# swspy

*Release 1.0.4*

Tom S Hudson

Sep 11, 2023

# CONTENTS:

SWSPy is a python package for performing shear wave splitting in an automated manner. Based on the eigenvalue method of Silver and Chan (1991), with multi-windowing of Teanby et al (2004), coordinate system transformations from Walsh et al (2013), automation methods based on Wustefeld et al (2010), support for multi-layered anisotropic media and some other additions.

CHAPTER

ONE

# INSTALLATION

swspy can currently be installed via PyPi (pip install swspy) or a manual install. For the most stable version, select PyPi, for the most up to date versionm select a manual install.

## 1.1 Dependencies

The package has the following dependencies:

numpy scipy pandas matplotlib scikit-learn obspy numba (And for testing: pytest nbformat nbconvert ipykernel)

## 1.2 Installing

### 1.2.1 Manual install from source

Download or clone the package from , and install by:

```
python setup.py install
```

### 1.2.2 pip install

To install via PyPi you can use the following command to install the package:

```
pip install swspy
```

### 1.2.3 conda install

We hope to soon support installation via conda.

CHAPTER

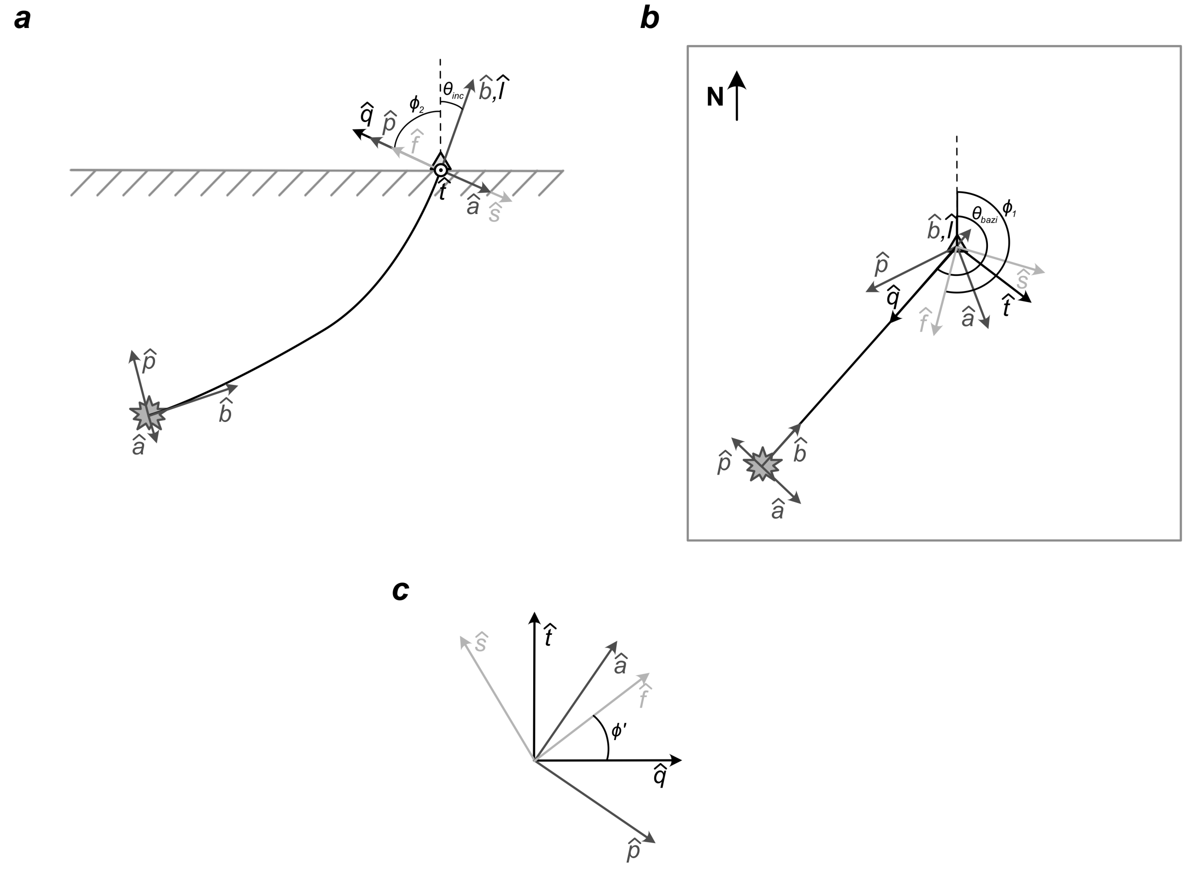
TWO

# COORDINATE SYSTEM

It is important to understand the coordinate system used by swspy, given that swspy can measure ansiotropy in 3D, as opposed to most previous shear-wave splitting codes.

There are two options that one can use for coordinate system: 1. ZNE - In this coordinate system, the package assumes that the ray is coming in at vertical incidence, therefore measuring fst-direction in the horizontal plane. This is conventionally how most previous codes have performed shear-wave splitting analysis. 2. LQT - In this coordinate system, the package works in the emerging coordinate system, i.e. in 3D. In this system, the fast direction is described by two angles, phi_1 and phi_2.

Below is a figure summarising the coordinate system. In both the above cases, the problem is solved in the QT plane then translated to the correct coordinate system for the outputs. For the ZNE coordinate system option, theta_inc = 0.



CHAPTER
THREE

# TUTORIALS

Included here are a number of tutorials showing examples of how to use swspy.

## 3.1 Example of single source-receiver measurement for ScS arrival, manually specifying input paramters

This example shows how the code can be applied to undertake shear-wave splitting analysis for an ScS arrival at one receiver.

Note: Here, event information such as S arrival time, ray back-azimuth and ray inclination are manually specified (see other examples for automatic passing of these parameters based on swspy or nonlinloc formats).

The data is from:

J Asplet, J Wookey, M Kendall. (2020) “A potential post-perovskite province in D beneath the Eastern Pacific: evidence from new analysis of discrepant SKS–SKKS shear-wave splitting”. GJI.

