



Deliverable 1.5: DAS Processing Algorithms

DigiMon

Digital monitoring of CO₂ storage projects

Prepared by

Antony Butcher (UoB)

Tom Hudson (UoO)

Wen Zhou (UoB)

Sacha Lapin (UoB)

J-Michael Kendall (UoO)

Alan Baird (UoB)

DigiMon Deliverable D.1.5 Version 1,

November 2021

Revision

Version	Date	Change	Page
1.0	23.07.2021	First version	All
1.1	20.08.2021	Amendments by T Hudson	All
1.2	26.08.2021	Update to noise module documentation	26-31

Document distribution

ACT Coordinator

- Research Council of Norway

ACT national funding agencies

- Forschungszentrum Jülich GmbH, Projektträger Jülich, (FZJ/PtJ), Germany.
- Geniki Grammatia Erevnas kai Technologias/The General Secretariat for Research and Technology (GSRT), Greece.
- Ministry of Economic Affairs and Climate/Rijksdienst voor Ondernemend Nederland (RVO), the Netherlands.
- The Research Council of Norway (RCN), Norway.
- Gassnova, Norway.
- Development and Executive Agency for Higher Education, Research, Development and Innovation Funding (UEFISCDI), Romania.
- Department for Business, Energy and Industrial Strategy (BEIS), UK.
- Department of Energy (DoE), USA.

DigiMon partners

- NORCE Norwegian Research Centre AS
- OCTIO Environmental Monitoring AS
- NTNU Norwegian University of Science and Technology
- University of Bristol
- University of Oxford
- CRES Centre for Renewable Energy Sources and Saving
- Helmholtz–Centre for Environmental Research
- Sedona Development SRL
- TNO Nederlandse Organisatie voor toegepast -natuurwetenschappelijk Onderzoek
- Geotomographie GmbH
- LLC Lawrence Livermore National Security
- SILIXA LTD
- EQUINOR ASA
- REPSOL –NORGE AS

Table of contents

1	Introduction	5
2	Installation	6
3	Module Overview	7
4	Module Descriptions	8
4.1	<i>IO module</i>	8
4.1.1	tdms_reader.py	8
4.1.2	utils.py	8
4.2	<i>plot module</i>	11
4.2.1	plot.py	11
4.3	<i>model module</i>	13
4.3.1	e3d_creator.py	13
4.3.2	raytrace.py	15
4.4	<i>Convert module</i>	16
4.4.1	convert.py	16
4.5	<i>filters module</i>	18
4.5.1	filters.py	18
4.5.2	qc.py	19
4.6	<i>detect module</i>	21
4.6.1	detect.py	21
4.6.2	raddetect.py	23
4.7	<i>active module</i>	25
4.8	<i>Noise module</i>	26
4.8.1	ani.py	26
4.8.2	visual.py	28
4.8.3	disp_inversion.py	30

1 Introduction

This report addresses deliverable D1.5 of the DigiMon project, which covers processing algorithms for Distributed Acoustic Systems (DAS) datasets that are contained within a python library called DASpy. The objective of DigiMon is to develop an early-warning system for Carbon Capture and Storage (CCS) which utilises a broad range of sensor technologies including DAS. While the system is primarily focused on the CCS projects located in shallow offshore environment of the North Sea, it is also intended to be adaptable to onshore settings. Some of the key areas that the systems will monitor include the movement of the plume within the reservoir, well integrity and CO₂ leakage into the overburden. A combination of both active and passive seismic methods will be deployed to track the movement of CO₂, for example seismic reflection to image seismic velocity changes or microseismics to capture small earthquakes relating to fault activation.

