swprocess
Release 0.1.0b0

Joseph P. Vantassel

Apr 21, 2021
# CONTENTS

1 Contents: 2
2 Indices and tables 22
Python Module Index 23
Index 24
swprocess is a Python package for surface wave processing. It includes features for performing active-source surface wave processing [i.e., mulitchannel analysis of surface wave (MASW)], post-processing passive-wavefield surface wave data [i.e., microtremor array measurements (MAM)] processed using the open-source software Geopsy, and combining active-source and/or passive-wavefield data to calculate rigorous statistics on the surface wave dispersion data, which account for epistemic and aleatory uncertainty.

This package is actively being developed, so if you do not see a feature you would like it may very well be under development and released in the near future. To be notified of future releases, you can either watch the repository on Github or Subscribe to releases on the Python Package Index (PyPI).
1.1 Installation

```
pip install swprocess or pip install swprocess --upgrade
```
pip will handle the rest!

1.2 API Reference

1.2.1 activetimeseries

ActiveTimeSeries class definition.

```
class ActiveTimeSeries(amplitude, dt, nstacks=1, delay=0)
Bases: sigpropy.timeseries.TimeSeries
```

A class for working with active-source TimeSeries.

**Variables**

- `amplitude (ndarray)` – Recording’s amplitude, one per sample.
- `dt (float)` – Time step between samples in seconds.

**__init__**(amplitude, dt, nstacks=1, delay=0)

Initialize an ActiveTimeSeries object.

**Parameters**

- `amplitude (array-like)` – Recording’s amplitude, one per sample. The first value is associated with \( time=0 \) seconds and the last is associate with \( time=(\text{len}(amplitude)-1)*dt \) seconds.
- `dt (float)` – Time step between samples in seconds.
- `nstacks (int, optional)` – Number of stacks used to produce amplitude, default is 1.
- `delay (float, optional)` – Delay to the start of the record in seconds, default is 0.

**Returns** ActiveTimeSeries – Initialized ActiveTimeSeries object.

```
class crosscorr(a, b, correlate_kwargs=None, exclude='nsamples')
```

Cross correlation of two ActiveTimeSeries objects.

**Parameters**

- `a (ActiveTimeSeries)` – Base ActiveTimeSeries to which \( b \) is correlated.
• \( \mathbf{b} \) (ActiveTimeSeries) – ActiveTimeSeries correlated to \( \mathbf{a} \).

• \( \texttt{correlate_kwargs} \) (dict, optional) – dict of keyword argument for the correlate function, see scipy.signal.correlate for details.

• \( \texttt{exclude} \) (tuple, optional) – tuple of attributes to exclude in an is_similar comparison, default is ‘nsamples’.

Returns ndarray – Containing the cross correlation.

\[
\text{static crosscorr_shift}(\mathbf{a}, \mathbf{b}, \text{exclude=\texttt{None}}) \]

Shift \( \mathbf{b} \) so that it is maximally correlated with \( \mathbf{a} \).

Parameters

• \( \mathbf{a} \) (ActiveTimeSeries) – ActiveTimeSeries to which \( \mathbf{b} \) will be correlated. \( \mathbf{a} \) should be similar to \( \mathbf{b} \).

• \( \mathbf{b} \) (ActiveTimeSeries) – ActiveTimeSeries which will be shifted so that it is maximally correlated with \( \mathbf{a} \). \( \mathbf{b} \) should be similar to \( \mathbf{a} \).

• \( \texttt{exclude} \) (tuple, optional) – tuple of attributes to exclude in an is_similar comparison, default is ‘nsamples’.

Returns ndarray – Which represents the stack of the correlated and padded \( \mathbf{b} \) onto \( \mathbf{a} \).

property delay

property df

classmethod from_activetimeseries (activetimeseries)

classmethod from_cross_stack (\( \mathbf{a}, \mathbf{b} \))

Create ActiveTimeSeries from cross-correlation.

Cross-correlate \( \mathbf{b} \) to \( \mathbf{a} \) and shift \( \mathbf{b} \) such that it is maximally correlated with \( \mathbf{a} \). Then stack the shifted version of \( \mathbf{b} \) onto \( \mathbf{a} \).

Parameters

• \( \mathbf{a} \) (ActiveTimeSeries) – ActiveTimeSeries to which \( \mathbf{b} \) will be correlated and stacked. \( \mathbf{a} \) should be similar to \( \mathbf{b} \).

• \( \mathbf{b} \) (ActiveTimeSeries) – ActiveTimeSeries which will be correlated with and stacked onto \( \mathbf{a} \). \( \mathbf{b} \) should be similar to \( \mathbf{a} \).

Returns ActiveTimeSeries – Which represents the correlated and potentially zero-padded \( \mathbf{b} \) stacked onto \( \mathbf{a} \).

classmethod from_trace (trace, nstacks=1, delay=0)

Create ActiveTimeSeries from a Trace object.

This method is more general than ActiveTimeSeries.from_trace_seg2(), as it does not attempt to extract any metadata from the Trace object.

Parameters

• \( \text{trace} \) (Trace) – Refer to obspy documentation for more information.

• \( \texttt{nstacks} \) (int, optional) – Number of stacks the time series represents, default is 1, signifying a single unstacked time record.

• \( \texttt{delay} \) (float \( \leq 0 \), optional) – Denotes the pre-event delay, default is zero, meaning no pre-event noise was recorded.

Returns ActiveTimeSeries – Initialized with information from \( \text{trace} \).
classmethod from_trace_seg2 (trace)
    Initialize from a SEG2 Trace object.
    This method is similar to ActiveTimeSeries.from_trace() except that it extracts additional in-
formation from the Trace header. So only use this method if you have a seg2 file and the header information
is correct.

    Parameters  trace (Trace) – Trace object from a correctly written seg2 file.
    Returns  ActiveTimeSeries – Instantiated with seg2 file.

property multiple
property n_stacks
property nstacks
stack_append (timeseries)
    Stack (i.e., average) a new timeseries onto the current one.

    Parameters  timeseries (ActiveTimeSeries) – ActiveTimeSeries to be stacked onto the current
object.
    Returns  None – Updates the attributes amplitude and nstacks.
    Raises ValueError – If timeseries is not an ActiveTimeSeries or it cannot be stacked to the
    current object (i.e., the two are dissimilar).

property time
    Time vector for ActiveTimeSeries.
trim (start_time, end_time)
    Trim in the interval \([start_time, end_time]\).
    For more information see sigpropy.TimeSeries.trim().