[1]:
```
%load_ext autoreload
%autoreload 2
```

[2]:
```
import swspy
import obspy
from obspy import UTCDateTime
import numpy as np
%matplotlib notebook
import matplotlib.pyplot as plt
import glob
import os, sys
import pandas as pd
```

### 3.1.1 Perform shear-wave splitting on event:

[3]:
```
# And get data for specific station:
station_to_analyse = "RDM"
event_uid = "RDM_2003174_121231_ScS"
S_wave_arrival_time = obspy.UTCDateTime("2003-06-23T12:31:20.000000Z")

# Manually specify key parameters:
stations_in = [station_to_analyse]
```

(continues on next page)

(continued from previous page)

```
S_phase_arrival_times = [S_wave_arrival_time]
back_azis_all_stations = [311.86] # Back-azimuth from North
receiver_inc_angles_all_stations = [0.0] # Inclination angle of ray at station from␣
↪vertical up (assume arriving with vertical incidence)
win_starttime = S_wave_arrival_time - 5
win_endtime = S_wave_arrival_time + 25

# Load data:
mseed_path = os.path.join("data","splittingsample","data", event_uid+".*")
downsample_factor = 1 # Factor to downsample data by (for faster slitting)
load_wfs_obj = swspy.io.load_waveforms(mseed_path, archive_vs_file="file", downsample_
↪factor=downsample_factor)
load_wfs_obj.filter = True
load_wfs_obj.filter_freq_min_max = [0.01, 0.5]
st = load_wfs_obj.read_waveform_data()

# Calculate splitting:
splitting_event = swspy.splitting.create_splitting_object(st, event_uid=event_uid,␣
↪stations_in=stations_in, S_phase_arrival_times=S_phase_arrival_times, back_azis_all_
↪stations=back_azis_all_stations, receiver_inc_angles_all_stations=receiver_inc_angles_
↪all_stations)
splitting_event.overall_win_start_pre_fast_S_pick = 4.0
splitting_event.win_S_pick_tolerance = 1.0
splitting_event.overall_win_start_post_fast_S_pick = (win_endtime - win_starttime) - 10
splitting_event.rotate_step_deg = 1.0
splitting_event.max_t_shift_s = 5.0
splitting_event.n_win = 10
splitting_event.perform_sws_analysis(coord_system="ZNE", sws_method="EV")

# And plot splitting result:
splitting_event.plot(outdir=os.path.join("outputs", "plots"))

# And save result to file:
splitting_event.save_result(outdir=os.path.join("outputs", "data"))
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
/Users/eart0504/Documents/python/github_repositories/swspy/swspy/splitting/split.py:1907:
↪ UserWarning: constrained_layout not applied.  At least one axes collapsed to zero␣
↪width or height.
  plt.savefig(os.path.join(outdir, ''.join((self.event_uid, "_", station, ".png"))),␣
↪dpi=300)
```

```
Saved sws result to: outputs/data/RDM_2003174_121231_ScS_sws_result.csv
```

```
[ ]:
```

## 3.2 Single earthquake example (for multiple receiver observations)

Example of shear-wave splitting applied to an icequake from Rutford Ice Stream, Antarctica.

The event is an icequake from Hudson et al. (2020).

Hudson, T.S., Brisbourne, A.M., Walter, F., Graff, D., & White, R. S. (2020). Icequake source mechanisms for studying glacial sliding. Journal of Geophysical Research: Earth Surface. https://doi.org/10.1029/2020JF005627

```
[1]: %load_ext autoreload
     %autoreload 2
```

```
[2]: import swspy
     import obspy
     from obspy import UTCDateTime
     import numpy as np
     %matplotlib notebook
     import matplotlib.pyplot as plt
     %matplotlib notebook
```

### 3.2.1 1. Load data for event:

Data can be from an archive of continuous seismic data, or from a mseed file containing only data for the event.

```
[3]: # Load data:
     archive_vs_file = "file"
     mseed_file_path = "data/20090121042009180_ice_flow_dir_corrected.m"
     nonlinloc_event_path = "data/loc.Tom__RunNLLoc000.20090121.042009.grid0.loc.hyp"
     starttime = UTCDateTime("20090121T042009.18523") - 0.5
     endtime = UTCDateTime("20090121T042009.18523") + 2.5
     load_wfs_obj = swspy.io.load_waveforms(mseed_file_path, starttime=starttime,␣
     ↪endtime=endtime, archive_vs_file=archive_vs_file)
     load_wfs_obj.filter = True
     load_wfs_obj.filter_freq_min_max = [1.0, 80.0]
     st = load_wfs_obj.read_waveform_data()
```

### 3.2.2 2. Calculate splitting:

Note: Here, event information such as S arrival time, ray back-azimuth and ray inclination are taken automatically from a nonlinloc event file (see other examples for manual passing of these parameters).

```
[4]: # Calculate splitting:
     splitting_event = swspy.splitting.create_splitting_object(st, nonlinloc_event_
     ↪path=nonlinloc_event_path) #(st, nonlinloc_event_path) #(st.select(station="ST01"),␣
     ↪nonlinloc_event_path)
     splitting_event.overall_win_start_pre_fast_S_pick = 0.3
     splitting_event.win_S_pick_tolerance = 0.1
     splitting_event.overall_win_start_post_fast_S_pick = 0.2
     splitting_event.rotate_step_deg = 2.0
     splitting_event.max_t_shift_s = 0.1
```

(continues on next page)

(continued from previous page)

```
splitting_event.n_win = 10
splitting_event.perform_sws_analysis(coord_system="ZNE", sws_method="EV")
```

```
No S phase pick for station: ST08 therefore skipping this station.
No S phase pick for station: ST06 therefore skipping this station.
No S phase pick for station: ST10 therefore skipping this station.
No S phase pick for station: ST09 therefore skipping this station.
No S phase pick for station: ST07 therefore skipping this station.
```

