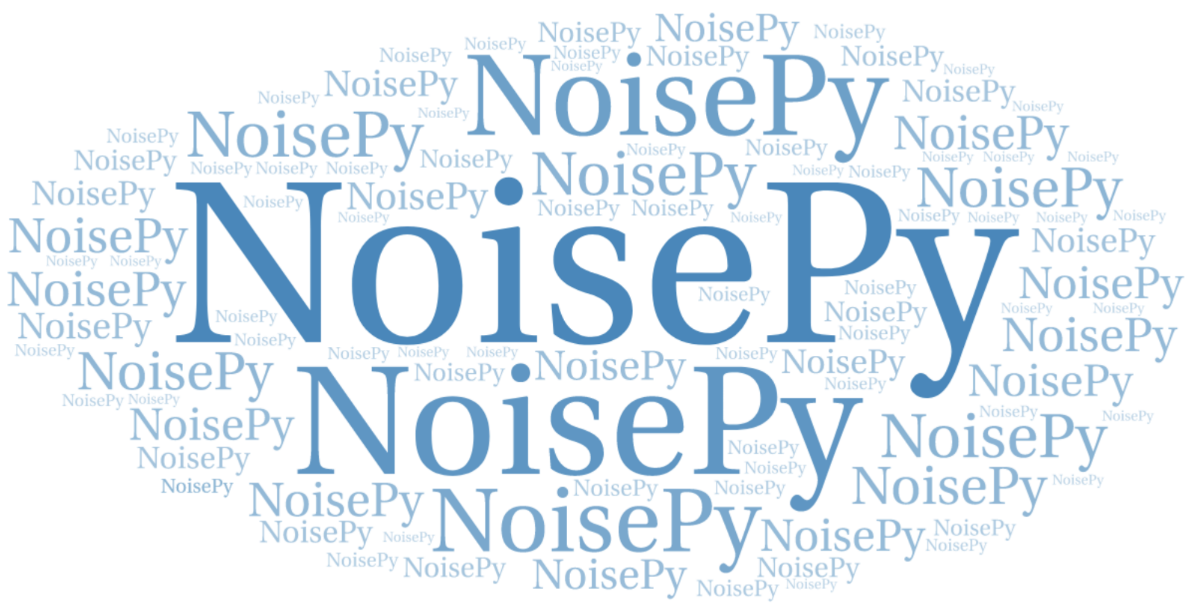
NoisePy Documentation

Release v1.0

Chengxin Jiang

Aug 25, 2020

Contents



This is the documentation for the Python package of **NoisePy**, which is a new high-performance python tool for seismic ambient noise seismology. For further information and contact information please see below website:

- Github repository of *NoisePy*: https://github.com/mdenolle/NoisePy

If you use NoisePy for your research and prepare publications, please consider citing **NoisePy**:

- Jiang, C., Yuan, C., and Denolle, M. NoisePy: a new high-performance python tool for seismic ambient noise seismology. In prep for Seismological Research Letter.

We gratefully acknowledge support from the Packard Fundation (www.packard.org).

CHAPTER 1

Functionality

- Download continous noise data based on obspy's core functions of get_station and get_waveforms
- Save seismic data in ASDF format, which convinently assembles meta, wavefrom and auxililary data into one single file (Turtorials on reading/writing ASDF files)
- Offers high flexibility to handle messy SAC/miniSEED data stored on your local machine and convert them into ASDF format data that could easily be pluged into NoisePy
- Performs fast and easy cross-correlation with functionality to run in parallel through MPI
- Includes a series of monitoring functions to measure dv/v on the resulted cross-correlation functions using some recently developed new methods (see our papers for more details)

1.1 Installation

1.1.1 NoisePy and Dependencies

The nature of NoisePy being composed of python scripts allows flexiable package installation. What you need to do is essentially build dependented libraries the scripts and related functions live upon.

`NoisePy` supports Python version 3.5, 3.6, and 3.7 and it depends on the following Python modules: `NumPy`, `ObsPy`, `pyasdf`, `mpi4py`, `numba`, `pycwt`. We recommand to use conda and pip to install the library due to their convinence. Below are command lines we have tested that would create a python environment to run NoisePy.

```
$ conda create -n noisepy -c conda-forge python=3.7.3 numpy=1.16.2 numba pandas pycwt␣
↪mpi4py=3.0.1
$ conda activate noisepy
$ pip install obspy pyasdf
```

Note: Please note that the test is performed on *macOS Mojave (10.14.5)*, so it could be slightly different for other OS.

1.1.2 Testing

To assert that your installation is working properly, execute

```
$ python S0_download_ASDF.py
$ python S1_fft_cc.py
$ python S2_stacking.py
```

and make sure the scripts all pass successfully. Otherwise please report issues on the github page or contact the developers.