    Parameters
        • start_time (float) – New time-zero in seconds.
        • end_time (float) – New end-time in seconds.
    Returns  None – Updates the attributes nsamples and delay.

zero_pad (df)
    Append zeros to amp to achieve a desired frequency step.
    Note for exact results, \(1/(df*\Delta t)\) must be an integer, otherwise a \(df\) close to the desired \(df\) will be returned.

    Parameters  df (float) – Desired frequency step in Hertz.
    Returns  None – Instead modifies attributes: amp, nsamples, multiple.
    Raises ValueError – If \(df < 0\) (i.e., non-positive).
1.2.2 array1d

Array1D class definition.

class Array1D(sensors, source)

    Bases: object

    A class to organize the information for a 1D (linear) array.

    Variables

    • sensors (list of Sensor1C) – Sensors which compose the 1D array.
    • source (Source) – Source for active shot gather.

    __init__ (sensors, source)

    Initialize from an iterable of Sensor1C’s and a `Source`.

    Parameters

    • sensors (iterable of Sensor1c) – Iterable of initialized Sensor1C objects.
    • source (Source) – Initialized Source object.

    Returns Array1D – Initialized Array1D object.

property array_center_distance

auto_pick_first_arrivals (algorithm='threshold', **algorithm_kwargs)

classmethod from_array1d (array1d)

    Create a deep copy of an existing Array1D object.

classmethod from_files (fnames, map_x=<function Array1D.<lambda>>, map_y=<function Array1D.<lambda>>)

    Initialize an Array1D object from one or more data files.

    This classmethod creates an Array1D object by reading the header information in the provided file(s). Each file should contain multiple traces where each trace corresponds to a single receiver. Currently supported file types are SEG2 and SU.

    Parameters

    • fnames (str or iterable) – File name or iterable of file names. If multiple files are provided the traces are stacked.
    • map_x, map_y (function, optional) – Convert x and y coordinates using some function, default is not transformation. Can be useful for converting between coordinate systems.

    Returns Array1D – Initialized Array1d object.

    Raises TypeError – If fnames is not of type str or iterable.

interactive_mute (mute_location='both', window_kwargs=None, waterfall_kwargs=None)

    Interactively select source window boundary.

    Parameters

    • mute_location ("before", "after", "both", optional) – Select which part of the record to mute, default is “both” indicating two lines defining the source window boundary will be required.
    • window_kwargs (dict, optional) – Dictionary of keyword arguments defining the signal window, see scipy.signal.windows.tukey for available options.
    • waterfall_kwargs (dict, optional) – Dictionary of keyword arguments defining how the waterfall should be created, see :meth Array1D.waterfall for the available options.
Returns tuple – Of the form (signal_start, signal_end).

is_similar (other)
Check if other is similar to self.

property kres
The array’s resolution wavenumber.

manual_pick_first_arrivals (waterfall_kwargs=None)
Allow for interactive picking of first arrivals.

Parameters waterfall_kwargs (dict, optional) – Dictionary of keyword arguments for meth: <Array1D.waterfall>, default is None indicating default keyword arguments.

Returns Tuple – Of the form (distance, picked_time)

mute (signal_start=None, signal_end=None, window_kwargs=None)
Mute traces outside of a narrow signal window.

Parameters
• signal_start, signal_end (iterable of floats, optional) – Two points to define start and stop of the narrow signal window of the form ((pt1_dist, pt1_time), (pt2_dist, pt2_time)), default is None .

• window_kwargs (dict, optional) – Dictionary of keyword arguments defining the signal window, see scipy.signal.windows.tukey for available options.

Returns None – Modifies the object internal state.

property nchannels
Number of Sensors in the array.

property offsets
Receiver offsets relative to source position as list.

plot (ax=None, sensor_kwargs=None, source_kwargs=None)
Plot a schematic of the Array1D object.

The schematic shows the position of the receivers and source and lists the total number of receivers and their spacing.

Parameters
• ax (Axis, optional) – Axes on which to plot, default is None indicating a Figure and Axis will be generated on-the-fly.

• sensor_kwargs, source_kwargs (None, dict, optional) – Kwargs for matplotlib.pyplot.plot to control the plotting of the sensors and source, respectively. Default is None, indicating the predefined default values will be used.

Returns Tuple – Of the form (fig, ax) where fig is the figure object and ax the axes object on which the schematic is plotted, if ax=None.

position (normalize=False)
Array’s sensor positions as list.

Parameters normalize (bool, optional) – Determines whether the array positions are shifted such that the first sensor is located at x=0.

property spacing

timeseriesmatrix (detrend=False, normalize='none')
Sensor amplitudes as 2D ndarray.
Parameters

- **detrend** *(bool, optional)* – Boolean to control whether a linear detrending operation is performed, default is `False` so no detrending is performed.

- **normalize** *({'none', 'each', 'all'}, optional)* – Enable different normalizations to be performed. “each” normalizes each traces by its maximum. “all” normalizes all traces by the same maximum. Default is “none” so no normalization is performed.

Returns *ndarray* – Of shape *(nchannels, nsamples)* where each row is the amplitude of a given sensor.

to_file *(fname, ftype='su')*

trim *(start_time, end_time)*

Trim time series belonging to each Sensor1C.

Parameters

- **start_time, end_time** *(float)* – Desired start time and end time in seconds measured from the point the acquisition system was triggered.

Returns *None* – Updates internal attributes.

**waterfall** *(ax=None, time_ax='y', amplitude_detrend=True, amplitude_normalization='each', amplitude_scale=None, position_normalization=False, plot_kwargs=None)*

Create waterfall plot for this array setup.

Parameters

- **ax** *(Axes, optional)* – Axes on which to plot, default is `None` indicating a `Figure` and `Axes` will be generated on-the-fly.

- **time_ax** *({'x', 'y'}, optional)* – Denotes the time axis, ‘y’ is the default.

- **amplitude_detrend** *(bool, optional)* – Boolean to control whether a linear detrending operation is performed, default is `False` so no detrending is performed.

- **amplitude_normalization** *({'none', 'each', 'all'}, optional)* – Enable different normalizations including: “each” which normalizes each traces by its maximum, “all” which normalizes all traces by the same maximum, and “none” which perform no normalization, default is “each”.

- **amplitude_scale** *(float, optional)* – Factor by which each trace is multiplied, default is `None` which uses a factor equal to half the average receiver receiver spacing.