[4]:
```
  station  phi_from_Q  phi_from_N  phi_from_U  phi_err     dt  dt_err  \
0    ST05        26.0       74.57        90.0     11.0  0.004   0.003
0    ST04       -48.0       54.07        90.0      3.0  0.044   0.001
0    ST02        38.0       67.53        90.0      3.0  0.042   0.018
0    ST03       -82.0      -87.53        90.0      6.0  0.020   0.001
0    ST01       -60.0       48.74        90.0      5.0  0.048   0.002

   src_pol_from_N  src_pol_from_U  src_pol_from_N_err  src_pol_from_U_err  \
0      154.197600       89.946359           17.214782           10.537896
0      165.368197       90.444536            6.430684            3.923207
0      163.575889       90.504097            6.436617            2.740569
0       26.199288       95.881281           12.880766           23.807806
0      168.369942       88.213619            9.483751            6.103969

   Q_w  lambda2/lambda1 ratio  ray_back_azi  ray_inc
0  NaN              0.055499         48.57    141.8
0  NaN              0.033150        282.07    163.9
0  NaN              0.034949         29.53    160.6
0  NaN              0.068566        354.47    144.4
0  NaN              0.052323        108.74    156.5
```

### 3.2.3 3. Plot result:

[5]:
```
splitting_event.plot(outdir='plots')
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
/Users/eart0504/Documents/python/github_repositories/swspy/swspy/splitting/split.py:1907:
↪ UserWarning: constrained_layout not applied.  At least one axes collapsed to zero␣
↪width or height.
  plt.savefig(os.path.join(outdir, ''.join((self.event_uid, "_", station, ".png"))),␣
↪dpi=300)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
/Users/eart0504/Documents/python/github_repositories/swspy/swspy/splitting/split.py:1907:
↪ UserWarning: constrained_layout not applied.  At least one axes collapsed to zero␣
↪width or height.
  plt.savefig(os.path.join(outdir, ''.join((self.event_uid, "_", station, ".png"))),␣
↪dpi=300)
```

```
No S phase pick for station: ST08 therefore skipping this station.
Skipping waveform correction for station: ST08
No S phase pick for station: ST06 therefore skipping this station.
Skipping waveform correction for station: ST06
No S phase pick for station: ST10 therefore skipping this station.
Skipping waveform correction for station: ST10
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
/Users/eart0504/Documents/python/github_repositories/swspy/swspy/splitting/split.py:1907:
↪ UserWarning: constrained_layout not applied.  At least one axes collapsed to zero␣
↪width or height.
  plt.savefig(os.path.join(outdir, ''.join((self.event_uid, "_", station, ".png"))),␣
↪dpi=300)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
/Users/eart0504/Documents/python/github_repositories/swspy/swspy/splitting/split.py:1907:
↪ UserWarning: constrained_layout not applied.  At least one axes collapsed to zero␣
↪width or height.
  plt.savefig(os.path.join(outdir, ''.join((self.event_uid, "_", station, ".png"))),␣
↪dpi=300)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
/Users/eart0504/Documents/python/github_repositories/swspy/swspy/splitting/split.py:1907:
↪ UserWarning: constrained_layout not applied.  At least one axes collapsed to zero␣
↪width or height.
  plt.savefig(os.path.join(outdir, ''.join((self.event_uid, "_", station, ".png"))),␣
↪dpi=300)
```

```
No S phase pick for station: ST09 therefore skipping this station.
Skipping waveform correction for station: ST09
No S phase pick for station: ST07 therefore skipping this station.
Skipping waveform correction for station: ST07
```

### 3.2.4 4. Save result:

[6]:
```
# And save result to file:
splitting_event.save_result()
```

```
Saved sws result to: /Users/eart0504/Documents/python/github_repositories/swspy/docs/
↪source/tutorials/single_event_example/20090121042009_sws_result.csv
```

[ ]:
```

```

## 3.3 Example of how to run automated multi-event manager

This example shows how the code can be applied to undertake shear-wave splitting analysis for many events, with the data from the SKS example.

Here, the S-picks are defined as 10 s into the sac data for each event.

Data is from:

J Asplet, J Wookey, M Kendall. (2020) "A potential post-perovskite province in D beneath the Eastern Pacific: evidence from new analysis of discrepant SKS–SKKS shear-wave splitting". GJI.

[1]:
```
%load_ext autoreload
%autoreload 2
```

[2]:
```
# Import modules:
import swspy
import obspy
from obspy import UTCDateTime
import numpy as np
%matplotlib notebook
import matplotlib.pyplot as plt
import glob
import os, sys
import pandas as pd
```

### 3.3.1 Specify parameters for processing:

First one specifies all the relevent parameters for the splitting analysis. This changes various parameters from their default values to something more appropriate for a particular dataset.

Note: For a detailed description of each parameter/attribute, do help(proc_many_events()) or read the documentation.

[3]:
```
# Specify data management parameters:
datadir = "data"
outdir = "outputs"
```

[4]:
```
# Setup automation object and set key splitting data processing parameters:
sws_proc_obj = swspy.automate.proc_many_events()
sws_proc_obj.filter = True
sws_proc_obj.filter_freq_min_max = [0.01, 0.5]
sws_proc_obj.overall_win_start_pre_fast_S_pick = 4.0
sws_proc_obj.win_S_pick_tolerance = 1.0
sws_proc_obj.overall_win_start_post_fast_S_pick = 30.0
sws_proc_obj.rotate_step_deg = 1.0
sws_proc_obj.max_t_shift_s = 5.0
sws_proc_obj.n_win = 10
sws_proc_obj.downsample_factor = 1 #4
sws_proc_obj.coord_system = "ZNE"
sws_proc_obj.sws_method = "EV"
sws_proc_obj.output_plots = False
```