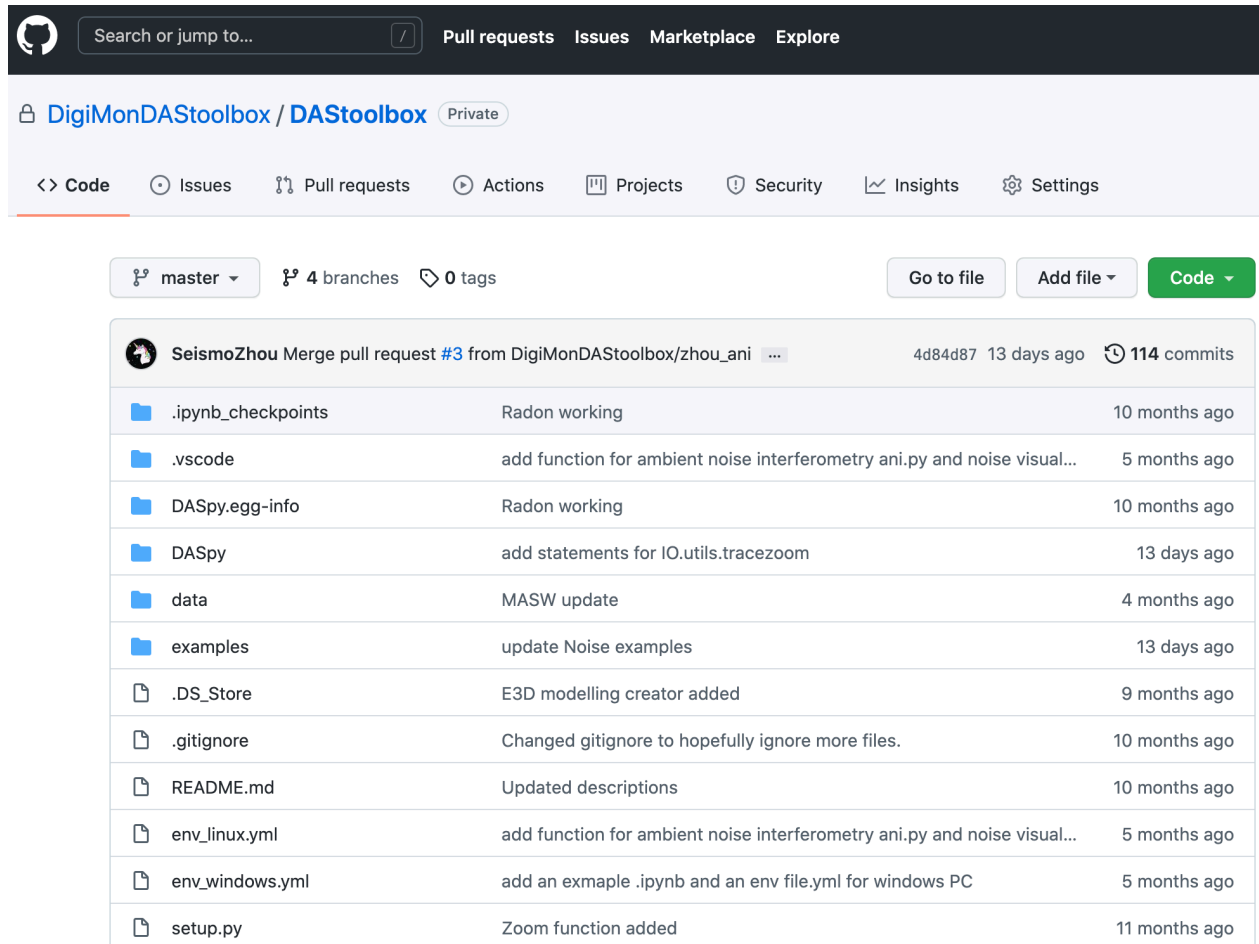
These processing algorithms have primarily been developed using the Antarctica DAS dataset, which was acquired on the Rutford Ice Stream and described in DigiMon report D1.1 and Hudson et. al (2021). The purpose of this dataset was to record icequakes that occur at the base of the Ice Stream and investigate the internal properties of the glacier. These were imaged using a surface deployed seismic array comprising of both fibre optic cables and 3-component geophones. This dataset contains both active and passive seismic measurements, which have been used to develop microseismic, Ambient Noise Interferometry (ANI), refraction and surface waves processing routines. These algorithms are under continuous development during the course of project and will be further refined by future DigiMon datasets (e.g. CAMi FRS Canada dataset).

The DASpy library is scripted using Python, which is a popular, widely supported language with a large range of dedicated libraries. We base the structure of the modules on the ObsPy library (Krischer et. al, 2015), an open-source library designed to facilitate the development of seismological software packages and workflows. Some of the DASpy workflows act as wrappers for existing programs, such as NonLinLoc (Lomax et. al, 2012) which is used to located seismic events. DASpy is hosted within a private GitHub repository call 'DAStoolbox' which also contains a number of example Jupyter notebooks and datasets which demonstrate the functionality of DASpy.

Within this report we describe the different modules contained within the DASpy library and its structure.

2 Installation

The DAStoolbox repository is hosted on GitHub, which is a hosting site for software development and version control. The repository is private and will require access rights to be granted prior to cloning or downloading the library. These can be obtain by contacting the authors of this report.



Search or jump to... / Pull requests Issues Marketplace Explore

DigiMonDAStoolbox / DAStoolbox Private

<> Code Issues Pull requests Actions Projects Security Insights Settings

master 4 branches 0 tags Go to file Add file Code

SeismoZhou Merge pull request #3 from DigiMonDAStoolbox/zhou_ani ... 4d84d87 13 days ago 114 commits

.ipynb_checkpoints	Radon working	10 months ago
.vscode	add function for ambient noise interferometry ani.py and noise visual...	5 months ago
DASpy.egg-info	Radon working	10 months ago
DASpy	add statements for IO.utils.tracezoom	13 days ago
data	MASW update	4 months ago
examples	update Noise examples	13 days ago
.DS_Store	E3D modelling creator added	9 months ago
.gitignore	Changed gitignore to hopefully ignore more files.	10 months ago
README.md	Updated descriptions	10 months ago
env_linux.yml	add function for ambient noise interferometry ani.py and noise visual...	5 months ago
env_windows.yml	add an exmaple .ipynb and an env file.yml for windows PC	5 months ago
setup.py	Zoom function added	11 months ago

Figure 1: Screen shot of the DAStoolbox GitHub repository.

3 DAsToolbox Overview

The python library, DAspy, contains a collection of functions and classes which are generically group into a range of different modules (Figure 2).

