculating for transversal component of the signal, which is the S wave in far field.
- **station_parameters** (*DefaultParameters*) – The station parameter
- **frequencies** (*numpy.array(float)*) – Frequencies for which the correction is calculated
- **travel_time** (*float*) – The signal travel time

Returns
The correction in frequency domain

Return type
numpy.array(float)

amw.mw.parameters.**get_far_response**(*travel_time*, *rho*, *r*, *fault_v*, *omega*)

The get_far_response function calculates the Green function in the far field $G^{(c)far}$ except the internal dumping and surface effect for phase P or S marked as (c) according to:

$$G^{(c)far} = \frac{j\omega \exp(-j\omega T_c)}{4\pi\rho r v_c^3},$$

where c is the wave name (P or S), ω is the circular frequency, $\omega = 2j\pi f$, r is the hypocentral distance, T_c is the phase travel time, v_c is the phase velocity at the source, ρ is the density at the source.

Parameters

- **travel_time** (*float*) – The phase travel time
- **rho** (*float*) – The density at the source [kg/m^3]
- **r** (*float*) – The hipocentral distance [m]
- **fault_v** – The phase velocity at the source [m/s]
- **fault_v** – float
- **omega** (*numpy.array(complex)*) – The circular frequencies, for which the response is counted. $\omega = 2j\pi f$

Returns
The far field part Green one phase function

Return type
numpy.array(complex)

amw.mw.parameters.**get_intermediate_response**(*travel_time*, *rho*, *r*, *fault_v*, *omega*)

The get_intermediate_response function calculates the radial or transversal component of Green function for phase P or S in the intermediate field $G_x^{(c)inter}$ except the internal dumping and surface effect for phase P or S marked as (c) according to:

$$G_x^{(c)inter} = \frac{exp\left(-j\omega T_c\right)}{4\pi\rho r^2 v_c^2},$$

where c is the wave name (P or S), ω is the circular frequency, $\omega = 2j\pi f$, r is the hipocentral distance, T_c is the phase travel time, v_c is the phase velocity at the source, ρ is the density at the source x describes the signal component (radial or transversal).

Parameters

- **travel_time** (*float*) – The phase travel time
- **rho** (*float*) – The density at the source [kg/m^3]
- **r** (*float*) – The hipocentral distance [m]
- **fault_v** (*float*) – The phase velocity at the source [m/s]
- **omega** (*numpy.array(complex)*) – The circular frequencies, for which the response is counted. $\omega = 2j\pi f$

Returns
The intermediate field part Green function of one phase

Return type
numpy array(complex)

amw.mw.parameters.**get_near_response**(*picks*, *source_parameters*, *station_inventory*, *omega*)

The get_near_response function calculates the common (radial and) transversal component of Green function in the intermediate field except the internal dumping and surface according to:

$$G^{near} = \frac{(\omega T_P + 1)\, exp\left(-\omega T_P\right) - (\omega T_S + 1)\, exp\left(-\omega T_S\right)}{4\omega^2\rho r^4}$$

where ω is the circular frequency, $\omega = 2j\pi f$, r is the hipocentral distance, T_P is the P phase travel time, T_S is the P phase travel time, ρ is the density at the source

Parameters

- **picks** (*list(ObsPy Pick)*) – A list of picks of waves in the near field. It must consist of two pick P or S and P must the first.
- **source_parameters** (*SourceParameters*) – The seismic source parameters required for of mw estimation procedure
- **station_inventory** (*ObsPy Inventory*) – The inventory of the station that the signal was picked and mw is estimated
- **omega** (*numpy.array(complex)*) – The circular frequencies, for which the response is counted. $\omega = 2j\pi f$

Returns
The near field part Green function

Return type
numpy.array(complex)

amw.mw.parameters.**get_phase_response**(*pick*, *source_parameters*, *station_inventory*, *station_parameters*, *frequencies*)

Function get_phase_response computes radial and transversal frequency responses of the seismic phase at the station to the source frequency function:

$$G^{(c)}(f) A^{(c)}(f) R(f) I(f),$$

where (c) is the phase name

The response consists of:

- Green function $G^{(c)}$, which for P wave is defined as

$$G_R^{(P)}(f) = G_R^{(P)far}(f) + G_R^{(P)inter}(f)$$
$$G_T^{(P)}(f) = G_T^{(P)inter}(f)$$

where $G_R^{(P)far} = G^{(P)far} R_R^{far}$ is the radial component of the P wave in the far field, $G_R^{(P)inter} = G^{(P)inter} R_R^{inter}$ is the radial component of the P wave in the intermediate field, and $G_T^{(P)inter} = G^{(P)inter} R_T^{inter}$ is the transversal component of the P wave in intermediate field. $R_R^{far} = R^{(P)}$ is the P wave average radiation coefficient in the far field. For S wave, it is defined as

$$G_R^{(S)}(f) = G_R^{(S)inter}(f)$$
$$G_T^{(S)}(f) = G_T^{(S)far}(f) + G_T^{(S)inter}(f)$$

where $G_R^{(S)inter} = G^{(S)inter} R_R^{inter}$ is the radial component of the S wave in intermediate field, $G_T^{(S)far} = G^{(S)far} R_T^{far}$ is the transversal component of the S wave in the far field, and $G_T^{(S)inter} = G^{(S)inter} R_T^{inter}$ is the transversal component of the S wave in the intermediate field. $R_T^{far} = R^{(S)}$ is the S wave average radiation coefficient in the far field.

Inelastic (internal) dumping $A^{(c)}$ is defined as

$$A(f) = exp\left(\frac{-\pi T_c f}{Q^{(c)}(f)}\right),$$

where

$$Q^{(c)}(f) = Q_0^{(c)} \left(\frac{f_q + f}{f_q}\right)^{\vartheta};$$

or

$$Q(f) = Q_0^{(c)} f^{\vartheta};$$

The near-surface losses and free surface amplification is assumed by

$$R(f) = R_c \exp(-\pi\kappa f).$$

The instrument response is $I(f)$.

Parameters

- **pick** (*str*) – The P or S pick name
- **source_parameters** (*SourceParameters*) – The seismic source parameters required for of mw estimation procedure

- **station_inventory** (*ObsPy Inventory*) – The inventory of the station that the signal was picked and mw is estimated
- **station_parameters** – The reference to the station_name (or default)
- **frequencies** (*numpy.array(float)*) – The frequencies, for which the response is counted

Returns
Tuple of two numpy arrays of complex radial and transversal response in frequency domain.

Return type
tuple(numpy.array(float), numpy.array(float))

amw.mw.parameters.**get_phases_response**(*picks*, *source_parameters*, *station_inventory*, *station_parameters*, *frequencies*)

Function get_phases_response computes radial and transversal frequency responses of the cumulated seismic phases P and S at the station to the source frequency function:

$$G_R^{(P+S)}(f) = G_R^{(P)far}(f) + G_R^{(P)inter}(f) + G_R^{(S)inter}(f) + G_R^{near}(f)$$
$$G_T^{(P+S)}(f) = G_T^{(P)inter}(f) + G_T^{(S)far}(f) + G_T^{(S)inter}(f) + G_T^{near}(f),$$

where $G_T^{near} = G^{near} R_T^{near}$ is the transversal component in the near field and $G_R^{near} = G^{near} R_R^{near}$ is the radial component in the near field. The remaining components are defined in the get_phase_response function.

Parameters

- **picks** – List of two P and S picks
- **source_parameters** (*SourceParameters*) – The seismic source parameters required for of mw estimation procedure
- **station_inventory** (*ObsPy Inventory*) – The inventory of the station that the signal was picked and mw is estimated
- **station_parameters** – The reference to the station_name (or default)
- **frequencies** – The frequencies, for which the response is counted

Returns
Tuple of two numpy arrays of complex radial and transversal response in frequency domain.

Return type
tuple(numpy.array(float), numpy.array(float))

amw.mw.parameters.**get_travel_time**(*pick*, *source_parameters*, *station_inventory*, *use_arrivals=False*)

Calculate the travel time of the picket wave from the source to the station, hipocentral distance to the station, and phase velocity at the fault

Parameters

- **use_arrivals** (*bool*) – Option whether to get travel time from origin arrivals. If it is false thr travel time is computed from source and station coordinates.
- **pick** (*ObsPy Pick*) – The pick of the wave that the travel time is assessed
- **source_parameters** (*SourceParameters*) – The seismic source parameters required for of mw estimation procedure
- **station_inventory** (*ObsPy Inventory*) – The inventory of the station that the signal was picked and mw is estimated

Returns
The phase travel time [s], hipocentral distance [m], phase velocity at the fault [m/s].