### 3.3.2 Run the processing for multiple events:

After specifying the various parameters, one can run the analysis over multiple events.

```
[5]: # And run events through automated processing:
S_pick_time_after_start_s = 10.0 # Time of S pick after start of SAC trace
sws_proc_obj.run_events_sws_fmt(datadir, outdir, S_pick_time_after_start_s=S_pick_time_
↪after_start_s)
```

```
Processing for event UID: 116A_2006360_122621_SKKS (1/11)
Saved sws result to: outputs/data/20061226125151_sws_result.csv
Saved sws wfs to: outputs/data/20061226125151_wfs_*.mseed
Processing for event UID: COR_2008321_170232_SKS (2/11)
Saved sws result to: outputs/data/20081116172713_sws_result.csv
Saved sws wfs to: outputs/data/20081116172713_wfs_*.mseed
Processing for event UID: DAN_2003174_121231_ScS (3/11)
Saved sws result to: outputs/data/20030623123100_sws_result.csv
Saved sws wfs to: outputs/data/20030623123100_wfs_*.mseed
Processing for event UID: FACU_2009297_144044_SKS (4/11)
Saved sws result to: outputs/data/20091024150503_sws_result.csv
Saved sws wfs to: outputs/data/20091024150503_wfs_*.mseed
Processing for event UID: HUMO_2008321_170232_SKS (5/11)
Saved sws result to: outputs/data/20081116172718_sws_result.csv
Saved sws wfs to: outputs/data/20081116172718_wfs_*.mseed
Processing for event UID: IRON_2009297_144044_SKS (6/11)
Saved sws result to: outputs/data/20091024150513_sws_result.csv
Saved sws wfs to: outputs/data/20091024150513_wfs_*.mseed
Processing for event UID: K20A_2009003_223342_SKKS (7/11)
Saved sws result to: outputs/data/20090103225943_sws_result.csv
Saved sws wfs to: outputs/data/20090103225943_wfs_*.mseed
Processing for event UID: L07A_2007256_094844_SKS (8/11)
Saved sws result to: outputs/data/20070913101323_sws_result.csv
Saved sws wfs to: outputs/data/20070913101323_wfs_*.mseed
Processing for event UID: L24A_2009003_194355_SKKS (9/11)
Saved sws result to: outputs/data/20090103201022_sws_result.csv
Saved sws wfs to: outputs/data/20090103201022_wfs_*.mseed
Processing for event UID: NE81_2006360_122621_SKKS (10/11)
Saved sws result to: outputs/data/20061226125216_sws_result.csv
Saved sws wfs to: outputs/data/20061226125216_wfs_*.mseed
Processing for event UID: RDM_2003174_121231_ScS (11/11)
Saved sws result to: outputs/data/20030623123110_sws_result.csv
Saved sws wfs to: outputs/data/20030623123110_wfs_*.mseed
Finished processing shear-wave splitting for data in: data
Data saved to: outputs
```

```
[ ]:
```

## 3.4 Example using SAC data

This example shows how the code can be applied to undertake shear-wave splitting analysis using SAC data format.

SAC data can be passed directly into SWSPy in isolation, so long as the SAC headers are populated with the neccessary information to perform the splitting measurement.

The relevent SAC headers that have to be populated are:

1. a - The arrival time of the phase to use (in secs after trace start time).
2. baz - The back-azimuth from receiver to event (in deg from N).

(Note: It assumes that rays arrive vertically, so doesn't currently read inclination for SAC data)

The data in this examples is from:

J Asplet, J Wookey, M Kendall. (2020) "A potential post-perovskite province in D beneath the Eastern Pacific: evidence from new analysis of discrepant SKS–SKKS shear-wave splitting". GJI.

[1]:
```
%load_ext autoreload
%autoreload 2
```

[2]:
```
import swspy
import obspy
import os
from obspy import UTCDateTime
import numpy as np
%matplotlib notebook
import matplotlib.pyplot as plt
```

### 3.4.1 Specify parameters for processing:

First one specifies all the relevent parameters for the splitting analysis. This changes various parameters from their default values to something more appropriate for a particular dataset.

Note: For a detailed description of each parameter/attribute, do help(proc_many_events()) or read the documentation.

[3]:
```
# Specify where data is:
sac_dir = "data"
outdir = "outputs"
```

[4]:
```
# Setup automation object and set key splitting data processing parameters:
sws_proc_obj = swspy.automate.proc_many_events()
sws_proc_obj.filter = True
sws_proc_obj.filter_freq_min_max = [0.01, 0.5]
sws_proc_obj.overall_win_start_pre_fast_S_pick = 4.0
sws_proc_obj.win_S_pick_tolerance = 1.0
sws_proc_obj.overall_win_start_post_fast_S_pick = 20.0
sws_proc_obj.rotate_step_deg = 1.0
sws_proc_obj.max_t_shift_s = 4.0
sws_proc_obj.n_win = 10
sws_proc_obj.downsample_factor = 1 #4
```

(continues on next page)

(continued from previous page)

```
sws_proc_obj.coord_system = "ZNE"
sws_proc_obj.sws_method = "EV"
sws_proc_obj.output_plots = True
```

### 3.4.2 Run the processing for event(s) in sac directory:

After specifying the various parameters, one can run the analysis over multiple events.

[6]:
```
sws_proc_obj.run_events_sac(sac_dir, outdir)
```

```
Processing for event: 0/1
Successfully retreived sac info.
Saved sws result to: outputs/data/NEE_2005036_122318_SKKS_sws_result.csv
Saved sws wfs to: outputs/data/NEE_2005036_122318_SKKS_wfs_*.mseed
```

```
/Users/eart0504/Documents/python/github_repositories/swspy/swspy/splitting/split.py:1907:
↪ UserWarning: constrained_layout not applied.  At least one axes collapsed to zero␣
↪width or height.
  plt.savefig(os.path.join(outdir, ''.join((self.event_uid, "_", station, ".png"))),␣
↪dpi=300)
```

```
Finished processing shear-wave splitting for data in: data
Data saved to: outputs
```

[ ]:
```

```

CHAPTER

FOUR

# TIPS FOR HPC USERS

Since SWSPy is parallelised, it may be beneficial to process data on High Performance Computing (HPC) infrastructure for large datasets. Here is a little information on the arcitecture of SWSPy and a brief example on how to structure a HPC job.

SWSPy is parallelised in an embarissingly parallel fashion for performing the phi-dt grid search. Each process is therefore independent during the grid search. However, the processes come together at the end of each process, and so the code should be treated as a shared memory model rather than a distributed memory model. Crucially, this means that for any job, one should only submit to a maximum of one node. SWSPy does not support parallelisation across mulitple nodes for one job.

