# SSSPy

*Release 0.0.1*

**Jan Wiszniowski**

**Feb 15, 2025**

# CONTENTS

**Simple seismic signal simulation from the simple source moment time function**

**Copyright**
2025 Jan Wiszniowski jwisz@igf.edu.pl

Contents:

CHAPTER

ONE

# METHODOLOGY

The goal of this programme was the assessment of the near and intermediate field influence on the seismic signal parameters, like maximum displacement amplitude of specific phases. The assessment is based on a very simple seismic signal simulation. The assumption point source, homogeneous and isotropic medium, simple Haskell [Haskell, 1964] source time function, and a double couple mechanism was taken. Additionally, the Brune model [Brune, 1970], [Brune, 1971] and Knopoff and Gillbert [Knopoff and Gilbert, 1959] are analysed to check the source time function impact on the estimates. Knopoff and Gillbert source time function is tested only in the frequency domain. The the radial component of displacement is analyzed, which partially equals the signal of the P wave and was used for moment tensor estimation in many anthropogenic seismicity cases for the moment tensor estimation e.g. [Wiejacz, 1992], [Lizurek *et al.*, 2017].

## 1.1 Source time functions

For simplicity the Haskell [Haskell, 1964] source time function is tested and, additionally, for comparison, the Brune model [Brune, 1970], [Brune, 1971] source time function is applied and the Knopoff and Gillbert [Knopoff and Gilbert, 1959] model only in the frequency domain.

The Haskell source time function is

$$M\left(t\right)=\begin{cases}0 & \text{for } t<0\\ tM_0/\tau & \text{for } 0\leqslant t\leqslant\tau\\ M_0 & \text{for } t>\tau\end{cases}, \tag{1.1}$$

where $M_0$ is the moment, and $\tau$ is the rupture time. The alternative form of (1.1) is

$$M\left(t\right)=\frac{M_0}{\tau}\left[tH\left(t\right)-\left(t-\tau\right)H\left(t-\tau\right)\right]$$

where $H\left(t\right)$ is Heaviside step function.

In the frequency domain, Haskell source function is

$$M\left(\omega\right)=\frac{M_0}{\tau\omega^2}\left[\exp\left(-j\omega\tau\right)-1\right], \tag{1.2}$$

where $\omega=2\pi f$.

The Brune source time function is

$$M\left(t\right)=M_0\left[1-\exp\left(-t/\tau\right)\left(t/\tau+1\right)\right]. \tag{1.3}$$

In the frequency domain, Brune source function is

$$M\left(\omega\right)=\frac{M_0\omega_0^2}{j\omega\left(\omega_0^2-\omega^2\right)}, \tag{1.4}$$

where $\omega_0 = 1/\tau$.

The simplest Knopoff and Gillbert model well displays the near and intermediate effect in the frequency domain, because

$$M(\omega) = M_0. \tag{1.5}$$

## 1.2 Displacement calculation

With assumptions that simplify the model, we use the total displacement in homogeneous and isotropic medium caused by the point double couple formula [Aki and Richards, 2002]

$$\begin{array}{llll} \mathbf{u}(\mathbf{r},t) & = 9\sin 2\theta\cos\phi\mathbf{R} & -6\left(\cos 2\theta\cos\phi\mathbf{\Theta}-\cos\theta\sin\phi\mathbf{\Phi}\right) & \frac{1}{4\pi\rho r^4}\int_{r/v_p}^{r/v_s}\tau M(t-\tau)\,d\tau \\ & +4\sin 2\theta\cos\phi\mathbf{R} & -2\left(\cos 2\theta\cos\phi\mathbf{\Theta}-\cos\theta\sin\phi\mathbf{\Phi}\right) & \frac{1}{4\pi\rho v_p^2 r^2}M(t-r/v_p) \\ & -3\sin 2\theta\cos\phi\mathbf{R} & +3\left(\cos 2\theta\cos\phi\mathbf{\Theta}-\cos\theta\sin\phi\mathbf{\Phi}\right) & \frac{1}{4\pi\rho v_s^2 r^2}M(t-r/v_s) \\ & +\sin 2\theta\cos\phi\mathbf{R} & & \frac{1}{4\pi\rho v_p^3 r}\dot{M}(t-r/v_p) \\ & & +\left(\cos 2\theta\cos\phi\mathbf{\Theta}-\cos\theta\sin\phi\mathbf{\Phi}\right) & \frac{1}{4\pi\rho v_s^3 r}\dot{M}(t-r/v_s), \end{array} \tag{1.6}$$