Return type
tuple(float, float, float)

9.2.3 The plot functions

class amw.mw.plot.**PlotMw**(*configuration*)

Class used to plot results of spectral magnitude estimation

Parameters
configuration (*dict*) – The full local mw configuration. The class uses the 'plot' subdirectory and 'method' describing the magnitude estimation method for preparing appropriate figure.

format_plot(*what_to_show*)

Does nothing. Prepared to reformat final view

Parameters
what_to_show (*str*) – Condition what to show. I can be only values 'single figure', 'many figures' or 'many figures inline'. Format is performed if the parameter equals with the 'what to show' 'plot' parameter in the configuration.

plot_results(*m0*, *f0*, *function_parameters*)

Plots spectra of estimation results. It can plot:

- The signal spectrum,
- The source spectrum at the station
- The mean value of the noise spectrum
- The uncertainty range of the noise spectrum

Parameters

- **m0** (*float*) – Scalar moment magnitude M_0
- **f0** (*float*) – Cornel focal function frequency f_0
- **function_parameters** (MwFunctionParameters) – All parameters used to estimate the mw

plot_seismogram(*picks*, *traces*, *stream*, *n_noises*)

Plots seismogram with picks and taper window

Parameters

- **picks**
- **traces**
- **stream**
- **n_noises**

set_plot(*station_name*, *pick_name=None*)

The method sets the plot area for station and optional the string. In the case of magnitude estimation on single phase the `pick` must be the phase name `P` or `S`. In the case of magnitude estimation on many phases together the `pick` must be None or omitted.

Parameters

- **station_name** (*str*) – The station name in the form "NN.SSSS" where NN is the network code and SSSS is the station code
- **pick_name** (*str*) – The pick name. It must be P or S for single phase mw or None for many phases mw

show_plot(*what_to_show*)

Visualize the plot and save figure to the file.

Parameters

what_to_show (*str*) – Condition what to show. I can be only values 'single figure', 'many figures' or 'many figures inline'. Plot is shown if the parameter equals with the 'what to show' 'plot' parameter in the configuration.

start_plot(*stations*)

The method configures the plots, defines figures and axis. The way of plotting depends on the `plot`, `how_to_show` parameter in the configuration file. There are three possibilities:

- `single figure` - results of magnitude estimation of all station are in plot one figure, station under station. On the left is seismogram and the left are spectra,
- `many figures` - results of magnitude estimation of all station are in one figure. On the left is the seismogram and the bottom are spectra,
- `many figures inline` - results of magnitude estimation of all station are in one figure. On the left is the seismogram and the left are spectra,

Parameters

stations (*list(str)*) – List of station names

9.2.4 Single phase spectral magnitude estimation

amw.mw.single_phase_mw.**estimate_single_phase_mw**(*signal*, *pick_name*, *picks*, *origin*, *station_inventory*, *configuration*)

Estimates spectral moment magnitude on the single phase P or S

Parameters

- **signal** (*ObsPy.Stream*) – The signal is the 3D seismic displacement stream, which must cover both the P wave, the S wave, and the noise before the P onset.
- **pick_name** (*str*) – The name of the pick 'P' or 'S'
- **picks** (*list(ObsPy.Pick)*) – Two picks P and S are recommended. At least one wave_name must be given. If the P or S wave_name is missing, the function tries to determine it based on the earthquake time at the focus and the remaining wave_name time.
- **origin** (*ObsPy.Origin*) – The event origin.
- **station_inventory** (*ObsPy.Inventory*) – The inventory of the station that the signal was picked on
- **configuration** (*dict*) – The configuration container of all parameters dictionary required for the program to work.
- **inventory** (*ObsPy.Inventory*) – The inventory of all stations and channels

Returns

mw : Estimated moment magnitude f0 : Source function corner frequency m0 : Scalar moment time_window : The assessed time window of P and S waves

Return type
tuple

9.2.5 Cumulated P and S phases spectral magnitude estimation

amw.mw.double_phase_mw.**estimate_double_phase_mw**(*signal*, *picks*, *origin*, *station_inventory*, *configuration*)

Estimates the moment magnitude on the signal covering both phases P and S together.