- **position_normalization** *(bool, optional)* – Determines whether the array positions are shifted such that the first sensor is located at x=0.

- **plot_kwargs** *(None, dict, optional)* – Kwargs for `matplotlib.pyplot.plot` to control the style of each trace, default is `None`.

Returns *Tuple* – Of the form *(fig, ax)* where `fig` is the figure object and `ax` the axes object on which the schematic is plotted, if `ax=None`.

**zero_pad** *(df)*

Append zero to sensors to achieve a desired frequency step.

Parameters **df** *(float)* – Desired linear frequency step in Hertz.

Returns *None* – Instead modifies *sensors*.
1.2.3 masw

Masw class definition.

```python
class Masw
    Bases: object

Customizable Multichannel Analysis of Surface Waves workflow.
Convenient customer-facing interface for implementing different and extensible MASW processing workflows.
```

```python
static create_settings_dict
    (workflow='time-domain', trim=False, trim_begin=0.0, trim_end=1.0, mute=False, method='interactive', window_kwargs=None, pad=False, df=1.0, transform='fdbf', fmin=5, fmax=100, vmin=100, vmax=1000, nvel=200, vspace='linear', weighting='sqrt', steering='cylindrical', snr=False, noise_begin=-0.5, noise_end=0.0, signal_begin=0.0, signal_end=0.5, pad_snr=True, df_snr=1.0)
```

Create settings dict using function arguments.
See `Masw.create_settings_file()` for details.

```python
static run
    (fnames, settings, map_x=<function Masw.<lambda>>, map_y=<function Masw.<lambda>>)
```

Run an MASW workflow from SU or SEGY files.
Create an instance of an `Masw` object for a specific `Masw` workflow. Note that each file should contain multiple traces where each trace corresponds to a single receiver. The header information for these files must be correct and readable. Currently supported file types are SEGY and SU.

Parameters
- `fnames` *(str or iterable of str)* – File name or iterable of file names.
- `settings` *(str)* – JSON settings file detailing how MASW should be performed. See `Masw.create_settings_file()` for more information.
- `map_x`, `map_y` *(function, optional)* – Functions to convert the x and y coordinates of source and receiver information, default is no transformation. Useful for converting between coordinate systems.

Returns `AbstractTransform-like` – Initialized subclass (i.e., child) of `AbstractTransform`.

Raises `TypeError` – If `fnames` is not of type `str` or `iterable`.

1.2.4 masw workflows

Masw workflow class definitions.

```python
class AbstractMaswWorkflow
    (fnames=None, settings=None, map_x=None, map_y=None)
```

Abstract base class (ABC) defining an MASW workflow.

```python
__init__
```

Perform initialization common to all MaswWorkflows.

```python
calculate_snr()
```

```python
check()
```

Check array is acceptable for WavefieldTransform.

1.2. API Reference


detrend()
    Perform linear detrend operation.

mute()
    Mute record in the time domain.

pad()
    Pad record in the time domain.

abstract run()

select_noise()
    Select a portion of the record as noise.

select_signal()
    Select a portion of the record as signal.

trim()
    Trim record in the time domain.

class FrequencyDomainMaswWorkflow (fnames=None, settings=None, map_x=None, map_y=None)
    Bases: swprocess.maswworkflows.AbstractMaswWorkflow
    Stack in the frequency-domain.

    run()

class SingleMaswWorkflow (fnames=None, settings=None, map_x=None, map_y=None)
    Bases: swprocess.maswworkflows.TimeDomainWorkflow
    Perform transform on a single time-domain record.

    run()

class TimeDomainMaswWorkflow (fnames=None, settings=None, map_x=None, map_y=None)
    Bases: swprocess.maswworkflows.TimeDomainWorkflow
    Stack in the frequency-domain.

class TimeDomainWorkflow (fnames=None, settings=None, map_x=None, map_y=None)
    Bases: swprocess.maswworkflows.AbstractMaswWorkflow

    run()

1.2.5 peaks

Peaks class definition.

class Peaks (frequency, velocity, identifier='0', **kwargs)
    Bases: object
    Class for handling dispersion peaks.

    Variables

    • frequency (ndarray) – Frequency associate with each peak.
    • velocity (ndarray) – Velocity associate with each peak.
    • identifier (str) – Used to uniquely identify the Peaks object.
    • attrs (list) – List of strings describing Peak attributes.
__init__ (frequency, velocity, identifier='0', **kwargs)
Create Peaks from a iterable of frequencies and velocities.

Parameters

- **frequency, velocity** (iterable of floats) – Frequency and velocity (one per peak), respectively.
- **identifier** (str, optional) – String to uniquely identify the provided Peaks, default is “0”.
- **kwargs** (kwargs) – Optional keyword argument(s) these may include additional information about the dispersion peaks such as: azimuth, ellipticity, power, and noise. Will generally not be entered directly.

Returns Peaks – Instantiated Peaks object.

axes_defaults = {'azimuth': {'label': 'Azimuth (deg)', 'scale': 'linear'}, 'frequency': {'label': 'Frequency (Hz)', 'scale': 'linear'}, 'velocity': {'label': 'Velocity (m/s)', 'scale': 'linear'}, 'wavelength': {'label': 'Wavelength (m)', 'scale': 'log'}}

property azimuth

property ellipticity

property extended_attrs
List of available Peaks attributes, including calculated.

property frequency

classmethod from_dict (data_dict, identifier='0')
Initialize Peaks from dict.

Parameters

- **data_dict** (dict) – Of the form {"frequency":freq, “velocity”:vel, “kwarg1”: kwarg1} where freq is a list of floats denoting frequency values. vel is a list of floats denoting velocity values. kwarg1 is an optional keyword argument denoting some additional parameter (may include more than one).
- **identifiers** (str) – String to uniquely identify the provided Peaks object.

Returns Peaks – Initialized Peaks instance.

classmethod from_json (fname)
Read Peaks from json file.

Parameters fnames (str) – Name of the input file, may contain a relative or the full path.

Returns Peaks – Initialized Peaks object.

classmethod from_max (fname, wavetype='rayleigh')
Initialize a Peaks object from a .max file.

Parameters

- **fname** (str) – Denotes the filename for the .max file, may include a relative or the full path.
- **wavetype** (‘rayleigh’, ‘love’, optional) – Wavetype to extract from file, default is ‘rayleigh’.