where $\theta$ and $\phi$ are ratiation angles, $\mathbf{R}$ is the unit vector of the source-receiver radial direction, $\mathbf{\Phi}$ is the perpendicular to the radial direction horizontal unit vector, and $\mathbf{\Theta}$ is the unit vector completing the coordinate system.

We will organize the formula (1.6) algorithmically as follows:

$$\mathbf{u}(\mathbf{r},t) = \mathbf{u}_R(\mathbf{r},t) + \mathbf{u_T}(\mathbf{r},t), \tag{1.7}$$

where $u_R$ is the radial part of the displacement, $u_T$ is the transversal part of the displacement.

$$\mathbf{u}_*(r,t) = \frac{\mathbf{R}^{N*}}{4\pi\rho r^4}\int_{r/v_p}^{r/v_s}\tau M(t-\tau)\,d\tau + \left[\frac{\mathbf{R}^{I*P}}{v_p^2} + \frac{\mathbf{R}^{I*S}}{v_s^2}\right]\frac{1}{4\pi\rho r^2}M(t) + \frac{\mathbf{R}^{F*}}{4\pi\rho v_*^3 r}\dot{M}(t), \tag{1.8}$$

where $*$ means either a radial ($R$) or transversal ($T$) member of (1.7), $v_* = v_p$ for radial component and $v_* = v_s$ for transversal component. radiation patterns of the near and intermediate fields depend on the far field radiation patterns according to $\mathbf{R}^{IRP} = 4\mathbf{R}^{FR}$, $\mathbf{R}^{IRS} = -3\mathbf{R}^{FR}$, $\mathbf{R}^{ITP} = -2\mathbf{R}^{FT}$, $\mathbf{R}^{ITS} = 3\mathbf{R}^{FT}$, $\mathbf{R}^{NR} = 9\mathbf{R}^{FT}$, $\mathbf{R}^{NT} = -6\mathbf{R}^{FT}$. Far field patterns exact description, which depend on the direction angles [Aki and Richards, 2002], has no significance for our research.

The assessment of the near field displacement required the calculation of $\int_{r/v_p}^{r/v_s}\tau M(t-\tau)\,d\tau$, which in the time domain is the convolution of the source time function $M(t)$ and the function described by the formula

$$G(t) = t(H(t-r/v_p) - H(t-r/v_s)), \tag{1.9}$$

where $H(t)$ is Heaviside step function. In the frequency domain, the calculation of the near field displacement required the multiplication of source complex function in the frequency domain $M(\omega)$ and the function

$$G(\omega) = \frac{(j\omega r/v_s+1)\,exp(-j\omega r/v_s) - (j\omega r/v_p+1)\,exp(-j\omega r/v_p)}{\omega^2} \tag{1.10}$$

where $\omega = 2\pi f$ and $j = \sqrt{-1}$ is.

The assessment of the far field displacement required the differentiate of source time function.

CHAPTER

TWO

# CONFIGURATION

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.

The configuration file (example name: `config.json`) is a file in JavaScript Object Notation (JSON) Here is the example file:

```
{
  "source_model": "Haskell",
  "green_function": "homogeneous",
  "dt": 0.001,
  "density": 2700,
  "radial_radiation": 1.0,
  "vp": 5000.0,
  "vs": 2900.0,
  "stop_simulation": "rupture_time",
  "source_parameters": [
    {"moment_scalar": 1e14, "rupture_time": 0.05},
    {"moment_scalar": 1e15, "rupture_time": 0.1},
    {"moment_scalar": 1e16, "rupture_time": 0.3}
  ],
  "inversion_type": "general",
  "distances": [500, 1000, 2000, 5000, 10000, 20000],
  "inventory": {
    "file_name": "VN_Stations.xml",
    "file_format": "STATIONXML"
  },
  "stream": {
    "source": "arclink",
    "host": "tytan.igf.edu.pl",
    "port": "18001",
    "user": "anonymous@igf.edu.pl",
    "timeout": 300,
    "net": "VN",
    "cache" : "cache_Mw"
  }
}
```

## 2.1 Configuration parameters

Below is the description of parameters.