Parameters

- **signal** (*ObsPy Stream*) – The signal is the 3D seismic displacement stream, which must cover both the P wave, the S wave, and the noise before the P onset.
- **picks** (*list(ObsPy.Pick)*) – The list of two picks: P and S. The P wave_name is first the S wave_name is second. If the wave_name is missing there should be None value. At list one wave_name is required, but two picks P and S are recommended. At least one wave_name must be given. If the P or S wave_name is missing, the function tries to determine it based on the earthquake time at the focus and the remaining wave_name time.
- **origin** (*ObsPy Origin*) – The event origin.
- **station_inventory** (*ObsPy.Inventory*) – The inventory of the station that the signal was picked on
- **configuration** (*dict*) – The configuration container of all parameters dictionary required for the program to work.
- **inventory** (*ObsPy.Inventory*) – The inventory of all stations and channels

Returns
mw : Estimated moment magnitude f0 : Source function corner frequency m0 : Scalar moment time_window : The assessed time window of P and S waves

Uses functions :
get_theoretical_s get_theoretical_p estimate_mw

9.2.6 Seismic source models in frequency domain

class amw.mw.source_models.**BoatwrightSourceModel**(*frequencies*, *gamma=1.0*, *n=2.0*)

The Boatwright (1978; 1980) seismic source model is:

$$S\left(f|M_0, f_0\right) = \frac{1}{2\pi f} M_0 \left[1 + \left(\frac{f}{f_0}\right)^{n\gamma}\right]^{\frac{-1}{\gamma}},$$

where M_0 is a scalar moment and f_0 is a cornel frequency. Constant values γ and n controls the sharpness of the corners of the spectrum. For $\gamma = 1$ and $n = 2$, it is Brune (1970; 1971) source model:

$$S\left(f|M_0, f_0\right) = \frac{1}{2\pi f} M_0 \left[1 + \left(\frac{f}{f_0}\right)^{2}\right]^{-1}$$

Parameters

- **frequencies** (*numpy array*) – The frequencies values, for w spectral function values will be computed
- **gamma**

- **n**

Default parameters gamma=1 and n=2 are for Brunea source model

References:

- Boatwright, J. (1978). Detailed spectral analysis of two small New York State earthquakes, Bull. Seism. Soc. Am. 68 (4), 1117–1131. https://doi.org/10.1785/BSSA0680041117
- Boatwright, J. (1980). A spectral theory for circular seismic sources; simple estimates of source dimension, dynamic stress drop, and radiated seismic energy, Bull. Seism. Soc. Am. 70 (1), 1–27. https://doi.org/10.1785/BSSA0700010001
- Brune, J. N. (1970). Tectonic stress and the spectra of seismic shear waves from earthquakes, J. Geophys. Res. 75, 4997-5009. https://doi.org/10.1029/JB075i026p04997
- Brune, J.N. (1971) Correction [to "Tectonic Stress and the Spectra of Seismic Shear Waves from Earthquakes"], J. Geophys. Res. 76, 5002. http://dx.doi.org/10.1029/JB076i020p05002

class amw.mw.source_models.**BruneSourceModel**(*frequencies*)

Brune (1970; 1971) seismic source model is:

$$S\left(f|M_0, f_0\right) = \frac{1}{2\pi f} M_0 \left[1 + \left(\frac{f}{f_0}\right)^2\right]^{-1},$$

where M_0 is a scalar moment and f_0 is a corner frequency.

Parameters
frequencies (*numpy array*) – The frequencies values, for w spectral function values will be computed

class amw.mw.source_models.**HaskellSourceModel**(*frequencies*, *gamma=1.0*, *n=2.0*)

Haskell (1964) seismic source model is:

$$S\left(f|M_0, f_0\right) = \frac{1}{2\pi f} M_0 \text{sinc} \frac{f}{f0},$$

where M_0 is a scalar moment and f_0 is a corner frequency.

Parameters
frequencies (*numpy array*) – The frequencies values, for w spectral function values will be computed

9.2.7 The function minimization by checking all solutions in a grid

amw.mw.MinimizeInGrid.**grid_search**(*function*, *x*, *args=None*)

It minimizes the function by checking all solutions in a grid.

Parameters

- **function** (*func*) – The minimized function
- **x** (*list*) – Initial values (not used, only for compatibility)
- **args** (*list*) – The minimized function arguments. It must be the configuration dictionary.

Returns
The minimization result

Return type
RetClass

9.3 Core modules

Core modules are used in the Anthropogenic Mw package, but are designated for more general use, therefor the are described separately.

The arclink_client.py is located the core utils. Because it is now obsolete but still used in some seismic networks, it is taken from the older version of ObsPy and put in the core subdirectory.

9.3.1 The waveform and inventory manipulation

class amw.core.signal_utils.**Cache**(*configuration*, *file_name*)

The cache class for manipulating the cache metadata

Parameters

- **configuration** (*dict*) – The container of general seismic processing configuration. The required parameter is a cache path kept in the 'cache'.
- **file_name** (*str*) – The cache metadata file name

backup()

Backs up the cache metadata. Saves to the JSON file.