Github repository of *NoisePy* can be found here: https://github.com/mdenolle/NoisePy

1.2 Tutorial

1.2.1 S0A. Downloading seismic noise data

The script of *S0_download_ASDF_MPI.py* (located in *src* directory) and its existing parameters allows to download all available broadband CI stations *(BH?)* located in a certain region and operated during 1/Jul/2016-2/Jul/2016 through the SCEC data center. In the script, short summary is provided for all input parameters that can be changed according to the user's needs. In the current form of the script, we set *inc_hours=24* to download day-long continous noise data as well as the meta info and store them into a single ASDF file. To increase the signal-to-noise (SNR) of the final cross-correlation functions (see Seats et al.,2012 for more details), we break the day-long sequence into smaller segments, each of *cc_len* (s) long with some overlapping defined by *step*. You may wanto to set *flag* to be *True* if intermediate outputs/operational time is preferred during the downloading process. To run the code on a single core, open the terminal and activate the noisepy environment before run following command. (NOTE that things may go completely different if you want to run NoisePy on a cluster. Better check it out first!!)

```
$ python S0_download_ASDF.py
```

If you want to use multiple cores (e.g, 4), run the script with the following command using mpi4py.

```
$ mpirun -n 4 python S0_download_ASDF_MPI.py
```

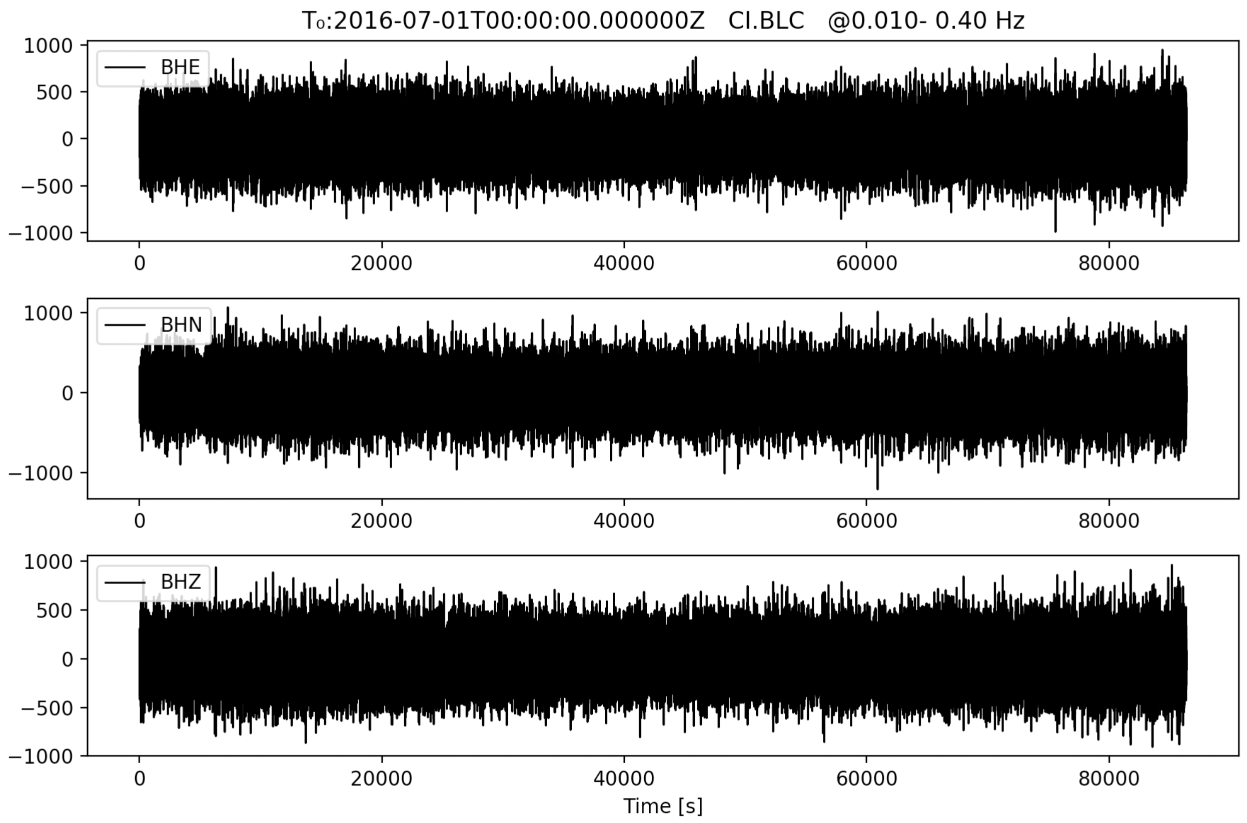
The outputted files from S0A include ASDF files containing daily-long (24h) continous noise data, a parameter file recording all used parameters in the script of S0A and a CSV file of all station information (more details on reading the ASDF files with downloaded data can be found in docs/src/ASDF.md). The continous waveforms data stored in the ASDF file can be displayed using the plotting modules named as *plotting_modules* in the directory of *src* as shown below.

```
>>> import plotting_modules (cd to your source file directory first before loading␣
↪this module)
>>> sfile = '/Users/chengxin/Documents/SCAL/RAW_DATA/2016_07_01_00_00_00T2016_07_02_
↪00_00_00.h5'
```

(continues on next page)

(continued from previous page)

```
>>> plotting_modules.plot_waveform(sfile,'CI','BLC',0.01,0.4)
```



Note: Please note that the script also offers the option to download data from an existing station list in a format same to the outputed CSV file. In this case, *down_list* should be set to *True* at L53. In reality, the downloading speed is dependent on many factors such as the original sampling rate of targeted data, the networks, the data center where it is hosted and the general structure you want to store on your machine etc. We tested a bunch of the parameters to evaluate their performance and the readers are referred to our paper for more details (Jiang et al., 2019).