**source_model**
(str) The source model name. Two values are allowed "Haskell", or "Brune"

**green_function**
(str) The Green function type name. So far only "homogeneous" is allowed.

**dt**
(float) The time sampling step for calculations

**density**
(float) The density at the source [kg/m^3]

**radial_radiation**
(float) The radial radiation absolute value in the far field.

**transversal_radiation**
(float) The transversal radiation absolute value in the far field. This parameter is not used so far in the assessment. Together with the radial_radiation, using both parameters required correct calculations according [Aki and Richards, 2002] page 79 or [Gibowicz and Kijko, 1994] page 192.

**vp**
(float) The P wave velocity [m/s].

**vs**
(float) The S wave velocity [m/s].

**stop_simulation**
(str or float) The information when stop simulation and signal visualization. There are three possibilities: "rupture_time" stops simulation at phase S time arrival + the rupture time + 0.5 s, "phase_time" stops simulation at phase S time arrival, where the digit value stops simulation after the defined number of seconds.

**source_parameters**
(list) The list of source parameters, that figures are plotted See *Source parameters description*.

**inversion_type**
The name of tensor inversion type. It must belong to the QuakeML MTInversionType category: `'general'`, `'zero trace'`, `'double couple'`, or None.

**inversion_type**
The focal mechanism inversion type name for choosing the focal mechanism.

**distances**
(list) The list of distances at which displacements plotted are plotted in each figure.

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

**stream**
(dict) *Stream parameters* describing how to get streams and inventories from the seismic data server (required only if the inventory file must be created, see. *Stream parameters*)

## 2.2 Source parameters description

The source parameters are dictionary of two items required to calculate the source.

**moment_scalar**
(float) The moment scalar of the DC seismic moment - $M_0$.

**rupture_time**
(float) The rupture time in the case of Haskell model. In other models it is the time parameters of the model.

## 2.3 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

## 2.4 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)

CHAPTER

THREE

# SSSPY API

## 3.1 Executable modules

Executable modules can be run. Hovewer, they can contain functions used in other modules.

### 3.1.1 Simple seismic simulation

**copyright**
Jan Wiszniowski (jwisz@igf.edu.pl)

**license**
GNU Lesser General Public License, Version 3 (https://www.gnu.org/copyleft/lesser.html)

**version 0.0.1**
2025-02-07

The program print displacements in near, far in time domain for various distances and in separate subplots for various source parameters: scalar moment and rupture time.

**call**
*python ssspy.py config.json*, where *config.json* is a configuration file (see *Configuration*)

`ssspy.`**`plot_simulations_radial_p`**(*configuration*)

Plots the figure containing P wave displacement in radial direction simulation for various source parameters in separate subplots. The configuration parameters must contain all required information for simulation and plotting. This procedure runs, when you call ssspy program.

**Parameters**
**`configuration`** (*`dict`*) – The configuration container of all parameters dictionary required for the program to work.

`ssspy.`**`time_simulate`**(*configuration*, *distance*, *source_parameters*, *ax=None*)

The time_simulate function simulate and optionally plots the displacement simulation in near, intermediate and fal fields for given distance, and source parameters

**Parameters**

- **`configuration`** (*`dict`*) – The configuration container of all parameters dictionary required for the program to work.
- **`distance`** (*`float`*) – The hypocentral distance in meters
- **`source_parameters`** (*`dict`*)
- **`ax`** (*`Matplotlib.Axes`*) – An object encapsulating an individual subplot in a figure. Missing or None parameter turn off plotting.

**Returns**

The tuple of values:

- The time $t_{max}$ from the rupture beginning, where displacement reaches the maximum value,
- The maximum displacement,
- The displacement in the near field at $t_{max}$,
- The displacement in the intermediate field at $t_{max}$,
- The displacement in the far field at $t_{max}$,
- The maximum displacement in the far field.