Returns
None

exception amw.core.signal_utils.**SignalException**(*message='other'*)

class amw.core.signal_utils.**StreamLoader**(*configuration*, *preprocess=None*)

The stream loader loads seismic waveforms from servers ArcLink or FDSNWS and process data initially. The loaded and processed data can be kept on local disc in the cache directory for increase the reloading speed.

Parameters

- **configuration** (*dict*) – The container of general seismic processing configuration. The required parameters are kept in the 'stream' sub-dictionary:
- **preprocess** (StreamPreprocessing)

The parameters present in the 'stream' sub-dictionary:

Source
The waveforms source. Available options are 'arclink' or 'fdsnws' (required)

Host
The server host

Port
The server port

User
The request user id (if required)

Password
The request password (if required)

Timeout
The downloading timeout limit

Net
The default network name.

Sta
The default station name.

Loc
The default location name.

Chan
The default channel name.

Cache
The cache directory. In the cache directory are kept all downloaded and preprocessed waveform files and the file 'loaded_signals.json' containing info

Stations
The default request station list

`download`(*begin_time*, *end_time*, *event_id*, *new_file_name=None*)

Downloads the stream from the seismic data sever with optional caching.

Parameters

- **new_file_name** (*str*) – The proposed name of the file tobe stored in the cache. If it is missing the unique random name is generated.
- **begin_time** (*ObsPy.UTCDateTime*) – The begin time of waveforms
- **end_time** (*ObsPy.UTCDateTime*) – The end time of waveforms
- **event_id** (*str*) – The event id, but it can be any string defining the stream request, which can identify the data in case of repeated inquiry.

Returns
The requested stream or None if it can not be downloaded

Return type
ObsPy.Stream

`exist_file`(*begin_time*, *end_time*, *event_id*)

The method checks if the requested waveform exists. A few conditions are checked. First it checks if the cache exists. Then checks if event id exists. The requested period must include in the existing file period. The requested station list must include in the existing file station list. The preprocessor name must be the same.

Parameters

- **begin_time** (*ObsPy.UTCDateTime*) – The requested waveforms begin time
- **end_time** (*ObsPy.UTCDateTime*) – The requested waveforms begin time
- **event_id** (*str*) – The request event id. It can be the event id that the waveforms are associated or any string that identify the request.

Returns
The parameters of existed file or None, if the function can not fit request to existing files list

Return type
dict

`get_signal`(*begin_time*, *end_time*, *event_id=None*, *stations=None*, *new_file_name=None*)

Provides seismic signal waveform based on request. If matching the request file exist in the cache it reads signal from the file, otherwise download from the seismic waveforms' server.

Parameters

- **begin_time** (*ObsPy.UTCDateTime*) – The requested waveforms begin time
- **end_time** (*ObsPy.UTCDateTime*) – The requested waveforms begin time
- **event_id** (*str*) – The request event id. It can be the event id that the waveforms are associated or any string that identify the request. (optional If missing waveform is only downloaded from the server)
- **stations** (*list(str)*) – The request stations list. (optional) If it is missing the station list from the configuration is checked.
- **new_file_name** (*str*) – The name of a file in the cache. (optional) If missing the unique file name is generated.

Returns
The waveform stream. Return None if it can not (or could not) download waveforms.

Return type
ObsPy.Stream

class amw.core.signal_utils.**StreamPreprocessing**(*name*)

The base class of streams preprocessing

Parameters
name (*str*) – The name of the preprocessing

amw.core.signal_utils.**get_inventory**(*sta_name*, *date*, *inventory*)

Extracts inventory for the station.

Parameters

- **sta_name** (*str*) – The station name as the string in the form 'NN.SSS', where 'NN' is the network code and 'SSS' is the station code.
- **date** (*ObsPy.UTCDateTime*) – The date of the inventory
- **inventory** (*ObsPy.Inventory*) – The inventory of all stations

Returns
The inventory of the station

Return type
ObsPy.Inventory

amw.core.signal_utils.**load_inventory**(*configuration*)

Loads inventory from the file. The file name and format are in 'inventory' configuration. If inventory file is missing the inventory is downloaded from the waveform server, which configuration is in the 'stream' sub-dictionary.

Parameters
configuration (*dict*) – The container of general seismic processing configuration. The required parameters are kept in the 'inventory' sub-dictionary.