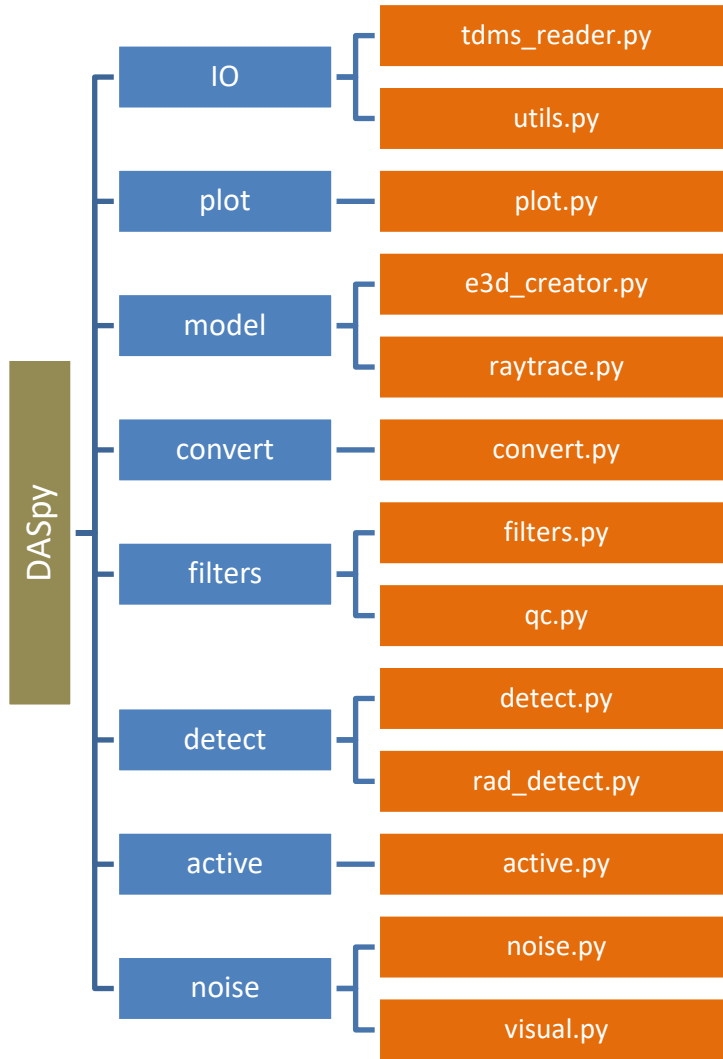
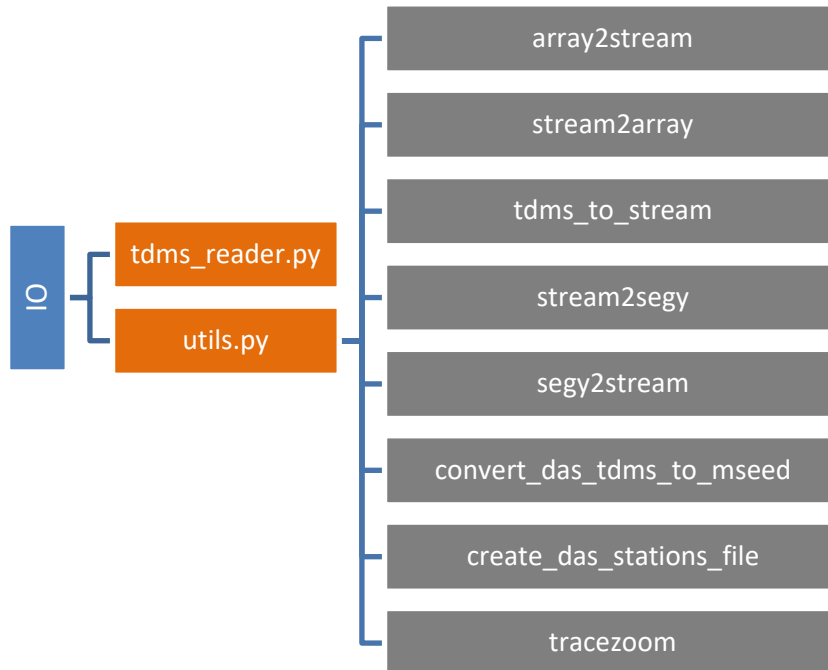


Figure 2: Structure of the DAspy library.

4 Module Descriptions

4.1 IO module

A collection of functions for importing, manipulating and exporting DAS data.



4.1.1 tdms_reader.py

Script for reading Silixa's iDAS TDMS file format. Downloaded from <https://silixa.com/resources/software-downloads/>.

4.1.2 utils.py

array2stream(np_array, network, fs, start, channel_spacing=1,units='Strain rate')

Converts numpy array format DAS data into an obspy stream.

Parameters:

- nparray - numpy array of DAS data
- network - seismic network name
- fs - sample rate in Hz
- start - data start time
- channel_spacing - spacing in metres
- units - DAS units

Returns:

- st - obspy stream

stream2array(st)

Populates a 2D np.array that is the traces as rows by the samples as cols.

Parameters:

st - obspy stream

Returns:

nparray - numpy array

tdms_to_stream(data_path,network, **kwargs)

Wrapper for the array2stream function. Reads in TDMS data and outputs an obspy stream.

Parameters:

data_path - path for TDMS file

network - seismic network name

Returns:

obspsy stream

stream2segy(st,path,id)

SEG Y writer for DAS data. Note: appears to remove ms from time.

Parameters:

path - path of segy file

id - unique file name

Returns:

st - obspsy stream

Returns:

SEG Y file

segy2stream(path,network,channel_spacing=1,units='Strain rate')

Reads in SEG Y file and outputs and obspsy stream with correct header information.

Parameters:

path - path of segy file

network - seismic network name

channel_spacing - spacing in metres

units - DAS units

Returns:

st - obspsy stream

convert_das_tdms_to_mseed(tdms_data_dir, mseed_out_dir, first_last_channels=[0,1000], network_code="AA", station_prefix="D", spatial_down_samp_factor=10, fk_filter_params={}, duplicate_Z_and_E=True, fold=False, apply_notch_filter=False, notch_freqs=[], notch_bw=2.5)

Function to read in tdms files and export them into mseed, in a format supported by QMigrate. Note: Works best with DAS data split into less than 1 hour chunks.

create_das_stations_file(fibre_lats, fibre_lons, fibre_elevs, dist_between_samps_m, station_dowsample_factor, out_fname, station_static_disp_offset=0.0)

Function to find das station coords and write to file for input to QMigrate. Note that elevations must be in km.

tracezoom(st,d1,d2,t1,t2)

Function to zoom in on a specific section of the trace.