A simple example of a possible submission script is shown below. This script is written for a SLURM submission management system with the Anaconda package manager installed.

```
# Setup SBATCH params (example only):
SBATCH --nodes=1 # NOTE: swspy will only run on a single node
SBATCH --ntasks-per-node=1
SBATCH --cpus-per-task=48  # NOTE: Make sure number of NUMBA threads specified in swspy!
SBATCH --mem=64000
SBATCH --time=12:00:00
SBATCH --job-name=swspy_run

# Load python environment:
module load Anaconda3
source activate $DATA/swspy_env #Path to anaconda environment with swspy and all␣
↪dependencies installed

# Run SWSPy:
# NOTE: Very important that the number of processors specified above (--cpus-per-task␣
↪parameter)
python swspy_run_script.py # Python script detailing specific commands for running swspy␣
↪(see examples in tutorials).
```

# CHAPTER FIVE

# SWSPY

## 5.1 swspy package

### 5.1.1 Subpackages

#### swspy.automate package

#### Submodules

#### swspy.automate.automation_manager module

**class** swspy.automate.automation_manager.**proc_many_events**

Bases: `object`

Class to process many events to calculate shear-wave splitting.

**Parameters**
**None.** –

**filter**
If True, then filters the data by <filter_freq_min_max>.

**Type**
bool (default = False)

**filter_freq_min_max**
Filter parameters for filtering the data.

**Type**
list of two floats (default = [1.0, 100.0])

**overall_win_start_pre_fast_S_pick**
Overall window start time in seconds before S pick.

**Type**
float (default = 0.1 s)

**overall_win_start_post_fast_S_pick**
[float (default = 0.2 s)] Overall window start time in seconds after S pick.

**win_S_pick_tolerance**
[float (default = 0.1 s)] Time before and after S pick to not allow windows to start within (in seconds). For example, start windows start at: S arrival time - (<overall_win_start_pre_fast_S_pick> +

<win_S_pick_tolerance>) And end times windows start at: S arrival time + <win_S_pick_tolerance> + <overall_win_start_post_fast_S_pick>

**rotate_step_deg**
[float (default = 2.0 degrees)] Rotation step size of phi in degrees for the grid search in phi-delay-time space.

**max_t_shift_s**
[float (default = 0.1 s)] The maximum time shift the data by in seconds.

**n_win**
[int (default = 10)] The number of window start and end times to pick. Currently implemented as constant window step sizes within the specified range, as defined by <overall_win_start_pre_fast_S_pick> amd <win_S_pick_tolerance>. Therefore, will calculate splitting for n_win^2 windows in total.

**downsample_factor**
[int (default = 1)] Factor by which to downsample the data, to speed up processing. If <downsample_factor> = 1, doens't apply downsampling.

**upsample_factor**
[int (default = 1)] Factor by which to upsample the data, to smooth waveforms for enhanced timeshift processing. Currently uses weighted average slopes interpolation method. If <upsample_factor> = 1, doens't apply upsampling.

**coord_system**
[str] Coordinate system to perform analysis in. Options are: LQT, ZNE. Will convert splitting angles back into coordinates relative to ZNE whatever system it performs the splitting within. Default = ZNE.

**sws_method**
[str] Method with which to calculate sws parameters. Options are: EV, EV_and_XC. EV - Eigenvalue method (as in Silver and Chan (1991), Teanby (2004), Walsh et al. (2013)). EV_and_XC - Same as EV, except also performs cross-correlation for automation approach, as in Wustefeld et al. (2010). Default is EV_and_XC.

**output_wfs**
[bool] If True, will save uncorrected and corrected waveforms to: <outdir>/<data>/<event_uid>_wfs_uncorr.png and <outdir>/<data>/<event_uid>_wfs_corr.png Default is True.

**output_plots**
[bool] If True, will save output plots to: <outdir>/<plots>/<event_uid>_<station>.png Default is False.

**suppress_direct_plotting**
[bool] If True, suppresses direct plotting so plots are only saved to file (and will only save plots to file if <output_plots=True>). Default = False

`run_events_from_nlloc()`

`run_events_sws_fmt()`

`run_events_from_nlloc`(*mseed_archive_dir*, *nlloc_dir*, *outdir*, *archive_vs_event_mseed='archive'*, *event_uids_nlloc_fname_pairs_df=None*, *event_prepad=1.0*, *event_postpad=30.0*, *nproc=1*)

Function to run many events through shear-wave splitting analysis using nonlinloc and mseed data (in archive format: <mseed_archive_dir>/<year>/ <julday>/...).

**Parameters**

- **mseed_archive_dir** (*str*) – Default: Path to mseed archive overall directory. Subdirectory paths should be in format: <mseed_archive_dir>/<year>/<julday>/yearjulday_station_channel.m. Alternative:

However, user can optionally specify to read cut mseed files for each event, if archive_vs_event_mseed = “event” (rather than archive_vs_event_mseed = “archive”). In this alternative case, <mseed_archive_dir> should be a directory containing a mseed file for each event. A list of event uids and corresponding nlloc hyp filenames then also needs to be specified, passed via the parameter <event_uids_nlloc_fname_pairs_df>.

- **nlloc_dir** (*str*) – Path to nlloc .grid0.loc.hyp output files corresponding to events that want to process.
- **outdir** (*str*) – Path to output directory to save data to. Saves results to: csv event summary file: <outdir>/<data>/event_uid.csv And if <output_plots> is specified, then will output plots to: png event station file: <outdir>/<data>/<event_uid>_<station>.png
- **archive_vs_event_mseed** (*str*) – Two options: 1. “archive” - Specifies that <mseed_archive_dir> points to an archive directory. 2. “event” - Specifies that <mseed_archive_dir> points to a directory containing

  mseed data named by each event uid individually.