Returns
The inventory

Return type
ObsPy.Inventory

The parameters present in the 'inventory' sub-dictionary:

File_name
The inventory file name. (optional, default is 'inventory.xml')

File_format
The format of the inventory file name. (optional, default is 'STATIONXML')

9.3.2 Commonly used utils for seismic data processing be the seismic processing in Python packages

class amw.core.utils.**ExtremeTraceValues**(*trace*, *begin_time=None*, *end_time=None*)

Class that assess the extreme trace values: maximum, minimum, and absolute maximum value

Parameters

- **trace** (*ObsPy.Trace*) – The processed trace
- **begin_time** (*ObsPy.UTCDateTime*) – It limits the period, where a process is performed. If begin_time is not defined or it is earlier than the beginning of the trace the process is performed from the beginning of the trace
- **end_time** (*ObsPy.UTCDateTime*) – It limits the period, where a process is performed. If end_time is not defined or it is later than the end of the trace, the process is performed to the end of the trace

Class variables

Data
The optionally cut to time limits data. The data are not a new array but subarray of the Trace data

Start_time
The time of the first data sample.

End_time
The time of the next sample after the last data sample. It differs from the ObsPy trace end_time, which points to the last sample of the trace

Max_value
Maximum data value

Max_value
Minimum data value

Abs_max
Absolute maximum value. abs_max = max(abs(min_value), abs(max_value))

class amw.core.utils.**IndexTrace**(*trace*, *begin_time=None*, *end_time=None*)

Class for operating directly on time limited part of trace data

Parameters

- **trace** (*ObsPy.Trace*) – The processed trace
- **begin_time** (*ObsPy.UTCDateTime*) – It limits the period, where a process is performed. If begin_time is not defined or it is earlier than the beginning of the trace the process is performed from the beginning of the trace
- **end_time** (*ObsPy.UTCDateTime*) – It limits the period, where a process is performed. If end_time is not defined or it is later than the end of the trace, the process is performed to the end of the trace

Class variables

Start_time
The time of the first data sample index

End_time
The time of the next sample after the last data sample. It differs from the ObsPy trace end_time, which points to the last sample of the trace

Begin_idx
The first data sample index

End_idx
The last data sample index + 1

Example:

```
>> from utils import IndexTrace
>> from obspy.core.utcdatetime import UTCDateTime
>> t1 = UTCDateTime(2024, 1, 3, 8, 28, 00)
>> t2 = UTCDateTime(2024, 1, 3, 8, 29, 00)
>> st = read('test.msd')
>> indexes = IndexTrace(st[1], begin_time=t1, end_time=t2
>> for idx in range(indexes.begin_idx, indexes.end_idx):
... pass
```

class amw.core.utils.**ProcessTrace**(*trace*, *begin_time=None*, *end_time=None*)

The base class of the trace processing. Implementations of objects of classes derived from the ProcessTrace do some processing on traces defined in the derived classes initialization

Parameters

- **trace** (*ObsPy.Trace*) – The processed trace
- **begin_time** (*ObsPy.UTCDateTime*) – It limits the period, where a process is performed. If begin_time is not defined or it is earlier than the beginning of the trace, the process is performed from the beginning of the trace
- **end_time** (*ObsPy.UTCDateTime*) – It limits the period, where a process is performed. If end_time is not defined or it is later than the end of the trace, the process is performed to the end of the trace

amw.core.utils.**get_focal_mechanism**(*event*, *inversion_type=None*)

Function get_focal_mechanism extracts the focal mechanism from the event. If preferred_focal_mechanism_id of the event is set it return the preferred focal mechanism. Otherwise, it returns the first focal mechanism from the list. The function is intended to extract the focal mechanism unconditionally and non-interactively. Therefore, if preferred_focal_mechanism_id is not set and there are multiple focal mechanisms, the returned focal mechanism may be random.

Parameters

- **event** (*ObsPy.Event*) – The seismic event object
- **inversion_type** (*(str)*) – The name of tensor inversion type. It must belong to the QuakeML MTInversionType category: `'general'`, `'zero trace'`, `'double couple'`, or None.

Returns
The focal mechanism object or None if none focal_mechanism is defined for the event or the focal_mechanism with the defined inversion type does not exist.

Return type
ObsPy.FocalMechanism

amw.core.utils.**get_hypocentral_distance**(*origin*, *station_inventory*)

Function get_hypocentral_distance computes the local hypocentral distance in meters from origin coordinates to station_name coordinates. The calculations do not take into account the curvature of the earth.