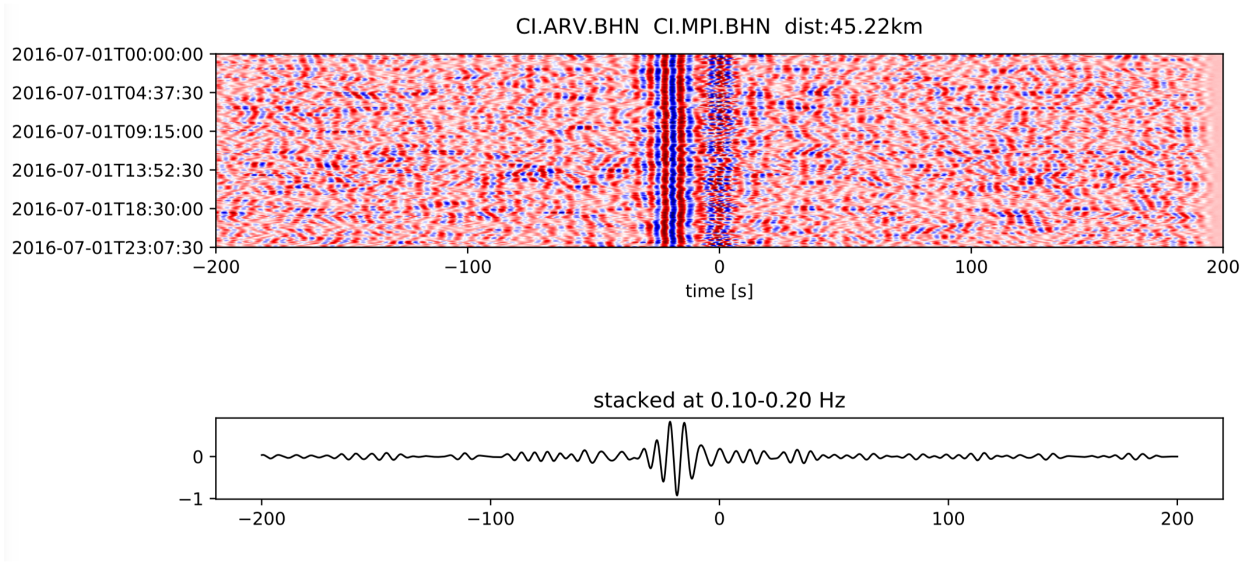
1.2.2 S0B. Deal with local SAC/miniseed data

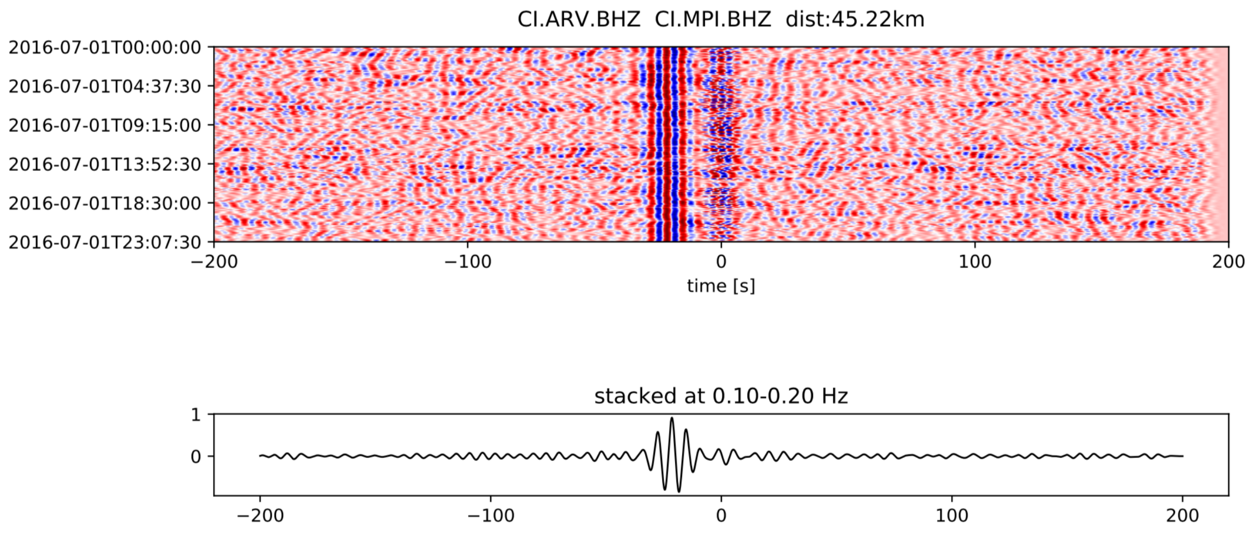
The script of *S0B_sacMSEED_to_ASDF.py* is developed for the users to handle local data in SAC/miniseed format stored on your own disk. Most of the variables are the same as those for S0A and thus should be pretty straighforward to follow and change. In this script, it preprocesses the data by merging, detrending, demeaning, downsampling and then trimming before saving them into ASDF format for later NoisePy processing. In particular, we expect the script to deal with very messydata, by which we mean that, seismic data is broken into small pieces and of messy time info such as overlapping time. REMEMBER to set *messydata* at L62 to *True* when you have messy data! (Tutorials on removing instrument response)