Returns Peaks – Initialized Peaks object.
Notes

If the results from multiple time windows are in the same .max file, as is most often the case, this method ignores all but the first instance found.

property noise

plot (xtype='frequency', ytype='velocity', plot_kwargs=None, mask=None)

Plot dispersion data in Peaks object.

Parameters

• xtype (‘frequency’, ‘wavelength’, optional) – Denote whether the x-axis should be either frequency or wavelength, default is frequency.

• ytype (‘velocity’, ‘slowness’, optional) – Denote whether the y-axis should be either velocity or slowness, default is velocity.

• plot_kwargs (dict, optional) – Keyword arguments to pass along to ax.plot, default is None indicating the predefined settings should be used.

• mask (ndarray, optional) – Boolean array mask to determine which points are to be plotted, default is None so all valid points will be plotted.

Returns tuple – Of the form (fig, ax) where fig and ax are the Figure and Axes objects which were generated on-the-fly.

property power

reject_box_inside (xtype, xlims, ytype, ylims)

Reject peaks inside the stated limits.

Parameters

• xtype, ytype (‘frequency’, ‘velocity’, ‘slowness’, ‘wavelength’) – Parameter domain in which the limits are defined.

• xlims, ylims (tuple) – Tuple with the lower and upper limits for each of the boundaries.

Returns None – Updates the Peaks object’s state.

reject_limits_outside (attr, limits)

Reject peaks outside the stated bounds.

Parameters

• attr (‘frequency’, ‘velocity’, ‘slowness’, ‘wavelength’) – Parameter domain in which the limits are defined.

• limits (tuple) – Tuple with the lower and upper limits. None may be used to perform one-sided rejections. For example limits=(None, 5) will reject all values above 5 and limits=(5, None) will reject all values below 5.

Returns None – Updates the Peaks object’s state.
Notes

This method is somewhat similar to `swprocess.Peaks.reject_inside()`, but is more computationally expensive.

`simplify_mpeaks(attr)`

Produce desired attribute with multiple peaks removed.

**Parameters**
- `attr` (f"`frequency", "velocity", "azimuth", "power", "ellipticity", "noise") – Attribute of interest.

**Returns**
- `ndarray` – With the attribute of interest simplified to remove duplicate peaks.

`property slowness`

`to_json(fname, append=False)`

Write `Peaks` to json file.

**Parameters**
- `fname` (str) – Output file name, can include a relative or the full path.
- `append` (bool, optional) – Controls whether `fname` (if it exists) should be appended to or overwritten, default is `False` indicating `fname` will be overwritten.

**Returns**
- `None` – Instead writes file to disk.

`property velocity`

`property wavelength`

`property wavenumber`

### 1.2.6 peakssuite

PeaksSuite class definition.

```python
class PeaksSuite(peaks)
    Bases: object

    __init__(peaks)
    Instantiate a PeaksSuite object from a Peaks object.

    **Parameters**
    - `peaks` (Peaks) – A Peaks object to include in the suite.

    **Returns**
    - `PeaksSuite` – Instantiated PeaksSuite object.

append(peaks)
    Append a Peaks object to PeaksSuite.

    **Parameters**
    - `peaks` (Peaks) – A Peaks object to include in the suite.

    **Returns**
    - `None` – Appends Peaks to PeaksSuite.

static calc_resolution_limits(xtype, attribute, ytype, limits, xs, ys)
    Calculate resolution limits for a variety of domains.

classmethod from_dict(dicts)
    Instantiate PeaksSuite from list of dict.

    **Parameters**
    - `dicts` (list of dict or dict) – List of dict or a single dict containing dispersion data.

    **Returns**
    - `PeaksSuite` – Instantiated PeaksSuite object.
```
classmethod from_json(fnames)
    Instantiate PeaksSuite from json file(s).

Parameters
    fnames (list of str or str) – File name or list of file names containing dispersion data. Names may contain a relative or the full path.

Returns
    PeaksSuite – Instantiated PeaksSuite object.

classmethod from_max(fnames, wavetype='rayleigh')
    Instantiate PeaksSuite from .max file(s).

Parameters
    • fnames (list of str or str) – File name or list of file names containing dispersion data. Names may contain a relative or the full path.
    • wavetype (‘rayleigh’, ‘love’), optional) – Wavetype to extract from file, default is ‘rayleigh’.

Returns
    Peaks – Initialized PeaksSuite object.

classmethod from_peaks(peaks)
    Instantiate PeaksSuite from iterable of Peaks.

Parameters
    peaks (iterable) – Iterable containing Peaks objects.

Returns
    PeaksSuite – Instantiated PeaksSuite object.

classmethod from_peakssuite(peakssuites)
    Instantiate PeaksSuite from iterable of PeaksSuite.

Parameters
    peakssuites (iterable) – Iterable containing PeaksSuite objects.

Returns
    PeaksSuite – Instantiated PeaksSuite object.

interactive_trimming(xtype='wavelength', ytype='velocity', plot_kwargs=None, resolution_limits=None, resolution_limits_plot_kwargs=None, margins=0.1)
    Interactively trim experimental dispersion data.

Parameters
    • xtype (‘frequency’, ‘wavelength’), optional) – Denote whether the x-axis should be either frequency or wavelength, default is frequency.
    • ytype (‘velocity’, ‘slowness’), optional) – Denote whether the y-axis should be either velocity or slowness, default is velocity.
    • plot_kwargs (dict, optional) – Keyword arguments to pass along to ax.plot can be in the form plot_kwargs = {“key”:value_allpeaks} or plot_kwargs = {“key”:[value_peaks0, value_peaks1, ...]}, default is None indicating the predefined settings should be used.
    • resolution_limits (iterable, optional) – Of form (“domain”, (min, max)) where “domain” is a str denoting the domain of the limits and min and max are floats denoting their value, default is None so no resolution limits are plotted for reference.
    • resolution_limits_plot_kwargs (dict, optional) – Formatting of resolution limits passed to ax.plot, default is None so default settings will be used.

Returns
    None – Updates the PeaksSuite state.

plot(xtype='frequency', ytype='velocity', ax=None, plot_kwargs=None, mask=None)
    Plot dispersion data in Peaks object.

Parameters
• **xtype** ('frequency', 'wavelength', optional) – Denote whether the x-axis should be either frequency or wavelength, default is frequency.

• **ytype** ('velocity', 'slowness', optional) – Denote whether the y-axis should be either velocity or slowness, default is velocity.