Parameters

- **origin** (*ObsPy.Origin*) – The ObsPy Origin object
- **station_inventory** (*ObsPy.Inventory*) – The station inventory object

Returns
The hypocentral distance in meters and epicentral distance in degrees

Return type
tuple(float, float)

amw.core.utils.**get_magnitude**(*event*, *magnitude_type=None*)

Function get_magnitude extracts the magnitude of the event. If you want to extract a specific magnitude you can define it as magnitude_type, e.g. `get_magnitude(event, magnitude_type='mw')`, otherwise, any magnitude will be extracted. If the preferred_magnitude_id of the event is set it returns the preferred origin. Otherwise, it returns the first magnitude from the list. The function is intended to extract the magnitude unconditionally and non-interactively. Therefore, if preferred_magnitude_id is not set and there are multiple magnitudes, the returned origin may be random.

If event magnitude does not exist, but station_name magnitudes exist, the new magnitude is computed as the mean value of station_name magnitudes.

Parameters

- **event** (*ObsPy.Event*) – The seismic event object
- **magnitude_type** (*str*) – (optional) Describes the type of magnitude. This is a free-text. Proposed values are: * unspecified magnitude (`'M'`) - function search for exactly unspecified magnitude, * local magnitude (`'ML'`), * moment magnitude (`'mw'`), * energy (`'Energy'`), * etc.

Returns
The magnitude object or None if the function cannot find or create the magnitude. If only station_name magnitudes exist, the new ObsPy Magnitude object is created, but it is not appended to the event

Return type
ObsPy.Magnitude

amw.core.utils.**get_net_sta**(*name*)

Function get_net_sta extracts network and station_name codes as strings

Parameters
name (*str or ObsPy.WaveformStreamID*) – The trace name. It can be the string or the WaveformStreamID object. The text in the string is in the form 'NN.SSS.LL.CCC', where NN is the network code, SSS is the station_name code, LL is the location code, and CCC is the channel code.

Returns
The tuple of the network code the station_name code.

Return type
tuple(str, str)

amw.core.utils.**get_origin**(*event*)

Function get_origin extracts the origin from the event. If preferred_origin_id of the event is set it return the preferred origin. Otherwise, it returns the first origin from the list. The function is intended to extract the event origin unconditionally and non-interactively. Therefore, if preferred_origin_id is not set and there are multiple origins, the returned origin may be random

Parameters
event (*ObsPy.Event*) – The seismic event object

Returns
The origin (event location) object or None if none origin is defined for the event.

Return type
ObsPy.Origin

amw.core.utils.**get_station_id**(*name*)

Function get_station_id extracts the station_name name as a WaveformStreamID object

Parameters
name (*str or ObsPy.WaveformStreamID*) – The trace name. It can be the string or the ObsPy WaveformStreamID object. The text in the string is in the form 'NN.SSS.LL.CCC', where NN is the network code, SSS is the station_name code, LL is the location code, and CCC is the channel code.

Returns
The waveform stream object containing only the network code and the station_name code.

Return type
ObsPy.WaveformStreamID

amw.core.utils.**get_station_name**(*name*)

Function get_station_name extracts the station_name name as a string

Parameters
name (*str or ObsPy.WaveformStreamID*) – The trace name. It can be the string or the WaveformStreamID object. The text in the string is in the form 'NN.SSS.LL.CCC', where NN is the network code, SSS is the station_name code, LL is the location code, and CCC is the channel code.

Returns
The string in the form 'NN.STA', where NN is the network code and SSS is the station_name code.

Return type
str

amw.core.utils.**get_units**(*trace*)

Return the signal units of the trace

Parameters
trace (*ObsPy.Trace*) – The trace object

Returns
The string with units: 'm/s', 'm/s^2', or 'm', if the response was removed, when in the processing_parameters is the remove_response process defined, or 'counts' otherwise

Return type
str

amw.core.utils.**time_ceil**(*time*, *step*)

Returns the time rounded-up to the specified accuracy.

Parameters

- **time** (*ObsPy.UTCDateTime*) – The time object
- **step** (*float*) – The accuracy units in seconds

Returns
The new rounded-up time object

Return type
ObsPy.UTCDateTime

Example:

```
>> from obspy.core.utcdatetime import UTCDateTime
>> from core.utils import time_ceil
>> time = UTCDateTime(2024, 1, 3, 8, 28, 33, 245678)
>> time_ceil(time,1.0)
>> UTCDateTime(2024, 1, 3, 8, 28, 34)
>> time_ceil(time,60.0)
>> UTCDateTime(2024, 1, 3, 8, 29)
>> time_ceil(time,0.1)
UTCDateTime(2024, 1, 3, 8, 28, 33, 300000)
>> time_ceil(time,0.01)
>> UTCDateTime(2024, 1, 3, 8, 28, 33, 250000)
>> time_ceil(time,0.001)
>> UTCDateTime(2024, 1, 3, 8, 28, 33, 246000)
```

amw.core.utils.**time_ceil_dist**(*time*, *step*)

Returns seconds from the time to the time rounded up to the specified accuracy.