1.2.3 S1. Perform cross correlations

S1_fft_cc_MPI.py is the core script of NoisePy, which performs Fourier transform to all noise data first and loads them into the memory before they are further cross-correlated. This means that we are performing cross-correlation in the frequency domain. In the script, we provide several options to calculate the cross correlation, including *raw*, *coherency* and *deconv* (see our paper for detailed definition). We choose *coherency* as an example here. After running the script, it will create a new folder named *CCF*, in which new ASDF files containing all cross-correlation functions between different station pairs are located. It also creates a parameter file of *fft_cc_data.txt* that records all useful parameters used in this script. Once you get the cross-correlation file, you can show the daily temporal variation between all station-pair by calling *plot_substack_cc* function in *plotting_modules* as follows.

```
>>> import plotting_modules
>>> sfile = '/Users/chengxin/Documents/SCAL/CCF/2016_07_01_00_00_00T2016_07_02_00_00_
↪00.h5'
>>> plot_modules.plot_substack_cc(sfile,0.1,0.2,200,True,'/Users/chengxin/Documents/
↪SCAL/CCF/figures')
```

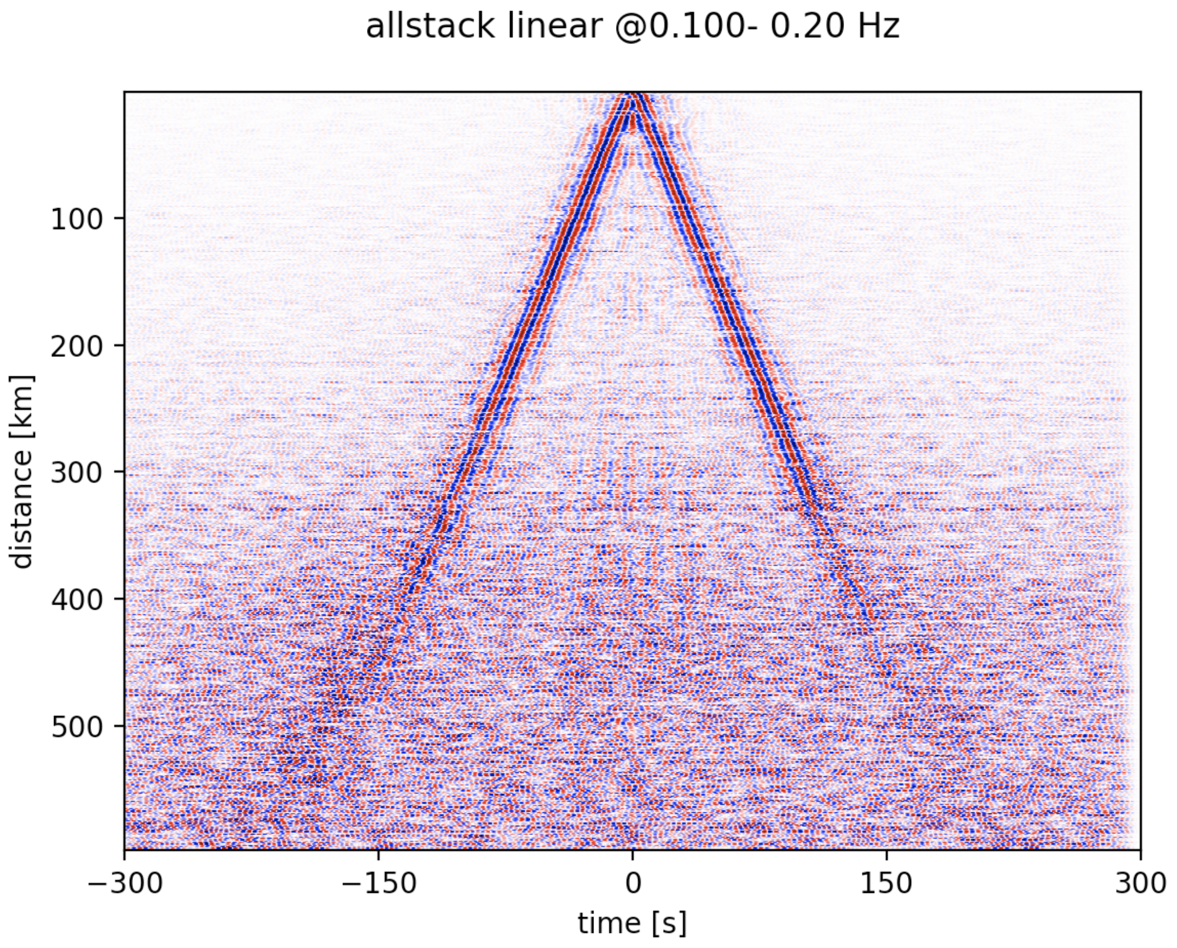


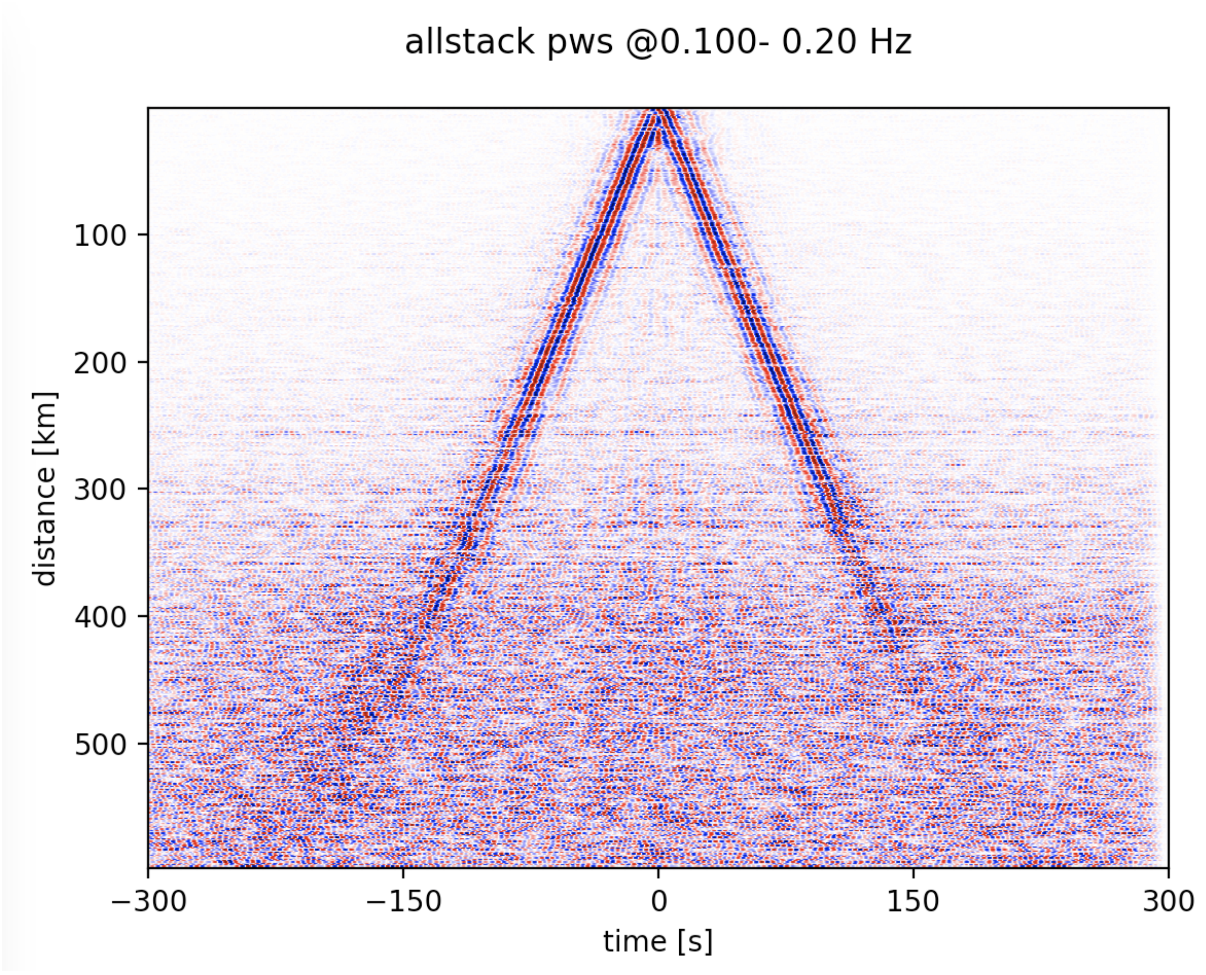


1.2.4 S2. Do stacking

The script of *S2_stacking.py* is used to assemble and/or stack all cross-correlation functions computed for the staion pairs in S1 and save them into ASDF files for future analysis (e.g., temporal variation and/or dispersion extraction). In particular, there are two options for the stacking process, including linear and phase weighted stacking (pws). In general, the pws produces waveforms with high SNR, and the snapshot below shows the waveform comparison from the two stacking methods. We use the folloing commend lines to make the move-out plot.