• **ax** (Axes, optional) – Axes object on which to plot the dispersion peaks, default is None so Axes will be generated on-the-fly.

• **plot_kwargs** (dict, optional) – Keyword arguments to pass along to ax.plot can be in the form plot_kwargs = {'key':value_allpeaks} or plot_kwargs = {'key':[value_peaks0, value_peaks1, ...]}, default is None indicating the predefined settings should be used.

• **mask** (list of ndarray, optional) – Boolean array mask for each Peaks object in the PeaksSuite to control which points will be plotted, default is None so no mask is applied.

**Returns** None or tuple – None if ax is provided, otherwise tuple of the form (fig, ax) where fig is the figure handle and ax is the axes handle.

**static plot_resolution_limits** (ax, xtype, ytype, attribute, limits, plot_kwargs=None)
Plot resolution limits on provided Axes.

**Parameters**

• **ax** (Axes) – Axes on which resolution limit is to be plotted.

• **xtype**, **ytype** ('frequency', 'wavelength') – Attribute on x-axis.

• **attribute** ('velocity', 'slowness', 'wavenumber') – Attribute on y-axis.

• **limits** (tuple) – Of the form (lower limit, upper limit).

• **plot_kwargs** (dict, optional) – Keyword arguments to pass along to ax.plot, default is None indicating the predefined settings should be used.

**Returns** None – Updates Axes with resolution limit (if possible).

**plot_statistics** (ax, xx, mean, stddev, errorbar_kwargs=None)

**reject_box_inside** (xtype, xlims, ytype, ylims)
Reject peaks inside the stated limits.

**Parameters**

• **xtype**, **ytype** ('frequency', 'velocity', 'slowness', 'wavelength') – Parameter domain in which the limits are defined.

• **xlims**, **ylims** (tuple) – Tuple with the lower and upper limits for each of the boundaries.

**Returns** None – Updates the PeaksSuite internal state.

**reject_limits_outside** (attribute, limits)
Reject peaks outside the stated limits.

**Parameters**

• **attr** ('frequency', 'velocity', 'slowness', 'wavelength') – Parameter domain in which the limits are defined.

• **limits** (tuple) – Tuple with the lower and upper limits. None may be used to perform one-sided rejections. For example limits=(None, 5) will reject all values above 5 and limits=(5, None) will reject all values below 5.

**Returns** None – Updates the PeaksSuite internal state.
statistics (xtype, ytype, xx, ignore_corr=True, drop_sample_if_fewer_count=3)

Determine the statistics of the PeaksSuite.

**Parameters**

- **xtype** ("frequency", "wavelength") – Axis along which to calculate statistics.
- **ytype** ("velocity", "slowness") – Axis along which to define uncertainty.
- **xx** (iterable) – Values in xtype units where statistics are to be calculated.
- **ignore_corr** (bool, optional) – Ignore calculation of data’s correlation coefficients, default is True.
- **drop_sample_if_fewer_count** (int, optional) – Remove statistic sample if the number of valid entries is fewer than the specified number, default is 3.

**Returns**

tuple – Of the form (xx, mean, std, corr) where mean and std are the mean and standard deviation at each point and corr are the correlation coefficients between every point and all other points.

to_array (xtype, ytype, xx)

Create an array representation of the PeaksSuite.

**Parameters**

- **xtype** ("frequency", "wavelength") – Axis along which to define samples.
- **ytype** ("velocity", "slowness") – Axis along which to define values.
- **xx** (iterable) – Values, in the units of xtype, where PeaksSuite is to be discretized.

**Returns**

tuple – Of the form (xx, array) where xx is the discretized values and array is a two-dimensional array with one row per Peaks in the PeaksSuite and one column for each entry of xx. Missing values are denoted with np.nan.

to_json (fname)

Write PeaksSuite to json file.

**Parameters**

- **fname** (str) – Name of the output file, may contain a relative or the full path.

**Returns**

None – Write json to disk.

1.2.7 regex

Regular expression definitions.

get_all (wavetype='rayleigh', time='(\d+\.?\d*)')

Compile regular expression to identify peaks from a .max file.

**Parameters**

- **time** (str, optional) – Define a specific time of interest, default is “(d+.?d*)”, a generic regular expression which will match all time.

**Returns**

Compiled Regular Expression – To identify peaks from a .max file.

get_nmaxima ()

get_peak_from_max (time=\d+\.|\d*, wavetype='rayleigh', frequency=\d+.?\d*[eE]([-+]?)\d*)

Compile regular expression to extract peaks from a .max file.
Parameters

• **wavetype** ([‘rayleigh’, ‘love’, ‘vertical’, ‘radial’, ‘transverse’], optional) – Define a specific wavetype to extract, default is ‘rayleigh’.

• **time** (str, optional) – Define a specific time of interest, default is “(d+.?d*)”), a generic regular expression which will match all time.

Returns  Compiled Regular Expression – To extract peaks from a .max file.

get_spac_ratio (time=’(-?\d+.?\d*[eE]?[+-]?\d*)’, component=’(0)’, ring=’(\d+)’)
TODO (jpv): Finish docstring.

Parameters

• **component** (’0’, ‘1’, ‘2’), optional) – Component vertical=”0”, radial=”1”, and transverse=”2” to be read, default is “0”.

• **ring** (str) – Desired ring, default is “d+” so all rings will be exported.

Returns  Compiled regular expression – To read lines from SPAC-style .max file.

get_spac_ring ()
Find all rings in MSPAC .log file. TODO (jpv): Finish docstring.

1.2.8 register

Registry class definition.

class AbstractRegistry
Bases: abc.ABC

classmethod create_class (name)
classmethod create_instance (name, *args, **kwargs)
classmethod register (name)
Returns the subclass, to allow usage as a class decorator.

class MaswWorkflowRegistry
Bases: swprocess.register.AbstractRegistry

class WavefieldTransformRegistry
Bases: swprocess.register.AbstractRegistry

1.2.9 sensor1c

Sensor1C class definition.

class Sensor1C (amplitude, dt, x, y, z, nstacks=1, delay=0)
Bases: swprocess.activetimeseries.ActiveTimeSeries
Class for single component sensor objects.

   __init__ (amplitude, dt, x, y, z, nstacks=1, delay=0)
   Initialize Sensor1C.

classmethod from_activetimeseries (activetimeseries, x, y, z)
classmethod from_sensor1c (sensor1c)
Create deep copy of an existing Sensor1C object.
classmethod from_trace

Create a Sensor1C object from a Trace object.