  Note: If “event” is specified, then user must also specify <event_uids_nlloc_fname_pairs_df>. Default is “archive”.
- **event_uids_nlloc_fname_pairs_df** (*pandas DataFrame*) – Pandas DataFrame containing two columns: 1. event_uid. This corresponds to the event mseed files labelled and contained in

  <mseed_archive_dir>.

  2. **nlloc_fname. This corresponds to the path to the nlloc file corresponding to the** event with event_uid.

  Default is None.
- **event_prepad** (*float*) –
- **event_postpad** (*float*) –
- **nproc** (*int*) –

**Return type**
Data output to files in outdir, as specified above.

**run_events_sac**(*sac_dir*, *outdir*, *nproc=1*)

Function to run many events through shear-wave splitting analysis using sac data.

**Parameters**

- **sac_dir** (*str*) – Path to sac input data directory. sac files for multiple events can be stored together in this directory. There must be three files for each event, corresponding to Z,N,E components. Each must have a corresponding event unique ID, e.g.: event1.BHZ, event1.BHN, event1.BHE, event2.BHZ, event2.BHN, event2.BHE, …
- **outdir** (*str*) – Path to output directory to save data to. Saves results to: csv event summary file: <outdir>/<data>/event_uid.csv And if <output_plots> is specified, then will output plots to: png event station file: <outdir>/<data>/<event_uid>_<station>.png
- **event_prepad** (*float*) –
- **event_postpad** (*float*) –
- **nproc** (*int*) –

**Return type**
Data output to files in outdir, as specified above.

**run_events_sws_fmt**(*datadir*, *outdir*, *S_pick_time_after_start_s=10.0*)

Function to run many events through shear-wave splitting analysis using sws format and sac data.

——————————— sws format notes: ———————————

Data directories containing sac data must be formatted as follows: <datadir>/event_uid/.*?H*

Sac data for each event must be trimmed with <S_pick_time_after_start_s> seconds padding before the S pick and sufficient padding after. Additionally, sac data must have back azimuth information (and inclination info. if not using ZNE coordinate system). ———————————————————————————-

**Parameters**

- **datadir** (*str*) – The overall directory path containing all events with event unique IDs as each directory, with each directory containing SAC files for each component of each station. I.e. data follows format: <datadir>/event_uid/.*?H*
- **outdir** (*str*) – Path to output directory to save data to. Saves results to: csv event summary file: <outdir>/<data>/event_uid.csv And if <output_plots> is specified, then will output plots to: png event station file: <outdir>/<data>/<event_uid>_<station>.png
- **S_pick_time_after_start_s** (*float*) – Time, in seconds, of S pick after start of SAC trace. Default is 10.0 s.

### Module contents

Submodule for undertaking autmoated splitting analysis.

## swspy.io package

### Submodules

### swspy.io.load module

**class** swspy.io.load.**load_waveforms**(*path*, *starttime=None*, *endtime=None*, *archive_vs_file='archive'*, *downsample_factor=1*, *upsample_factor=1*, *sac=False*)

Bases: `object`

A class to load waveforms from file or an archive.

Notes: - Will currently only load archived data from the format year/jul_day/*station* - Does not currently remove instrument response

**Parameters**

- **path** (*str*) – The path to the overall data archive or file to load waveforms from.
- **archive_vs_file** (*str (default = "archive")*) – Describes what the parameter <path> is associated with. If archive, then <path> is to an archive. If file, then <path> is the path to a obspy readable file (e.g. mseed). Default is archive.

- `starttime` (*obspy UTCDateTime object*) – The starttime to cut the data by. Any filters are applied before cutting. If not supplied then will load all data from the supplied file path (archive_vs_file must = file for this to be valid).
- `endtime` (*obspy UTCDateTime object*) – The endtime to cut the data by. Any filters are applied before cutting. If not supplied then will load all data from the supplied file path (archive_vs_file must = file for this to be valid).

`filter`

If True, then filters the data by <filter_freq_min_max>.

**Type**
bool (default = False)

`filter_freq_min_max`

Filter parameters for filtering the data.

**Type**
list of two floats (default = [2.0, 20.0])

`zero_phase`

If True, applies zero phase filter (if <filter> = True).

**Type**
bool (default = True)

`remove_response`

If True, removes instrument response using the response file specified by <response_file>. (Note: Not currently implemented!)

**Type**
bool (default = False)

`response_file_path`

Path to response file used to remove instrument response.

**Type**
str (default = None)

`downsample_factor`

Factor by which to downsample the data, to speed up processing. If <downsample_factor> = 1, obviously doens't apply downsampling.

**Type**
int (default = 1)

`upsample_factor`

Factor by which to upsample the data, to smooth waveforms for enhanced timeshift processing. Currently uses weighted average slopes interpolation method. If <upsample_factor> = 1, doens't apply upsampling.

**Type**
int (default = 1)

`sac`

If passing sac data, will read in sac headers to output directly for doing the splitting analysis. Note that currently doesn't read a ray inclination angle, so sets to come in vertically. Explicitly, reads the following information from the sac headers: The arrival time of the phase to use (in secs after trace start time). Default header is <a>, but user can specify a different value (e.g. <t0>) by using the class attribute <sac_s_pick_hdr>. baz - The back-azimuth from receiver to event (in deg from N). (also reads station from the stream headers). If sac = True, then read_waveform_data() will output the dictionary sac_info as part of the class, with the following keys: event_id - id for the event, set from the sac input fname. stations - Receiver ids. s_arrival_times - S-wave arrival times in UTCDateTime fmt. bazs - Back-azimuths. incs - Ray inclination angles (all = 0 degrees from vertical).

**Type**
bool (default = False)

**`sac_s_pick_hdr`**

The sac header to use for the S pick arrival time. Value is float with units of seconds from start of trace.

**Type**
str (default = a)

**`read_waveform_data`**(*stations=[ ]*, *channels='*'*)

Read in the waveform data for the specified time.

**`read_waveform_data`**(*stations=None*, *channels='*'*, *event_uid='*'*)

Function to read waveform data. Filters if specified.