```
>>> import plotting_modules,glob
>>> sfiles = glob.glob('/Users/chengxin/Documents/SCAL/STACK/*/*.h5')
>>> plot_modules.plot_all_moveout(sfiles,'Allstack_linear'0.1,0.2,'ZZ',1,300,True,'/
↪Users/chengxin/Documents/SCAL/STACK') #(move-out for linear stacking)
>>> plot_modules.plot_all_moveout(sfiles,'Allstack_pws'0.1,0.2,'ZZ',1,300,True,'/
↪Users/chengxin/Documents/SCAL/STACK')     #(move-out for pws)
```





1.3 Pyasdf examples

The pyasdf format is developed by the **Theoretical and Computation Seismology Group** at Princeton University, and combines the capability to create comprehensive data sets including all necessary meta information with high-performance parallel I/O for the most demanding use cases. The users who are interested in the details of this format are referred to the following publication.

- Krischer, L., Smith, J., Lei, W., Lefebvre, M., Ruan, Y., de Andrade, E.S., Podhorszki, N., Bozdağ, E. and Tromp, J., 2016. An adaptable seismic data format. Geophysical Supplements to the Monthly Notices of the Royal Astronomical Society, 207(2), 1003-1011.

To better show the pyasdf format, we use the default examples downloaded from the pyasdf Github repository https://github.com/SeismicData/pyasdf for creating, processing and writing `pyasdf` format data.

- *Creating an ASDF File*
- *Processing Observed Data in Parallel*
- *Running pyflex in Parallel*

1.3.1 Creating an ASDF File

This example demonstrates how the create a new ASDF file from waveform data in any format ObsPy can read, a QuakeML file, and a list of StationXML files.