Parameters

- **trace** (*Trace*) – Trace object with attributes `data` and `stats.delta`.
- **read_header** (*bool*) – Flag to indicate whether the data in the header of the file should be parsed, default is `True` indicating that the header data will be read.
- **map_x, map_y** (*function, optional*) – Convert x and y coordinates using some function, default is not transformation. Can be useful for converting between coordinate systems.
- **nstacks** (*int, optional*) – Number of stacks included in the present trace, default is 1 (i.e., no stacking). Ignored if `read_header=True`.
- **delay** (*float, optional*) – Pre-trigger delay in seconds, default is 0 seconds. Ignored if `read_header=True`.
- **x, y, z** (*float, optional*) – Receiver’s relative position in x, y, and z, default is zero for all components (i.e., the origin). Ignored if `read_header=True`.

Returns **Sensor1C** – An initialized Sensor1C object.

Raises **ValueError** – If trace type cannot be identified.

property x

property y

property z

1.2.10 snr

SignaltoNoiseRatio class definition.

class SignaltoNoiseRatio (frequencies, snr)

Bases: object

classmethod from_array1ds

1.2.11 source

This file contains the Source class for storing information on the type and location of an active-source.

class Source (x, y, z)

Bases: object

A Source class for storing information about an active-source.

Attributes:

__init__(x, y, z)

Initialize a Source class object.

Args:

    position: Dictionary showing the relative position of the source from the first receiver of the form:
        {'x': xval, 'y':yval, 'z':zval}

Returns: This method returns no value.
**Raises:** This method raises no exceptions.

```python
classmethod from_source(other)
```

- **property x**
- **property y**
- **property z**

### 1.2.12 spacurve

### 1.2.13 spaccurvesuite

### 1.2.14 utils

Surface wave processing utilities.

```python
extract_mseed(startend_fname, network, data_dir='./', output_dir='./', extension='mseed')
```

Extract specific time blocks from a set of miniseed files.

Reads a large set of miniseed files, trims out specified time block(s), and writes the trimmed block(s) to disk. Useful for condensing a large dataset consisting of miniseed files written at the end of each hour to a single file that spans several hours. Stations which share an array name will appear in a common directory.

**Parameters**

- **startend_fname (str)** – Name of .csv file with start and end times. An example file is provided here.
- **network (str)** – Short string of characters to identify the network. Exported files will utilize this network code as its prefix.
- **data_dir (str, optional)** – The full or a relative file path to the directory containing the miniseed files, default is the current directory.
- **output_dir (str, optional)** – The full or a relative file path to the location to place the output miniseed files, default is the current directory.
- **extension ("mseed", "miniseed", optional)** – Extension used for miniSEED format, default is “mseed”.

**Returns** None – Writes folder and files to disk.

### 1.2.15 wavefieldtransforms

Wavefield transform class definitions.

```python
class AbstractWavefieldTransform(frequencies, velocities, power)
```

Bases: abc.ABC

Wavefield transformation of an Array1D.

```python
__init__(frequencies, velocities, power)
```

Define AbstractWavefieldTransform.

```python
find_peak_power(by='frequency-maximum')
```

Find maximum WavefieldTransform power.

**Parameters** by ("frequency-maximum"), optional – Determines how the maximum surface wave dispersion power is selected, default is ‘frequency-maximum’.
Returns \texttt{ndarray} – Containing the peak velocity at each frequency.

classmethod \texttt{from\_array}(\texttt{array}, \texttt{settings})

\texttt{normalize}(\texttt{by}='frequency-maximum')

Normalize WavefieldTransform power.

Parameters \texttt{by} \((\texttt{'none'}, \texttt{'absolute-maximum'}, \texttt{'frequency-maximum'}), \texttt{optional}) – Determines how the surface wave dispersion power is normalized, default is ‘frequency-maximum’.

Returns \texttt{None} – Update the internal state of power.

\texttt{plot}(\texttt{fig}=None, \texttt{ax}=None, \texttt{cax}=None, \texttt{normalization}='frequency-maximum', \texttt{peaks}='frequency-maximum', \texttt{nearfield}=None, \texttt{cmap}='jet', \texttt{peak\_kwargs}=None, \texttt{colorbar\_kwargs}=None, \texttt{rasterize}=False)

Plot the WavefieldTransform’s dispersion image.

Parameters

- \texttt{ax} (\texttt{Axes}, \texttt{optional}) – Axes object on which to plot the dispersion image, default is \texttt{None} so an \texttt{Axes} will be created on-the-fly.

- \texttt{cax} (\texttt{Axes}, \texttt{optional}) – Axes object on which to plot the colorbar for the dispersion image, default is \texttt{None} so an \texttt{Axes} will be created from \texttt{ax}.

- \texttt{normalization} \((\texttt{'none'}, \texttt{'absolute-maximum'}, \texttt{'frequency-maximum'}), \texttt{optional}) – Determines how the surface wave dispersion power is normalized, default is ‘frequency-maximum’.

- \texttt{peaks} \((\texttt{'none'}, \texttt{'frequency-maximum'}), \texttt{optional}) – Determines if the spectral peaks are shown and if so how they will be determined, default is ‘frequency-maximum’.

- \texttt{nearfield} (\texttt{int}, \texttt{optional}) – Number of array center distances per wavelength following Yoon and Rix (2009), default is \texttt{None} so nearfield criteria will not be plotted. A value of 1 corresponds to \(~15\%\) error and 2 \(~5\%\) error.

- \texttt{peak\_kwargs} (\texttt{dict}, \texttt{optional}) – Keyword arguments to control the appearance of the spectral peaks, default is \texttt{None} so the default settings will be used.

Returns \texttt{tuple} or \texttt{None} – \texttt{tuple} of the form (\texttt{fig}, \texttt{ax}) if \texttt{ax}=	exttt{None}, \texttt{None} otherwise.

\texttt{plot\_snr}(\texttt{ax}=	exttt{None}, \texttt{plot\_kwargs}=	exttt{None})

abstract classmethod \texttt{transform}()

A decorator indicating abstract classmethods.