**Return type**

tuple(float, float, float, float, float, float)

### 3.1.2 Plot tests for the real catalog

**copyright**

Anna Tymińska (atyminska@igf.edu.pl) Jan Wiszniowski (jwisz@igf.edu.pl)

**license**

GNU Lesser General Public License, Version 3 (https://www.gnu.org/copyleft/lesser.html)

**version 0.0.1**

2025-02-07

The script facilitates a visual comparison of the behavior of far and intermediate fields for defined stations. The script utilizes numerical methods and visualization tools provided by the NumPy and Matplotlib libraries in Python. The script generates numerical simulations for distances (r) and time intervals (delta_t) defined from each station. It computes the far field (ff) and intermediate field (fi) behaviors based on specified parameters and radiation pattern coefficients. The visualization is facilitated through two sets of plots: 3D visualization and set of 2D log-log plots.

**call**

*python ssspy.py config.json catalog.xml*, where *config.json* is a configuration file (see *Configuration*) and *catalog.xml* is a catalog file in QuakeML.

`ssscat.`**`fields_on_station`**(*configuration*, *catalog*)

**Description required here**

**Parameters**

- **`configuration`** (*`dict`*) – The configuration container of all parameters dictionary required for the program to work.
- **`catalog`** (*`ObsPy.Catalog`*) – The events catalog

## 3.2 Library modules

The library modules contains classes and function required in ether module includin executable modules.

### 3.2.1 Utils for simple seismic simulation

**copyright**
Jan Wiszniowski (jwisz@igf.edu.pl)

**license**
GNU Lesser General Public License, Version 3 (https://www.gnu.org/copyleft/lesser.html)

**version 0.0.1**
2025-02-07

**exception** `utils.`**`SSSException`**(*message='other'*)

The simple seismic signal simulation exception class

`utils.`**`extract_amplitude_delta`**(*event*, *station_code*)

Function extracting from the event parameters displacement amplitude and amplitude duration, which is treated as the event rupture time, of the station first peak amplitude with mechanism data.

**Parameters**

- **event** (*ObsPy.Event*) – The event object
- **station_code** (*(str)*) – The station code

**Returns**
The displacement amplitude and its duration

**Return type**
tuple(float, float)

### 3.2.2 Simple seismic source models

**copyright**
Jan Wiszniowski (jwisz@igf.edu.pl)

**license**
GNU Lesser General Public License, Version 3 (https://www.gnu.org/copyleft/lesser.html)

**version 0.0.1**
2025-02-07

**class** `source_models.`**`BaseSourceModel`**(*source_parameters*)

**class** `source_models.`**`BruneSourceModel`**(*source_parameters*)

Brune source model in the time domain is described as

$$M\left(t\right)=M_0\left[1-\exp\left(-t/\tau\right)\left(t/\tau+1\right)\right],$$

where $M_0$ is the seismic moment value, $\tau$ is the rupture time, and $H(t)$ is Heaviside step function.

**class** `source_models.`**`HaskellSourceModel`**(*source_parameters*)

Haskell source model in the time domain is described as

$$M\left(t\right)=\begin{cases}0 & \text{for } t<0\\ tM_0/\tau & \text{for } 0\leqslant t\leqslant\tau\\ M_0 & \text{for } t>\tau\end{cases},$$

where $M_0$ is the seismic moment value, $\tau$ is the rupture time, and $H(t)$ is Heaviside step function.

### 3.2.3 Green function in the time domain

**copyright**
Jan Wiszniowski (jwisz@igf.edu.pl)

**license**
GNU Lesser General Public License, Version 3 (https://www.gnu.org/copyleft/lesser.html)

**version 0.0.1**
2025-02-07

`class green_functions.BaseGreenFunction`(*dt*, *density*, *transversal_radiation=1.0*, *radial_radiation=1.0*)

The base class of Green function classes. It required in derived classes definitions of three functions:

- near(self, source_model, distance, vp, vs, times)
- intermediate(self, source_model, distance, vp, vs, times)
- far(self, source_model, distance, vp, vs, times)

They return radial and transversal displacement responses.

**Parameters**

- **dt** – The time sampling step for integration and differentiation calculations,
- **density** – The density at the source,
- **transversal_radiation** – The transversal_radiation pattern in the far field,
- **radial_radiation** – The radial radiation pattern in the far field.