Note: Do **NOT** run this with MPI. This would require some modifications and is very likely not worth the effort.

```python
import glob
import os

from pyasdf import ASDFDataSet

filename = "observed.h5"

if os.path.exists(filename):
    raise Exception("File '%s' exists." % filename)

ds = ASDFDataSet(filename)

# Add event
ds.add_quakeml(
    "./GCMT_event_SOUTH_SANDWICH_ISLANDS_REGION_Mag_5.6_2010-3-11-6.xml"
)
event = ds.events[0]

# Add waveforms.
filenames = glob.glob("./SAC/*.SAC")
for _i, filename in enumerate(filenames):
    print("Adding SAC file %i of %i..." % (_i + 1, len(filenames)))
    # We associate the waveform with the previous event. This is optional
    # but recommended if the association is meaningful.
    ds.add_waveforms(filename, tag="raw_recording", event_id=event)

# Add StationXML files.
filenames = glob.glob("./StationXML/*.xml")
for _i, filename in enumerate(filenames):
    print("Adding StationXML file %i of %i..." % (_i + 1, len(filenames)))
    ds.add_stationxml(filename)
```

1.3.2 Processing Observed Data in Parallel

This fairly complex examples takes an ASDF file and produces two new data sets, each processed in a different frequency band.

It can be run with MPI. It scales fairly well and will utilize parallel I/O if your machine supports it. Please keep in mind that there is a significant start-up cost for Python on each core (special Python versions that get around that if really necessary are in existence) so don't use too many cores.

```
$ mpirun -n 64 python process_observed.py
```

If you don't run it with MPI with will utilize Python's `multiprocessing` module and run it on each of the machines cores. I/O is not parallel and uses a round-robin scheme where only one core writes at single point in time.

```
$ python process_observed.py
```

```python
import obspy
from obspy.core.util.geodetics import gps2DistAzimuth
import numpy as np

from pyasdf import ASDFDataSet

ds = ASDFDataSet("./observed.h5")

event = ds.events[0]

origin = event.preferred_origin() or event.origins[0]
event_latitude = origin.latitude
event_longitude = origin.longitude

# Figure out these parameters somehonw!
starttime = obspy.UTCDateTime("2010-03-11T06:22:19.021324Z")
npts = 5708
sampling_rate = 1.0


# Loop over both period sets. This will result in two files. It could also be
# saved to the same file.
for min_period, max_period in [(27.0, 60.0)]:
    f2 = 1.0 / max_period
    f3 = 1.0 / min_period
    f1 = 0.8 * f2
    f4 = 1.2 * f3
    pre_filt = (f1, f2, f3, f4)

    def process_function(st, inv):
        st.detrend("linear")
        st.detrend("demean")
        st.taper(max_percentage=0.05, type="hann")

        st.attach_response(inv)
        st.remove_response(
            output="DISP", pre_filt=pre_filt, zero_mean=False, taper=False
        )

        st.detrend("linear")
        st.detrend("demean")
        st.taper(max_percentage=0.05, type="hann")

        st.interpolate(
            sampling_rate=sampling_rate, starttime=starttime, npts=npts
        )

        station_latitude = inv[0][0].latitude
        station_longitude = inv[0][0].longitude
        _, baz, _ = gps2DistAzimuth(
            station_latitude,
            station_longitude,
            event_latitude,
            event_longitude,
        )

        components = [tr.stats.channel[-1] for tr in st]
```

(continues on next page)

(continued from previous page)

```
        if "N" in components and "E" in components:
            st.rotate(method="NE->RT", back_azimuth=baz)

        # Convert to single precision to save space.
        for tr in st:
            tr.data = np.require(tr.data, dtype="float32")

        return st

    tag_name = "preprocessed_%is_to_%is" % (int(min_period), int(max_period))

    tag_map = {"raw_recording": tag_name}

    ds.process(process_function, tag_name + ".h5", tag_map=tag_map)

# Important when running with MPI as it might otherwise not be able to finish.
del ds
```

1.3.3 Running pyflex in Parallel

`pyasdf` can be used to run a function across the data from two ASDF data sets. In most cases it will be some kind of misfit or comparision function. This example runs pyflex to pick windows given a data set of observed and another data set of synthetic data.

It can only be run with MPI:

```
$ mpirun -n 16 python parallel_pyflex.py
```

```
import pyflex
from pyasdf import ASDFDataSet

ds = ASDFDataSet("./preprocessed_27s_to_60s.h5")
other_ds = ASDFDataSet("./preprocessed_synthetic_27s_to_60s.h5")

event = ds.events[0]


def weight_function(win):
    return win.max_cc_value


config = pyflex.Config(
    min_period=27.0,
    max_period=60.0,
    stalta_waterlevel=0.11,
    tshift_acceptance_level=15.0,
    dlna_acceptance_level=2.5,
    cc_acceptance_level=0.6,
    c_0=0.7,
    c_1=2.0,
    c_2=0.0,
    c_3a=1.0,
    c_3b=2.0,
    c_4a=3.0,
    c_4b=10.0,
```

(continues on next page)