Parameters:

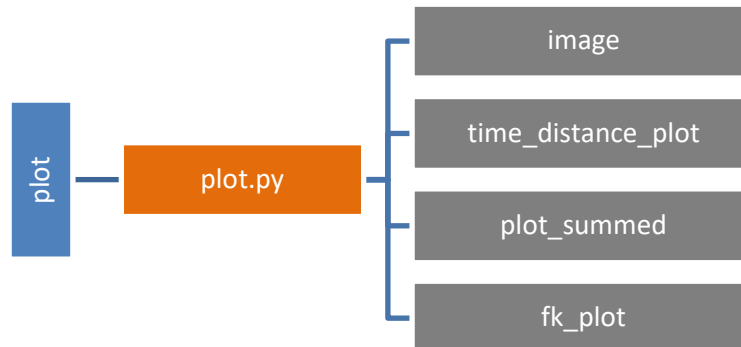
- st - obspy stream
- d1 - start zoom distance
- d2 - end zoom distance
- t1 - start zoom time
- t2 - end zoom distance

Returns:

- obsypy stream

4.2 plot module

A collection of plotting functions for DAS datasets.



4.2.1 plot.py

image(st,style=1,skip=10,clim=[0],tmin=0,tmax=None,physicalFiberLocations=False,picks=None)

Simple image plot of DAS stream, adapted from IRIS DAS workshop function.

#skip=10 is default to skip every 10 ch for speed

#style=1 is a raw plot, or 2 is a trace normalized plot

#clim=[min,max] will clip the colormap to [min,max], deactivated by default

Parameters:

st - The stream containing the DAS data to plot.

style - Type of plot. Default is raw plot (style=1). style=2 is a trace normalized plot.

skip - The decimation in the spatioal domain. Default is 10. (int)

clim - If specified, it is a list containing the lower and upper limits of the colormap. Default is [], which specifies that python defaults should be used. (list of 2 ints).

tmin - Plot start time in seconds.

tmax - Plot end time in seconds.

physicalFiberLocations - Defines distance from header information.

picks - DASpy event object. If specified, will plot phase picks on the figure. Default is None, which specifies it is unused. (DASpy detect.detect.event object)

Returns:

fig - A python figure object.

time_distance_plot(st, skip=10, channel_spacing_m=1.0, first_channel_offset_m=0.0, clim=[], event=None)

Function to plot DAS arrivals as simple time vs. distance plot.

Parameters:

st - The stream containing the DAS data to plot.

skip - The decimation in the spatial domain. Default is 10. (int)

channel_spacing_m - The spacing between each DAS channel, in metres. Default is 1.0 m. (float)

first_channel_offset_m - The spatial offset of the first channel in the stream <st>, in metres. Default is 0. (float)
clim - If specified, it is a list containing the lower and upper limits of the colormap. Default is [], which specifies that python defaults should be used. (list of 2 ints)
event - DASpy event object. If specified, will plot phase picks on the figure. Default is None, which specifies it is unused. (DASpy detect.detect.event object)

Returns:

fig - A python figure object.

plot_summed(img,xrange=None,dt=None,cmap='seismic'):

Plot function which displays an image of DAS data alongside amplitudes summed in both time and spatial domain.

Parameters:

img - narray of DAS data.
xrange - Spatial range of data
dt - sampling rate
cmap - colour map, default is 'seismic'

Returns:

fig - A python figure object.

fk_plot(st,wavenumber,max_freq,xrange=200,yrange=0.2,normalise=True):

FK filter for a 2D DAS numpy array. Returns a filtered image.

Parameters:

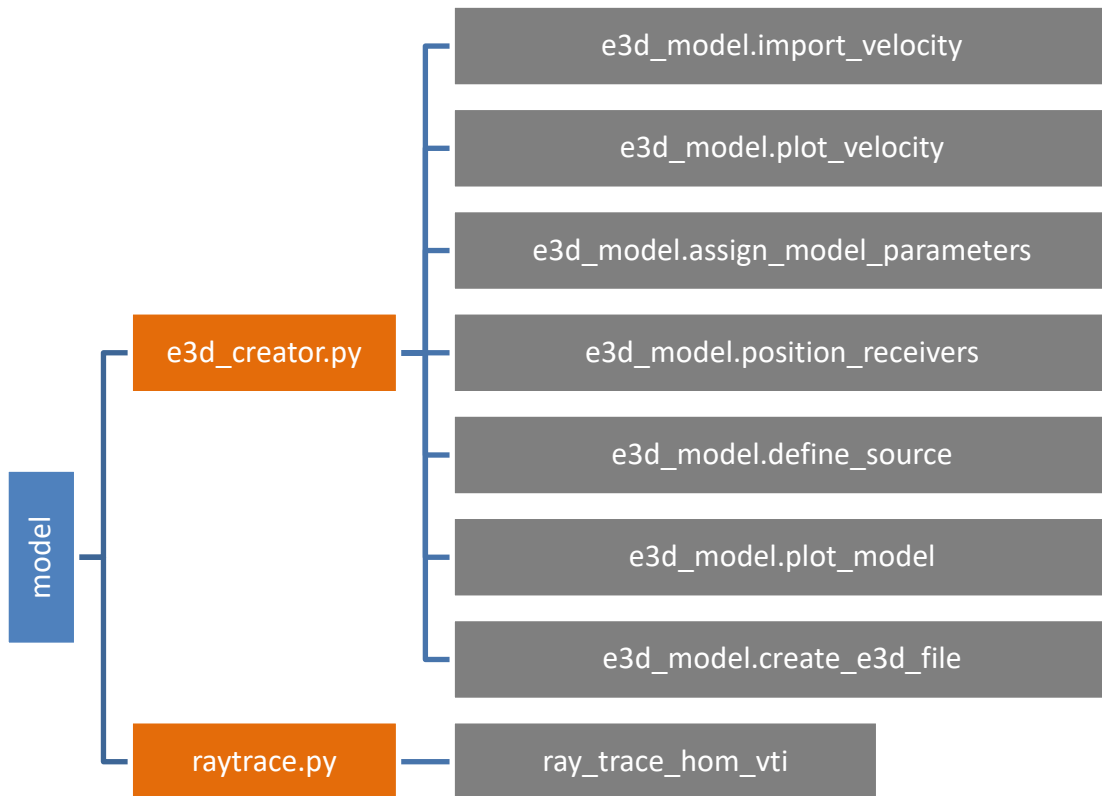
st - The stream containing the DAS data to plot.
wavenumber - maximum wavenumber value
max_freq - maximum frequency value
xrange - Spatial limit of data. Default is 200m
yrange - time limit of data. Default is 0.2s
normalise - apply fk plot to normalised data. Default it True, ie normalised.

Returns:

fig - A python figure object.

4.3 model module

Functions for modelling DAS data. This module effectively acts as a wrapper for E3D.



4.3.1 e3d_creator.py

These functions are used to create an input file for E3D.

import_velocity(self, fname, units='m'):

Imports velocity model from file

Parameters:

fname - velocity file in format [depth vp vs rho]
units - file units. Needs to be km for e3d.

plot_velocity(self):

Quick plotting function of velocity profile. To do: added attenuation to the model.

assign_model_parameters(self,xmax,zmax,dh,duration):

Defines the key model parameters.

Parameters:

xmax - Maximum length of model profile
zmax - Maximum depth of model profile
dh - Cell size. This is dependent on the minimum wavelength
duration - Time duration of the model

position_receivers(self,xstart,xend,dx=0,nrec=0,zstart=0,zend=0):

Defines receiver locations.

Parameters:

xstart - First receiver x location in km
xend - Last receiver x location in km
nrec - Number of receivers
zstart - First receiver z location in km
zend - Last receiver z location in km

define_source(self,srcx,srcz,src_type=1,freq=50,amp=1e+16,Mxx=1,Myy=1,Mzz=1,Mxy=0,Mxz=0,Myz=0)

Defines the source location and type.

Parameters:

srcx - x-coordinate of source
srcz - z-coordinate of source
src_type - Source types. 1: Explosive (p-wave); 4: Moment tensor

plot_model(self):

Quick plotting function of model dimensions. To do: add velocity model.

create_e3d_file(self,path='./')

Function to create an e3d input file

Parameters:

path - path to save file e.g. 'model/'

4.3.2 raytrace.py

`ray_trace_hom_vti(s,r,C,rho):`

Return ray attributes for given source and receiver locations and VTI stiffness tensor.

Based on equations from Chapman (2004): Fundamentals of seismic wave propagation, and Leaney (2014): Microseismic source inversion in anisotropic media.

Parameters:

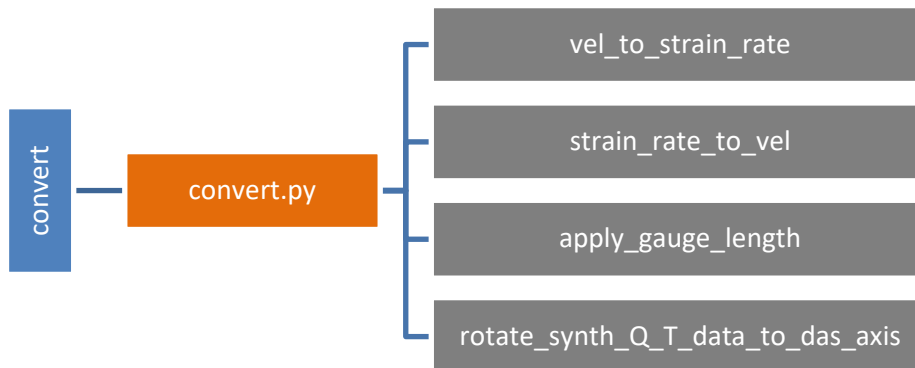
`s[3]` -- source coordinates
`r[3]` -- receiver coordinate
`C[3,3]` -- VTI stiffness tensor in Voigt notation
`rho` -- density

Returns:

`TqP` -- qP travel time
`TqSv` -- qSv travel time
`TSh` -- Sh travel time
`vqP` -- qP phase velocity
`vqSv` -- qSv phase velocity
`vSh` -- qSh phase velocity
`VqP` -- qP group velocity
`VqSv` -- qSv group velocity
`VSh` -- qSh group velocity
`SqP` -- qP spreading factor
`SqSv` -- qSv spreading factor
`SSh` -- qSh spreading factor
`gqP` -- qP polarization vector
`gqSv` -- qSv polarization vector
`gSh` -- qSh polarization vector
`qPp` -- qP slowness vector
`qSvp` -- qSv slowness vector
`Shp` -- qSh slowness vector

4.4 Convert module

Functions for converting DAS data into other units.