`abstract far`(*source_model*, *distance*, *vp*, *vs*, *times*, *phase='P'*)

Compute the far part of the displacement

**Parameters**

- **source_model** – The source model object,
- **distance** – The hypocentral distance,
- **vp** – The P wave velocity,
- **vs** – The S wave velocity,
- **times** – Time samples. The samples steps must equal dt,
- **phase** – The phase name: 'P' or 'S',

**Returns**
The radial and transversal displacement in the far field,

`abstract intermediate`(*source_model*, *distance*, *vp*, *vs*, *times*, *phase='P'*)

Compute the intermediate part of the displacement

**Parameters**

- **source_model** – The source model object,
- **distance** – The hypocentral distance,
- **vp** – The P wave velocity,
- **vs** – The S wave velocity,
- **times** – Time samples. The samples steps must equal dt,
- **phase** – The phase name: 'P' or 'S',

**Returns**
The radial and transversal displacement in the far field,

`abstract near`(*source_model*, *distance*, *vp*, *vs*, *times*)

Compute the near part of the displacement

**Parameters**

- **source_model** – The source model object,
- **distance** – The hypocentral distance,
- **vp** – The P wave velocity,
- **vs** – The S wave velocity,
- **times** – Time samples. The samples steps must equal dt,

**Returns**
The radial and transversal displacement in the far field,

`class green_functions.HomogeneousGreenFunction`(*dt*, *density*, *transversal_radiation=1.0*, *radial_radiation=1.0*)

The Green function in the homogeneous and isotropic medium

**Parameters**

- **dt** – The time sampling step for integration and differentiation calculations
- **density** – The density at the source
- **transversal_radiation** – The transversal_radiation pattern in the far field
- **radial_radiation** – The radial radiation pattern in the far field

`far`(*source_model*, *distance*, *vp*, *vs*, *times*, *phase='P'*)

Compute the far part of the displacement of the Green function in the homogeneous and isotropic medium.

$$u_{*}\left(r,t\right)=\frac{R^{I*}}{4\pi\rho v^{3}r}\dot{M}\left(t\right),$$

where :math * means radial or transversal part, which in the case of far field is equivalent of the P or S wave, $u\left(r,t\right)$ is the displacement, $R^{F*}$ is the radiation of radial or transversal near field pattern, $r$ is the hypocentral distance, $\rho$ is the density at the source, $v$ is the P or S velocity, $\dot{M}\left(t\right)$ is the time derivative of source time function.

**Parameters**

- **source_model** – The source model object.
- **distance** – The hypocentral distance
- **vp** – The P wave velocity.
- **vs** – The S wave velocity.
- **times** – Time samples. The samples steps must equal dt.
- **phase** – The phase name: 'P' or 'S'

**Returns**
The radial and transversal displacement in the near field

`intermediate`(*source_model*, *distance*, *vp*, *vs*, *times*, *phase='P'*)

Compute the intermediate part of the displacement of the Green function in the homogeneous and isotropic medium.

$$u_* (r,t) = \frac{R^{I*}}{4\pi\rho v^2 r^2} M(t),$$

where :math * means radial or transversal part, $u(r,t)$ is the displacement, $R^{I*}$ is the radiation of radial or transversal near field pattern, $r$ is the hypocentral distance, $\rho$ is the density at the source, $v$ is the P or S velocity, $M(t)$ is the source time function. For P wave $R^{IR} = 4R^{FR}$ and $R^{IT} = -2R^{FT}$ (see far field radiation), for S wave $R^{IR} = -3R^{FR}$ and $R^{IT} = 3R^{FT}$ (see far field radiation).