(continued from previous page)

```
    s2n_limit=0.5,
    max_time_before_first_arrival=-50.0,
    min_surface_wave_velocity=3.0,
    window_signal_to_noise_type="energy",
    window_weight_fct=weight_function,
)


def process(this_station_group, other_station_group):
    # Make sure everything thats required is there.
    if (
        not hasattr(this_station_group, "StationXML")
        or not hasattr(this_station_group, "preprocessed_27s_to_60s")
        or not hasattr(
            other_station_group, "preprocessed_synthetic_27s_to_60s"
        )
    ):
        return

    stationxml = this_station_group.StationXML
    observed = this_station_group.preprocessed_27s_to_60s
    synthetic = other_station_group.preprocessed_synthetic_27s_to_60s

    all_windows = []

    for component in ["Z", "R", "T"]:
        obs = observed.select(component=component)
        syn = synthetic.select(component=component)
        if not obs or not syn:
            continue

        windows = pyflex.select_windows(
            obs, syn, config, event=event, station=stationxml
        )
        print(
            "Station %s.%s component %s picked %i windows"
            % (
                stationxml[0].code,
                stationxml[0][0].code,
                component,
                len(windows),
            )
        )
        if not windows:
            continue
        all_windows.append(windows)
    return all_windows


import time

a = time.time()
results = ds.process_two_files_without_parallel_output(other_ds, process)
b = time.time()

if ds.mpi.rank == 0:
    print(results)
```

(continues on next page)

(continued from previous page)

```
    print(len(results))

print("Time taken:", b - a)

# Important when running with MPI as it might otherwise not be able to finish.
del ds
del other_ds
```

1.3.4 Calculate Source-Receiver Geometry

This simple example demonstrates a fast way to extract the source-receiver geometry from an ASDF file. It assumes that the `event_id` has been correctly set for each waveform and that these events are part of the global QuakeML file.

```
import pyasdf

with pyasdf.ASDFDataSet("./asdf_example.h5", mode="r") as ds:
    # Get dictionary of resource_id -> Lat/Lng pairs
    events = {
        str(e.resource_id): [
            (e.preferred_origin() or e.origins[0]).get(i)
            for i in ["latitude", "longitude"]
        ]
        for e in ds.events
    }

    # Loop over all stations.
    for s in ds.waveforms:
        try:
            coords = s.coordinates
        except pyasdf.ASDFException:
            continue

        # Get set of all event ids.
        #
        # Get set for all event ids - the `get_waveform_attributes()`
        # method is fairly new. If you version of pyasdf does not yet
        # have it please update or use:
        # group = s._WaveformAccessor__hdf5_group
        # event_ids = list({group[i].attrs.get("event_id", None)
        #                   for i in s.list()})
        # event_ids = [i.decode() for i in event_ids if i]

        # Note that this assumes only one event id per waveform.
        event_ids = set(
            _i["event_ids"][0]
            for _i in s.get_waveform_attributes().values()
            if "event_ids" in _i
        )

        for e_id in event_ids:
            if e_id not in events:
                continue
            # Do what you want - this will be called once per src/rec pair.
            print(
```

(continues on next page)

(continued from previous page)

```
                "%.2f %.2f %.2f %.2f"
                % (
                    events[e_id][0],
                    events[e_id][1],
                    coords["latitude"],
                    coords["longitude"],
                )
            )
```

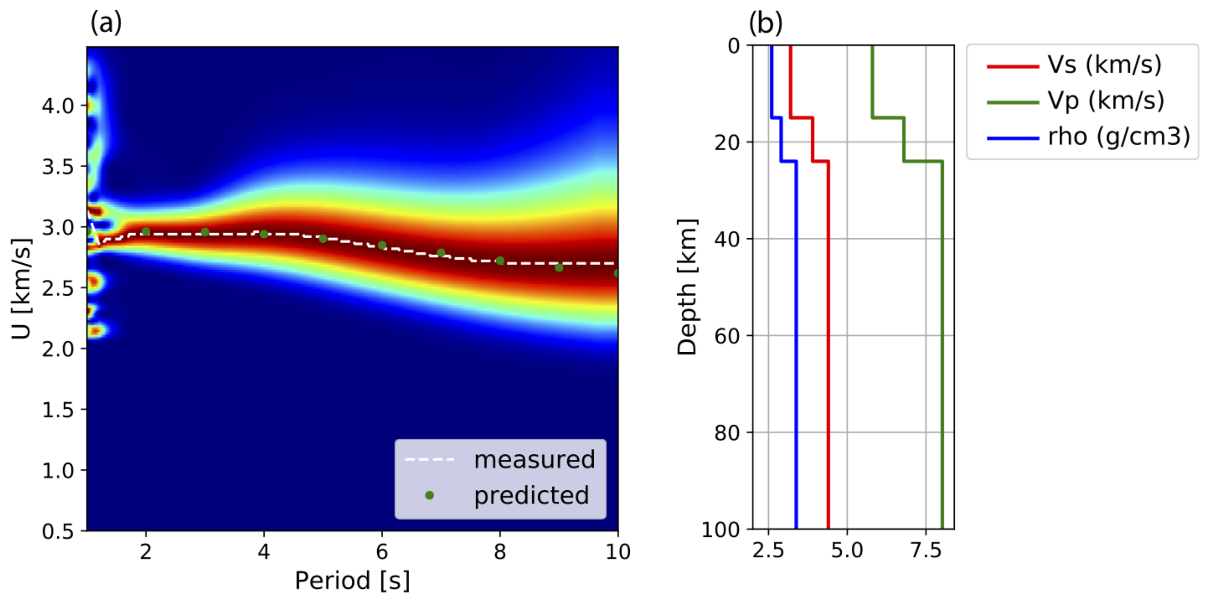
Note: The examples on pyasdf shown here are exclusively collected from the pyasdf offical website, which is subject to the BSD 3-Clause license.