4.4.1 convert.py

vel_to_strain_rate(das_data_in, dx=1.0):

Function to convert DAS data from velocity to strain-rate. Note: Doesn't apply gauge length effects. This is done separately in `apply_gauge_length()`.

Parameters:

`das_data_in` - Array of data to convert from velocity to strain-rate. (np array)
`dx` - The spacing between the DAS channels. Can be a float or the same shape as `das_data_in` (float or np array)

Returns:

`das_data_out` - np array of strain rate data.

strain_rate_to_vel(das_data_in, dx=1.0):

""""TO BE COMPLETED""""

apply_gauge_length(das_data_in, gauge_length=10.0):

""""TO BE COMPLETED""""

rotate_synth_Q_T_data_to_das_axis(synth_data_q, synth_data_t, das_azi_from_N, azi_event_to_sta_from_N, aniso_angle_from_N=0.0, aniso_delay_t=0.0, fs=1000.0):

Function to rotate synthetic QT data into das axis, assuming vertical arrival angles.

Parameters:

synth_data_q - Array of synthetic Q component data. (np array of floats)

synth_data_t - Array of synthetic T component data. (np array of floats)

das_azi_from_N - DAS positive axis (away from interrogator) from North, in degrees (float)

azi_event_to_sta_from_N - Epicentral angle of station from event, from North, in degrees (float)

aniso_angle_from_N - Anisotropy angle from North in degrees. If -1.0 or 0.0, does not apply anisotropy. (float)

aniso_delay_t - Delay time between fast and slow shear waves, in seconds. If

aniso_angle_from_N <= 0, then no anisotropy is applied. Default is 0, i.e. no anisotropy applied. (float)

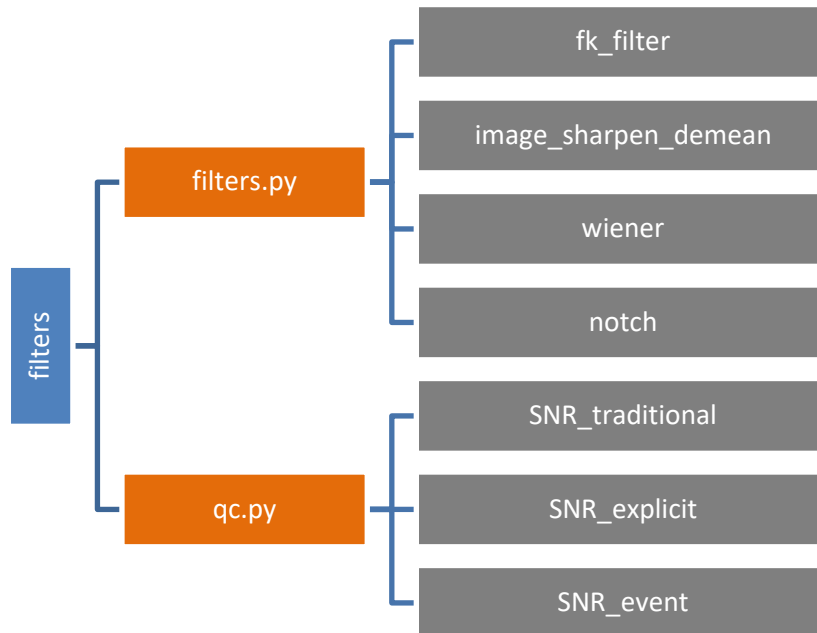
fs - Sampling rate of data in Hz (float)

Returns:

data_out_das_axis - Data out, rotated accordingly (array of floats)

4.5 filters module

Collection of filter functions for DAS data.



4.5.1 filters.py

fk_filter(*st, wavenumber, max_freq*):

FK filter for a 2D DAS numpy array. Returns a filtered image.

Parameters:

st - Stream of DAS data to apply notch filter to (obspy stream)
wavenumber - maximum value for the filter
max_freq - maximum value for the filter

Returns:

st_fk - FK filtered time series.

image_sharpen_demean(*st, sigma=3, alpha=30*):

Image sharpening function.

Parameters:

st - Stream of DAS data to apply notch filter to (obspy stream)
sigma -
alpha -

Returns:

st_shp - Image sharpened time series.

wiener(st):

Wiener filter with wrapper for applying to obspy streams

Parameters:

st - Stream of DAS data to apply notch filter to (obspy stream)

Returns:

st_wiener - The filtered time series.

notch(st, f_notch, bw):

Notch filter to filter out a specific frequency. Note: Applies a zero phase filter.

Parameters:

st - Stream of DAS data to apply notch filter to (obspy stream)

f_notch - The frequency to apply a notch filter for in Hz (float)

bw - The bandwidth of the notch filter in Hz (float)

Returns:

st_notch - The filtered time series.

4.5.2 qc.py

SNR_traditional(st_data, pow_vs_amp='power', return_all_channels=False):

Function to calculate traditional SNR ($\text{mean}^2 / \text{stdev}^2$).

SNR_explicit(st_data, st_noise, power_vs_amp='power', return_all_channels=False):

Function to calculate the SNR of a data window given a window of noise.