**Parameters**

- **`stations`** (*`list of strs (default = None)`*) – List of stations to import. If None then imports all available stations (default).
- **`channels`** (*`str (default = "*")`*) – List of channels to import. Default is to import all channels.
- **`Returns`** –
- **`st`** (*`obspy stream`*) – Obspy stream object containing all the data requested.

## swspy.io.read_nonlinloc module

`class swspy.io.read_nonlinloc.`**`read_hyp_file`**(*hyp_fname*)

Bases: `object`

Class to import all useful information from a NonLinLoc hyp file.

**`read`**()

Function to read the lines of the file and append to the class structure.

## Module contents

Submodule for dealing with input and output management

## swspy.splitting package

## Submodules

## swspy.splitting.forward_model module

`exception swspy.splitting.forward_model.`**`CustomError`**

Bases: `Exception`

`swspy.splitting.forward_model.`**`add_splitting`**(*ZNE_st*, *phi_from_N*, *dt*, *back_azi*, *event_inclin_angle_at_station*, *snr=None*)

Function to add splitting to waveforms in ZNE notation.

Parameters

**phi_from_N**
[float] Angle of fast direction, in degrees from N.

**dt**
[float] Delay time between fast and slow S-waves, in seconds.

**back_azi**
[float] Back-azimuth from source to receiver, in degrees from N.

**event_inclin_angle_at_station**
[float] Inclination of ray at station, in degrees from vertical up.

**phi_from_up**
[float] Angle of fast direction, in degrees from vertical up. Optional. Default = 0 degrees.

**snr**
[float] Signal-to-noise-ratio of output source-time function. Optional. If not None, then will add white gaussian noise to data.

`swspy.splitting.forward_model.`**`create_src_time_func`**(*dur*, *fs*, *src_pol_from_N=0*, *src_pol_from_up=0*, *src_freq=10.0*, *wavelet='ricker'*, *t_src=1.0*)

Function to create synthetic source time function.

Parameters

**fs**
[float] Sampling rate, in Hz.

**src_pol_from_N**
[float] Source polarisation from North, in degrees. Optional. Default = 0 degrees.

**src_pol_from_up**
[float] Source polarisation from vertical up, in degrees. Optional. Default = 0 degrees.

**src_freq**
[float] Dominant source frequency, in Hz. Optional. Default is 10 Hz.

**wavelet**
[str] Type of wavelet to use for the source-time function. Optional. Default is ricker. Other options not currently implemented.

**t_src**
[float] Time, in seconds, from the beginning of the trace when the source-time function peaks. Optional. Default = 1 s.

## swspy.splitting.split module

**exception** `swspy.splitting.split.`**`CustomError`**

Bases: `Exception`

`swspy.splitting.split.`**`calc_dof`**(*y*)

Finds the number of degrees of freedom using a noise trace, y. Uses definition as in Walsh2013. Note: Doesn't apply any form of smoothing filter (f(t,h)).

**class** `swspy.splitting.split.`**`create_splitting_object`**(*st*, *nonlinloc_event_path=None*, *event_uid=None*, *stations_in=[]*, *S_phase_arrival_times=[]*, *back_azis_all_stations=[]*, *receiver_inc_angles_all_stations=[]*)

Bases: `object`

Class to create splitting object to perform shear wave splitting on.

**Parameters**

- **sts** (*obspy Stream object*) – Obspy stream containing all the waveform data for all the stations and channels to perform the splitting on.
- **nonlinloc_event_path** (*str*) – Path to NonLinLoc .grid0.loc.hyp file for the event.

**overall_win_start_pre_fast_S_pick**

Overall window start time in seconds before S pick.

**Type**

float (default = 0.1 s)

**overall_win_start_post_fast_S_pick**

[float (default = 0.2 s)] Overall window start time in seconds after S pick.

**win_S_pick_tolerance**

[float (default = 0.1 s)] Time before and after S pick to not allow windows to start within (in seconds). For example, start windows start at: S arrival time - (<overall_win_start_pre_fast_S_pick> + <win_S_pick_tolerance>) And end times windows start at: S arrival time + <win_S_pick_tolerance> + <overall_win_start_post_fast_S_pick>

**rotate_step_deg**

[float (default = 2.0 degrees)] Rotation step size of phi in degrees for the grid search in phi-delay-time space.

**max_t_shift_s**

[float (default = 0.1 s)] The maximum time shift the data by in seconds.

**n_win**

[int (default = 10)] The number of window start and end times to pick. Currently implemented as constant window step sizes within the specified range, as defined by <overall_win_start_pre_fast_S_pick> amd <win_S_pick_tolerance>. Therefore, will calculate splitting for n_win^2 windows in total.

**`perform_sws_analysis : Function to perform shear-wave splitting analysis.`**

**`plot : Function to plot the shear-wave splitting results.`**

**`save_result : Function to save sws results to file.`**

**`save_wfs : Function to save uncorrected and corrected waveforms to file.`**

**perform_sws_analysis**(*coord_system='ZNE'*, *sws_method='EV'*, *return_clusters_data=True*, *num_threads=2*)

Function to perform splitting analysis. Works in LQT coordinate system as then performs shear-wave-splitting in 3D.

**Parameters**

- **coord_system** (*str*) – Coordinate system to perform analysis in. Options are: LQT, ZNE. Will convert splitting angles back into coordinates relative to ZNE whatever system it performs the splitting within. Default = ZNE.
- **sws_method** (*str*) – Method with which to calculate sws parameters. Options are: EV, EV_and_XC. EV - Eigenvalue method (as in Silver and Chan (1991), Teanby (2004), Walsh et al. (2013)). EV_and_XC - Same as EV, except also performs cross-correlation for automation approach, as in Wustefeld et al. (2010). Default is EV.
- **return_clusters_data** (*bool*) – If True, returns clustering data information. This is primarily used for plotting. Default is False.
- **num_threads** (*int*) – Number of threads to use for parallel computing. Default is to use all available threads on the system.