1.4 Applications

To fulfill the strong need at user's end for applications based on ambient noise, NoisePy provides two application scripts for further surface wave dispersion analysis and seismic monitoring through measuring velocity change with time.

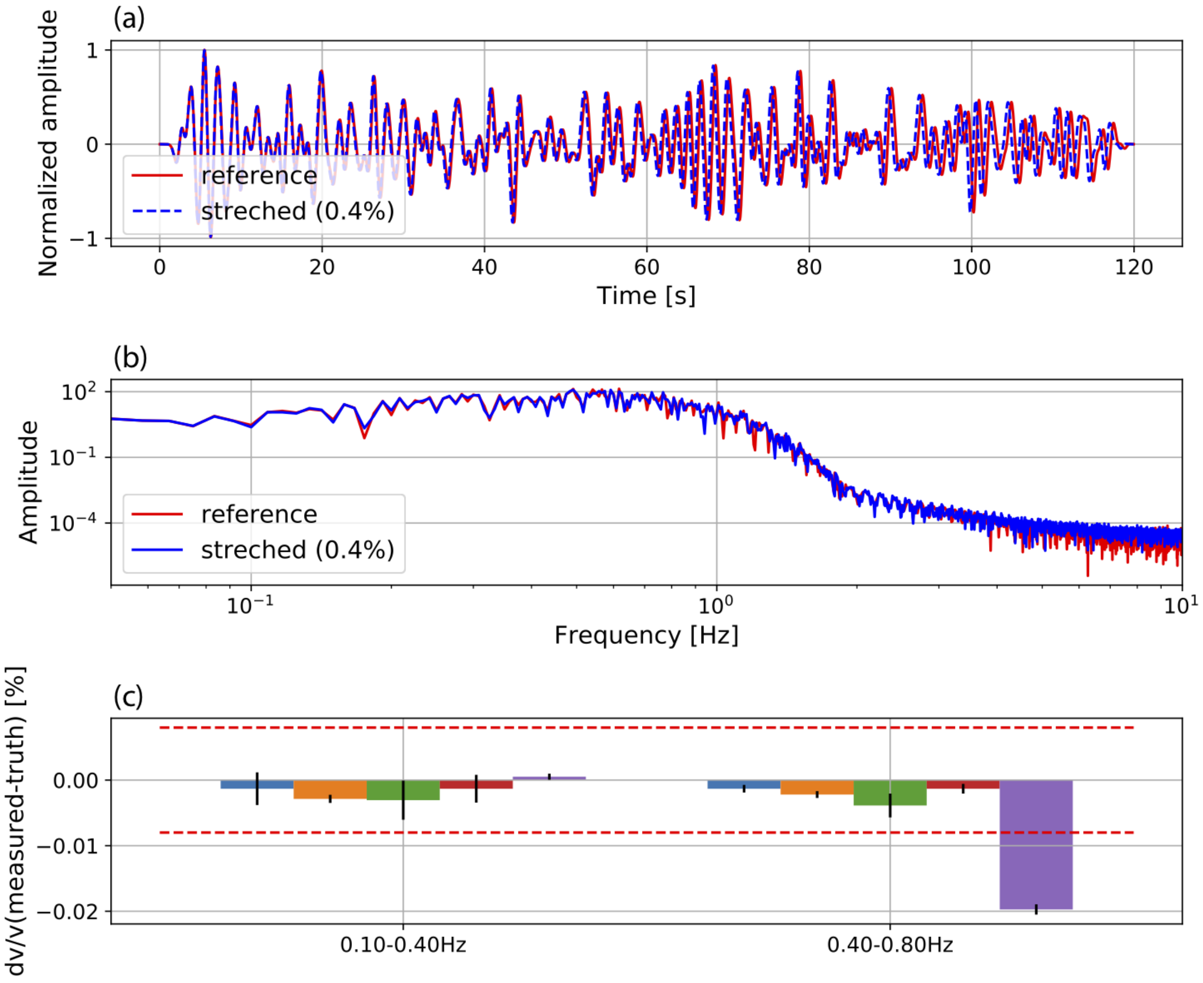
1.4.1 I. Group velocity measurements

The script of *I_group_velocity.py* is to estimate the group velocity using wavelet transform. The general idea is to apply narrow bandpass filters to the waveform and track the energy peak in each narrow frequency bands at multiple frequencies. The image below shows our synthetic test by cross-comparing the predicted dispersion curves using the wave propagation matrix method from CPS (Hermann et al., 2012) and those measured using our script upon a synthetic waveform from SPECFEM2D.



1.4.2 II. Monitoring velocity changes

The script of *II_measure_dvv.py* combines several general and popular methods for dv/v measurement including waveform stretching (Sens-Schönfelder and Wegler, 2006), dynamic time warping (Mikesell et al., 2015), moving-window cross spectrum (Clark et al., 2011), and the two newly developed methods in wavelet domain including 1) wavelet cross-spectrum (wcs; Mao et al., 2018) and wavelet stretching (Yuan et al., in prep).


(a)
Normalized amplitude
reference
streched (0.4%)
Time [s]
(b)
Amplitude
reference
streched (0.4%)
Frequency [Hz]
(c)
dv/v(measured-truth) [%]
0.10-0.40Hz
0.40-0.80Hz