**Parameters**

- **source_model** – The source model object.
- **distance** – The hypocentral distance
- **vp** – The P wave velocity.
- **vs** – The S wave velocity.
- **times** – Time samples. The samples steps must equal dt
- **phase** – The phase name: 'P' or 'S'

**Returns**

The radial and transversal displacement in the near field

`near`(*source_model*, *distance*, *vp*, *vs*, *times*)

Compute the near part of the displacement of the Green function in the homogeneous and isotropic medium.

$$u_* (r,t) = \frac{R^{N*}}{4\pi\rho r^4} \int_{r/v_p}^{r/v_s} \tau M(t-\tau)\, d\tau,$$

where * means radial or transversal part, $u(r,t)$ is the displacement, $R^{N*}$ is the radiation of radial or transversal near field pattern $R^{NR} = 9R^{FR}$ and $R^{NT} = -6R^{FT}$ (see far field radiation), $r$ is the hypocentral distance, $\rho$ is the density at the source, $v_p$ and $v_s$ are P and S velocities at the source, $M(t)$ is the source time function. The integration is realised by the convolution of the source time function nad the signal $t(H(t-r/v_p) - H(t-r/v_s))$, where $H(t)$ is Heaviside step function.

**Parameters**

- **source_model** – The source model object.
- **distance** – The hypocentral distance
- **vp** – The P wave velocity.
- **vs** – The S wave velocity.
- **times** – Time samples. The samples steps must equal dt

**Returns**

The radial and transversal displacement in the near field

## 3.3 Core modules

Core modules are used in the SSSPy package, but are designated for more general use and use in other packages,

### 3.3.1 The waveform and inventory manipulation

**copyright**
Jan Wiszniowski (jwisz@igf.edu.pl)

**license**
GNU Lesser General Public License, Version 3 (https://www.gnu.org/copyleft/lesser.html)

**version 0.0.1**
2025-01-15

`class 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 core.signal_utils.SignalException`(*message='other'*)

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

The base class of streams preprocessing

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

`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

`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')

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

**copyright**
Jan Wiszniowski (jwisz@igf.edu.pl)

**license**
GNU Lesser General Public License, Version 3 (https://www.gnu.org/copyleft/lesser.html)

**version 0.0.1**
2024-11-07

`class 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 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** 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

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

`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)

`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

`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)

`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

`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

`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

`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

`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)
```

`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
```

`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)
```

`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
```

# BIBLIOGRAPHY


[1] N. A. Haskell. Total energy and energy spectral density of elastic wave radiation from propagating faults. *Bulletin of the Seismological Society of America*, 54(6A):1811–1841, 12 1964. URL: https://doi.org/10.1785/BSSA05406A1811, arXiv:https://pubs.geoscienceworld.org/ssa/bssa/article-pdf/54/6A/1811/5348868/bssa05406a1811.pdf, doi:10.1785/BSSA05406A1811.

[2] 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.

[3] 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.

[4] Leon Knopoff and Freeman Gilbert. Radiation from a strike-slip fault. *Bulletin of the Seismological Society of America*, 49(2):163–178, 04 1959. URL: https://doi.org/10.1785/BSSA0490020163, arXiv:https://pubs.geoscienceworld.org/ssa/bssa/article-pdf/49/2/163/5303329/bssa0490020163.pdf, doi:10.1785/BSSA0490020163.

[5] Paweł Wiejacz. Calculation of seismic moment tensor for mine tremors from the legnica-głogów copper basin. *Acta Geophysica Polonica*, XI(2):103–122, 1992.

[6] Grzegorz Lizurek, Jan Wiszniowski, Nguyen Van Giang, Beata Plesiewicz, and Dinh Quoc Van. Clustering and stress inversion in the song tranh 2 reservoir, vietnam. *Bulletin of the Seismological Society of America*, 107(6):2636–2648, 09 2017. URL: https://doi.org/10.1785/0120170042, arXiv:https://pubs.geoscienceworld.org/ssa/bssa/article-pdf/107/6/2636/3992585/bssa-2017042.1.pdf, doi:10.1785/0120170042.

[7] Keiiti Aki and Paul G. Richards. *Quantitative Seismology*. University Science Books, 2 edition, 2002. ISBN 0935702962. URL: http://www.worldcat.org/isbn/0935702962.

[8] Sławomir Jerzy Gibowicz and Andrzej Kijko. *An introduction to mining seismology*. Academic Press, San Diego, 1 edition, 1994. ISBN 9780122821202. URL: https://api.semanticscholar.org/CorpusID:126723846.

# PYTHON MODULE INDEX

# INDEX

## T



## U