Usage:

\begin{verbatim}
class C(metaclass=ABCMeta): @abstractclassmethod def my\_abstract\_classmethod(cls, \ldots):
    ...
\end{verbatim}

‘abstractclassmethod’ is deprecated. Use ‘classmethod’ with ‘abstractmethod’ instead.

class \texttt{EmptyWavefieldTransform}(\texttt{frequencies}, \texttt{velocities}, \texttt{power})

Bases: \texttt{swprocess.wavefieldtransforms.AbstractWavefieldTransform}

classmethod \texttt{from\_array}(\texttt{array}, \texttt{settings})

stack \((\texttt{other})$

\texttt{classmethod transform}(\texttt{array}, \texttt{velocities}, \texttt{settings})

Empty transform method.
**class** FDBF *(frequencies, velocities, power)*
  
  Bases: `swprocess.wavefieldtransforms.AbstractWavefieldTransform`

  **classmethod transform** *(array, velocities, settings)*
  
  Perform Frequency-Domain Beamforming.

  **Parameters**
  
  - **array** *(Array1D)* – Instance of Array1D.
  - **velocities** *(ndarray)* – Vector of trial velocities.
  - **settings** *(dict)* – dict with processing settings.

  **Returns** `tuple` – Of the form *(frequencies, power)*.

**class** FK *(frequencies, velocities, power)*
  
  Bases: `swprocess.wavefieldtransforms.FDBF`

  **classmethod from_array** *(array, settings)*

**class** PhaseShift *(frequencies, velocities, power)*
  
  Bases: `swprocess.wavefieldtransforms.AbstractWavefieldTransform`

  **classmethod transform** *(array, velocities, settings)*
  
  Perform the Phase-Shift Transform.

  **Parameters**
  
  - **array** *(Array1D)* – Instance of Array1D.
  - **velocities** *(ndarray)* – Vector of trial velocities.
  - **settings** *(dict)* – dict with processing settings.

  **Returns** `tuple` – Of the form *(frequencies, power)*.

**class** SlantStack *(frequencies, velocities, power)*
  
  Bases: `swprocess.wavefieldtransforms.AbstractWavefieldTransform`

  **classmethod slant_stack** *(array, velocities)*
  
  Perform a slant-stack on the given wavefield data.

  **Parameters**
  
  - **array** *(Array1d)* – One-dimensional array object.
  - **velocities** *(ndarray)* – One-dimensional array of trial velocities.

  **Returns** `tuple` – Of the form *(tau, slant_stack)* where *tau* is an ndarray of the attempted intercept times and *slant_stack* are the slant-stacked waveforms.

  **classmethod transform** *(array, velocities, settings)*
  
  Perform the Slant-Stack transform.

  **Parameters**
  
  - **array** *(Array1D)* – Instance of Array1D.
  - **velocities** *(ndarray)* – Vector of trial velocities.
  - **settings** *(dict)* – dict with processing settings.

  **Returns** `tuple` – Of the form *(frequencies, power)*.
1.3 License Information

Copyright (C) 2020 Joseph P. Vantassel (jvantassel@utexas.edu)

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.
INDICES AND TABLES

- genindex
- modindex
- search
PYTHON MODULE INDEX

S
  swprocess.activetimeseries, 2
  swprocess.array1d, 5
  swprocess.masw, 8
  swprocess.maswworkflows, 8
  swprocess.peaks, 9
  swprocess.peakssuite, 12
  swprocess.regex, 15
  swprocess.register, 16
  swprocess.sensor1c, 16
  swprocess.snr, 17
  swprocess.source, 17
  swprocess.spaccurve, 18
  swprocess.spaccurvesuite, 18
  swprocess.utils, 18
  swprocess.wavefieldtransforms, 18
Symbols

__init__() (AbstractMaswWorkflow method), 8
__init__() (AbstractWavefieldTransform method), 18
__init__() (ActiveTimeSeries method), 2
__init__() (Array1D method), 5
__init__() (Peaks method), 9
__init__() (PeaksSuite method), 12
__init__() (Sensor1C method), 16
__init__() (Source method), 17

A
AbstractMaswWorkflow (class in swprocess.maswworkflows), 8
AbstractRegistry (class in swprocess.register), 16
AbstractWavefieldTransform (class in swprocess.wavefieldtransforms), 18
ActiveTimeSeries (class in swprocess.activetimeseries), 2
append() (PeaksSuite method), 12
Array1D (class in swprocess.array1d), 5
array_center_distance() (Array1D property), 5
auto_pick_first_arrivals() (Array1D method), 5
axes_defaults (Peaks attribute), 10
azimuth() (Peaks property), 10

calc_resolution_limits() (PeaksSuite static method), 12
calculate_snr() (AbstractMaswWorkflow method), 8
check() (AbstractMaswWorkflow method), 8
create_class() (AbstractRegistry class method), 16
create_instance() (AbstractRegistry class method), 16
create_settings_dict() (Masw static method), 8
crosscorr() (ActiveTimeSeries static method), 2
crosscorr_shift() (ActiveTimeSeries static method), 3
delay() (ActiveTimeSeries property), 3
detrend() (AbstractMaswWorkflow method), 8
df() (ActiveTimeSeries property), 3

E
ellipticity() (Peaks property), 10
EmptyWavefieldTransform (class in swprocess.wavefieldtransforms), 19
extended_attrs() (Peaks property), 10
extract_mseed() (in module swprocess.utils), 18

F
FDBF (class in swprocess.wavefieldtransforms), 19
find_peak_power() (AbstractWavefieldTransform method), 18
FK (class in swprocess.wavefieldtransforms), 20
frequency() (Peaks property), 10
FrequencyDomainMaswWorkflow (class in swprocess.maswworkflows), 9
from_activetimeseries() (ActiveTimeSeries class method), 3
from_activetimeseries() (Sensor1C class method), 16
from_array() (AbstractWavefieldTransform class method), 19
from_array() (EmptyWavefieldTransform class method), 19
from_array() (FK class method), 20
from_array1d() (Array1D class method), 5
from_arraylds() (SignaltoNoiseRatio class method), 17
from_cross_stack() (ActiveTimeSeries class method), 3
from_dict() (Peaks class method), 10
from_dict() (PeaksSuite class method), 12
from_files() (Array1D class method), 5
from_json() (Peaks class method), 10
from_json() (PeaksSuite class method), 12
from_max() (Peaks class method), 10
from_max() (PeaksSuite class method), 13
from_peaks() (PeaksSuite class method), 13
from_peakssuite() (PeaksSuite class method), 13
from_sensor1c() (Sensor1C class method), 16
from_source() (Source class method), 18
from_trace() (ActiveTimeSeries class method), 3
from_trace() (Sensor1C class method), 17
from_trace_seg2() (ActiveTimeSeries class method), 3