SNR_event(st, event, phase='S', nsamp_sig_win=100, nsamp_noise_win=100, return_all_channels=False):

Function to calculate the SNR of an event from its picks and windows around the signal and the noise. The SNR is defined here as the rms amplitude of the signal window divided by the rms amplitude of the noise window, as in Stork et al 2020.

Parameter:

st - Stream containing data associated with event arrivals and sufficient time before to window noise. (obspy stream)

event - DASpy.detect.detect event object containing phase picks for the event. This function will only use the phase picks associated with the phase specified as an optional input, <phase>. (DASpy event object)

phase - The phase to use (P or S). This controls what phase arrival times to use from event_phase_picks. Default is 'S' (str)

nsamp_sig_win - The number of samples to use for the signal window. (int)

nsamp_noise_win - The number of samples to use for the noise window. (int)

return_all_channels - If True, returns array containing SNR for each individual channel. (bool)

Returns:

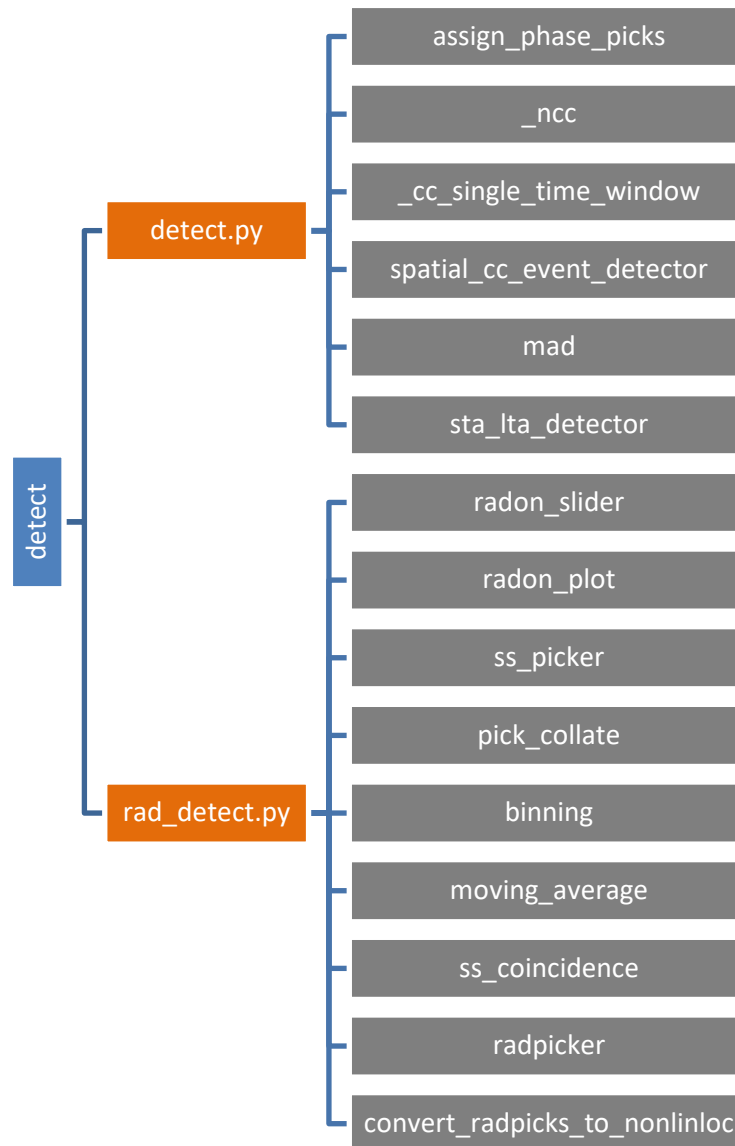
average_SNR - Average SNR for all channels combined. If return_all_channels = True, this value is not returned.

OR:

SNR_all_channels - If return_all_channels = True, will return array of SNR values for each individual channel.

4.6 detect module

Routines for detecting seismic events recorded using DAS.



4.6.1 detect.py

assign_phase_picks(self, stations, phase_time_picks, phase_labels, weights=[]):

Function to assign phase pick data to event.

Parameters:

stations - List of station labels. (list of str)

phase_time_picks - List of phase time picks for associated stations, in UTCDateTime format. (list of UTCDateTime objects)

phase_labels - List of phase labels associated with phase time picks. These can be <P> or <S>. e.g. ['P', 'S', 'S']. (list of specific str)

`_ncc(data, template):`

Function performing cross-correlation between long waveform data (`data`) and template. Performs normalized cross-correlation in fourier domain (since it is faster).

Returns normalised correlation coefficients. """

`_cc_single_time_window(st_curr_win, stations, max_samp_shift=10):`

Function to perform cross-correlation detection algorithm that shifts across a stream of channels in the spatial axis, for one time window, over the whole stream, `st`, input.

Parameters:

`st_curr_win` - Stream of DAS data, containing channels, labelled D???. (obspy stream)
`stations` - The stations to process for. This dictates the order over which the spatial cross-correlation is undertaken. (list of str).
`max_samp_shift` - The maximum shift to apply in samples. Default is 10. (int)

Returns:

`norm_cc_all_channels` - The normalised cc shift for all channels combined.

`spatial_cc_event_detector(st, win_len_secs=1.0, max_samp_shift=10, nproc=1):`

Function to detect events based on a spatial cross-correlation detection algorithm that shifts across a stream of channels in the spatial axis, one by one.