Parameters

- **time** (*ObsPy.UTCDateTime*) – The time object
- **step** (*float*) – The accuracy units in seconds

Returns
The period in seconds to the rounded-up time

Return type
float

Example:

```
>> from obspy.core.utcdatetime import UTCDateTime
>> time = UTCDateTime(2024, 1, 3, 8, 28, 33, 245678)
>> time_ceil_dist(time,0.1)
0.054322
>> time_ceil_dist(time,1.0)
0.754322
```

amw.core.utils.**time_floor**(*time*, *step*)

Returns the time rounded-down to the specified accuracy.

Parameters

- **time** (*ObsPy.UTCDateTime*) – The time object
- **step** (*float*) – The accuracy units in seconds

Returns
The new rounded-down time object

Return type
ObsPy.UTCDateTime

Example:

```
>> from obspy.core.utcdatetime import UTCDateTime
>> from utils import time_floor
>> time = UTCDateTime(2024, 1, 3, 8, 28, 33, 245678)
>> time_floor(time,0.001)
UTCDateTime(2024, 1, 3, 8, 28, 33, 245000)
>> time_floor(time,0.01)
UTCDateTime(2024, 1, 3, 8, 28, 33, 240000)
>> time_floor(time,0.1)
UTCDateTime(2024, 1, 3, 8, 28, 33, 200000)
>> time_floor(time,1.0)
UTCDateTime(2024, 1, 3, 8, 28, 33)
>> time_floor(time,60.0)
UTCDateTime(2024, 1, 3, 8, 28)
```

amw.core.utils.**time_floor_dist**(*time*, *step*)

Returns seconds from the time to the time rounded up to the specified accuracy.

Parameters

- **time** (*ObsPy.UTCDateTime*) – The time object
- **step** (*float*) – The accuracy units in seconds

Returns
The period in seconds to the rounded-down time

Return type
float

Example:

```
>> from obspy.core.utcdatetime import UTCDateTime
>> time = UTCDateTime(2024, 1, 3, 8, 28, 33, 245678)
>> time_floor_dist(time,0.1)
0.045678
>> time_floor_dist(time,1.0)
0.245678
```

CHAPTER

TEN

CHANGELOGS

10.1 amw Changelog

Earthquake source parameters from P, S, or PS waves displacement spectra in far, intermediate, and near fields

10.1.1 v0.0.1 - 2024-12-06

Initial Python version.

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] Erion-Vasilis Pikoulis, Olga-Joan Ktenidou, Emmanouil Z. Psarakis, and Norman A. Abrahamson. Stochastic modeling as a method of arriving at higher frequencies: an application to estimation. *Journal of Geophysical Research: Solid Earth*, 125(4):e2019JB018768, 2020. doi:https://doi.org/10.1029/2019JB018768.

[2] John Boatwright. Detailed spectral analysis of two small new york state earthquakes. *Bulletin of the Seismological Society of America*, 68(4):1117–1131, 1978. doi:10.1785/BSSA0680041117.

[3] John Boatwright. A spectral theory for circular seismic sources, simple estimates of source dimension, dynamic stress drop, and radiated seismic energy. *Bulletin of the Seismological Society of America*, 70(1):1–27, 1980. doi:10.1785/BSSA0700010001.

[4] James N. Brune. Tectonic stress and the spectra of seismic shear waves from earthquakes. *Journal of Geophysical Research (1896-1977)*, 75(26):4997–5009, 1970. doi:10.1029/JB075i026p04997.

[5] James N. Brune. Correction [to "Tectonic Stress and the Spectra of Seismic Shear Waves from Earthquakes"]. *Journal of Geophysical Research*, 76(5):5002, 1971. doi:10.1029/JB076i020p05002.

[6] Paweł Wiejacz and Jan Wiszniowski. Moment magnitude determination of local seismic events recorded at selected polish seismic stations. *Acta Geophysica*, 54(1):15–32, Mar 2006. URL: https://doi.org/10.2478/s11600-006-0003-1, doi:10.2478/s11600-006-0003-1.

PYTHON MODULE INDEX

a