G
get_all() (in module swprocess.regex), 15
get_nmaxima() (in module swprocess.regex), 15
get_peak_from_max() (in module swprocess.regex), 15
get_spac_ratio() (in module swprocess.regex), 16
get_spac_ring() (in module swprocess.regex), 16

I
interactive_mute() (Array1D method), 6
interactive_trimming() (PeaksSuite method), 13
is_similar() (Array1D method), 6

K
kres() (Array1D property), 6

M
manual_pick_first_arrivals () (Array1D method), 6
Masw (class in swprocess.masw), 8
MaswWorkflowRegistry (class in swprocess.register), 16
module
    swprocess.activetimeseries, 2
    swprocess.array1d, 5
    swprocess.masw, 8
    swprocess.maswworkflows, 8
    swprocess.peaks, 9
    swprocess.peakssuite, 12
    swprocess.regex, 15
    swprocess.register, 16
    swprocess.sensor1c, 16
    swprocess.snr, 17
    swprocess.source, 17
    swprocess.spaccurve, 18
    swprocess.spaccurvesuite, 18
    swprocess.utils, 18
    swprocess.wavefieldtransforms, 18
multiple() (ActiveTimeSeries property), 4
mute() (AbstractMaswWorkflow method), 9
mute() (Array1D method), 6

N
n_stacks() (ActiveTimeSeries property), 4
nchannels() (Array1D property), 6
noise() (Peaks property), 11
normalize() (AbstractWavefieldTransform method), 19
nstacks() (ActiveTimeSeries property), 4
offsets() (Array1D property), 6

O

P
pad() (AbstractMaswWorkflow method), 9
Peaks (class in swprocess.peaks), 9
PeaksSuite (class in swprocess.peakssuite), 12
PhaseShift (class in swprocess.wavefieldtransforms), 20
plot() (AbstractWavefieldTransform method), 19
plot() (Array1D method), 6
plot() (Peaks method), 11
plot() (PeaksSuite method), 13
plot_resolution_limits() (PeaksSuite static method), 14
plot_snr() (AbstractWavefieldTransform method), 19
plot_statistics() (PeaksSuite method), 14
position() (Array1D method), 6
power() (Peaks property), 11

R
register() (AbstractRegistry class method), 16
reject_box_inside() (Peaks method), 11
reject_box_inside() (PeaksSuite method), 14
reject_limits_outside() (Peaks method), 11
reject_limits_outside() (PeaksSuite method), 14
run() (AbstractMaswWorkflow method), 9
run() (FrequencyDomainMaswWorkflow method), 9
run() (Masw static method), 8
run() (SingleMaswWorkflow method), 9
run() (TimeDomainWorkflow method), 9

S
select_noise() (AbstractMaswWorkflow method), 9
select_signal() (AbstractMaswWorkflow method), 9
Sensor1C (class in swprocess.sensor1c), 16
SignaltoNoiseRatio (class in swprocess.snr), 17
simplify_mpeaks() (Peaks method), 12
SingleMaswWorkflow (class in swprocess.maswworkflows), 9
slant_stack() (SlantStack class method), 20
SlantStack (class in swprocess.wavefieldtransforms), 20
slowness() (Peaks property), 12
Source (class in swprocess.source), 17
spacing() (Array1D property), 6
stack() (EmptyWavefieldTransform method), 19
stack_append() (ActiveTimeSeries method), 19
statistics() (PeaksSuite method), 14
<table>
<thead>
<tr>
<th>Module</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>swprocess.activetimeseries</td>
<td>2</td>
</tr>
<tr>
<td>swprocess.array1d</td>
<td>5</td>
</tr>
<tr>
<td>swprocess.masw</td>
<td>8</td>
</tr>
<tr>
<td>swprocess.maswworkflows</td>
<td>8</td>
</tr>
<tr>
<td>swprocess.peaks</td>
<td>9</td>
</tr>
<tr>
<td>swprocess.peakssuite</td>
<td>12</td>
</tr>
<tr>
<td>swprocess.regex</td>
<td>15</td>
</tr>
<tr>
<td>swprocess.register</td>
<td>16</td>
</tr>
<tr>
<td>swprocess.sensor1c</td>
<td>16</td>
</tr>
<tr>
<td>swprocess.snr</td>
<td>17</td>
</tr>
<tr>
<td>swprocess.source</td>
<td>17</td>
</tr>
<tr>
<td>swprocess.spaccurve</td>
<td>18</td>
</tr>
<tr>
<td>swprocess.spaccurvesuite</td>
<td>18</td>
</tr>
<tr>
<td>swprocess.utils</td>
<td>18</td>
</tr>
<tr>
<td>swprocess.wavefieldtransforms</td>
<td>18</td>
</tr>
</tbody>
</table>

### V
- velocity() *(Peaks property)*, 12

### W
- waterfall() *(Array1D method)*, 7
- WavefieldTransformRegistry *(class in swprocess.register)*, 16
- wavelength() *(Peaks property)*, 12
- wavenumber() *(Peaks property)*, 12

### X
- x() *(Sensor1C property)*, 17
- x() *(Source property)*, 18

### Y
- y() *(Sensor1C property)*, 17
- y() *(Source property)*, 18

### Z
- z() *(Sensor1C property)*, 17
- z() *(Source property)*, 18
- zero_pad() *(ActiveTimeSeries method)*, 4
- zero_pad() *(Array1D method)*, 7

### T
- time() *(ActiveTimeSeries property)*, 4
- TimeDomainMaswWorkflow *(class in swprocess.maswworkflows)*, 9
- TimeDomainWorkflow *(class in swprocess.maswworkflows)*, 9
- timeseriesmatrix() *(Array1D method)*, 6
- to_array() *(PeaksSuite method)*, 15
- to_file() *(Array1D method)*, 7
- to_json() *(Peaks method)*, 12
- to_json() *(PeaksSuite method)*, 15
- transform() *(AbstractWavefieldTransform class method)*, 19
- transform() *(EmptyWavefieldTransform class method)*, 19
- transform() *(FDBF class method)*, 20
- transform() *(PhaseShift class method)*, 20
- transform() *(SlantStack class method)*, 20
- trim() *(AbstractMaswWorkflow method)*, 9
- trim() *(ActiveTimeSeries method)*, 4
- trim() *(Array1D method)*, 7