**Returns**
**self.sws_result_df** – A pandas DataFrame containing the key splitting results.

**Return type**
pandas DataFrame

**perform_sws_analysis_multi_layer**(*coord_system='ZNE'*, *multi_layer_method='explicit'*, *num_threads=2*)

**Function to perform splitting analysis for a multi-layered medium. Currently**
only a 2-layer medium is supported. Works in LQT coordinate system, therefore

supporting shear-wave-splitting in 3D. Currently doesn't support any method other than <sws_method> = EV and doesn't support returning clustered data. Method assumes that apparent delay-time is longer than fast S-wave arrival duration.

**Parameters**

- **coord_system** (*str*) – Coordinate system to perform analysis in. Options are: LQT, ZNE. Will convert splitting angles back into coordinates relative to ZNE whatever system it performs the splitting within. Default = ZNE.
- **multi_layer_method** (*str*) – Multi-layer method algorithm to apply. Two options are: 1. explicit - Applies layers individually, one at a time. Efficient and far fewer free parameters, as computation scales as (n_phi * n_dt) * n-layers. 2. direct - Applies layers directly together in the inversion. Computationally expensive relative to explicit method, with many free parameters, as computation scales as (n_phi * n_dt) ^ n-layers.
- **num_threads** (*int*) – Number of threads to use for parallel computing. Default is to use all available threads on the system.

**Returns**

- **self.sws_result_df** (*pandas DataFrame*) – A pandas DataFrame containing the key splitting results for an apparent splitting measurement (i.e. assuming only one layer).
- *self.sws_multi_layer_result_df pandas DataFrame* – A pandas DataFrame containing the key splitting results for hte multi-layer result.

`plot`(*outdir=None*, *suppress_direct_plotting=False*)

Function to perform plotting…

`save_result`(*outdir='/home/docs/checkouts/readthedocs.org/user_builds/swspy/checkouts/latest/docs/source'*)

Function to save output. Output is a csv file with all the splitting data for the event, for all stations. Saves result as <event_uid>, to <outdir>.

`save_wfs`(*outdir='/home/docs/checkouts/readthedocs.org/user_builds/swspy/checkouts/latest/docs/source'*)

Function to save waveforms outputs. Outputs are unccorrected and corrected waveforms for all events. Saves result as <event_uid>.mseed, to <outdir>.

`swspy.splitting.split.direct_multi_layer_inv_fun`(*params*, *theta_app*, *alpha_app*)

Function combining eqns 1-3 from Ozalaybey and Savage (1994). Note: All angles in radians.

`swspy.splitting.split.find_nearest`(*array*, *value*)

`swspy.splitting.split.find_nearest_2D`(*array*, *value*)

`swspy.splitting.split.ftest`(*data*, *dof*, *alpha=0.05*, *k=2*, *min_max='min'*)

Finds the confidence bounds value associated with data. Note that this version uses the minumum of the data by default. :param data: Data to process. :type data: np array :param dof: Number of degrees of freedom. :type dof: int :param alpha: Confidence level (e.g. if alpha = 0.05, then 95pc confidence level found). :type alpha: float :param k: Number of parameters (e.g. phi, dt). :type k: int :param min_max: Whether performs ftest on min or max of data. :type min_max: specific str

**Returns**
**conf_bound** – Value of the confidence bounds for the specified confidence level, alpha.

**Return type**
float

`swspy.splitting.split.remove_splitting`(*st_ZNE_uncorr*, *phi*, *dt*, *back_azi*, *event_inclin_angle_at_station*, *return_BPA=False*, *src_pol=0.0*, *return_FS=True*)

Function to remove SWS from ZNE data for a single station. Note: Consistency in this function with sws measurement. Uses T in x-direction and Q in y direction convention.

**Parameters**

- **st_LQT_uncorr** (*obspy stream object*) – Stream data for station corresponding to splitting parameters.
- **phi** (*float*) – Splitting angle, for LQT coordinates.
- **dt** (*float*) – Fast-slow delay time (lag) for splitting.
- **back_azi** (*float*) – Back azimuth angle from reciever to event in degrees from North.
- **event_inclin_angle_at_station** (*float*) – Inclination angle of arrival at receiver, in degrees from vertical down.
- **return_BPA** (*bool*) – If True, will return obspy stream with Z,N,E and B,P,A channels (as in Walsh (2013)). Optional. Default = False.
- **src_pol** (*float*) – If <return_BPA> = True, then uses src_pol to calculate the polarisiation and null (P,A) vectors. Units are degrees clockwise from North.
- **return_FS** (*bool*) – If True, will return obspy stream with F (fast) and S (slow) channels also included. Optional. Default = True.

**Returns**
**st_ZNE_corr** – Corrected data, in ZNE coordinates (unless <return_BPA> = True, then will also output BPA channels too).

**Return type**
obspy stream object

#### Module contents

Submodule for undertaking splitting analysis.

### 5.1.2 Submodules

### 5.1.3 swspy.testing module

`swspy.testing.`**`assert_angle_allclose`**(*actual*, *desired*, ***kwargs*)

Raises an AssertionError if two objects are not equal accounting for angle

This wraps numpy.testing.assert_allclose but values such as 0.0, 360.0 and 180.0 are all treated as being the same. Keyword arguments are passed on.

`swspy.testing.`**`clamp_angle`**(*angle_in*)

Assuming 180 degree periodicity, represent an angle between -90 and +90 degrees

angle is a numpy array.

`swspy.testing.`**`periodic_angular_difference`**(*a_angle*, *b_angle*)

Angular difference assuming 180 degree periodicity

a_angle and b_angle are numpy arrays of angle, retuns a numpy array of differences such that 0.0 and 360.0 and 180.0 are all the same.

### 5.1.4 Module contents

# CHAPTER SIX

# CONTRIBUTING

If you want to contribute to improving this package then you are most welcome. There are various ways to contribute:

1. If you find a bug/error you can raise an issue within the SWSPy github repository.
2. If you have implemented something new that would improve the package or fix an issue, then you can fork the repository, checkout a new branch, make your changes, and submit a pull request for formal approval.

If possible, please try to raise code related issues through the github repository. However, if you are unclear on any science aspects, then please feel free to email one of the core developers.

CHAPTER

# SEVEN

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

# INDEX