Parameters:

`st` - Stream of DAS data, containing channels, labelled D???. (obspy stream)
`win_len_secs` - Length of the moving window, in seconds. Default is 1 s. (float)
`max_samp_shift` - The maximum shift to apply in samples. Default is 10. (int)
`nproc` - The number of processors to use. Default is 1. (int)

Returns:

`ncc_st` - obspy stream

`mad(x, scale=1.4826):`

Calculates the Median Absolute Deviation (MAD) values for the input array `x`.

Returns:

MAD value with scaling.

`sta_lta_detector(st, sta_win=0.05, lta_win=0.25, MAD_multiplier=10.0, min_channels_trig=10, phase='S')`

Function to detect phase arrivals using an STA/LTA trigger with a Median Average Deviation trigger threshold. Note: This is currently a very simple trigger, which will just pick the highest STA/LTA value within the data for each channel.

4.6.2 raddetect.py

radon_slider(*st,start_distance=0,winsize=200,overlap=50,slowmin=-0.5e-3,slowmax=0,npx = 101*):

Sliding 2D radon transform using pylops

Parameters:

st - Stream of DAS data, containing channels
start_distance - beginning of the DAS trace, default=0
winsize - number of channels in the window
overlap - channel overlap

Returns:

semblance_out - a dictionary of the semblance and other key parameters

radon_plot(*semblance_out,windidx,picks=None*):

Radon transform plotter

Parameters:

semblance_out - dictionary output from the *radon_slider* function
windidx - window panel to plot

ss_picker(*semblance_out,windidx,neighborhood_size=50,threshold=130*):

Picks arrivals from the radon transform data

Parameters:

semblance_out - dictionary output from the *radon_slider* function
windidx - window panel to plot

Returns:

picks - dictionary containing slowness and pick time

pick_collate(*semblance_out,neighborhood_size=50,threshold=130*):

Collates picks from each window

Parameters:

semblance_out - dictionary output from the *radon_slider* function
neighborhood_size - window panel to plot

Returns:

picks - dictionary containing slowness and pick time

binning(*data,binsize,tmin,tmax*):

Bins data at user defined intervals

moving_average(*a, n=3*)

Fast moving average function

ss_coincidence(*semblance_out, event=1, threshold=120, pmin=-0.001, pmax=-0.0001, midpt=500, binsize=0.005, trigger_val=1, plot=True*)

A coincidence filter and wrapper for `ss_picker`

Parameters:

- `semblance_out` - dictionary output from the `radon_slider` function
- `threshold` - value for the radon picker
- `pmin` - minimum slowness value in s/km
- `pmax` - maximum slowness value in s/km
- `binsize` - bin size in seconds. Size dictates the sensitivity of the coincidence filter.

Returns:

- `picks` - dictionary containing slowness and pick time

radpicker(*files, event=1*):

This is a wrapper function which reads in a tdms file, filters the data, applies a radon transform and picks the events.

Parameters:

- `files` - list of files to process produced using `glob`
- `event` - starting event number

convert_radpicks_to_nonlinloc(*picks_all_df, nonlinloc_outdir="", phase='S', phase_pick_err=0.02*):

Function to take radon transform detection picks from `radpicker()` and convert these picks to individual events and nonlinloc files.

Parameters:

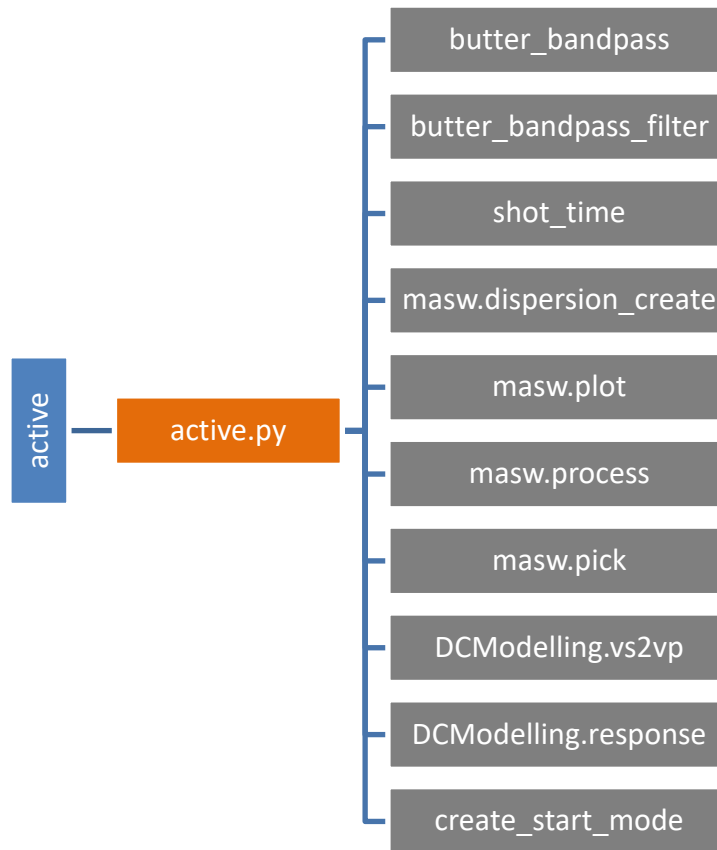
- `picks_all_df` - A pandas dataframe containing a list of event phase picks from `radpicker()`. (pandas df)
- `nonlinloc_outdir` - The directory to save the nonlinloc obs files to. Default is the current working directory. (str)
- `phase` - The phase labels to assign for nonlinloc. Should be P or S. (str)
- `phase_pick_err` - The time error associated with phase picks, in seconds. Default = 0.02 s (float)

Returns:

- `events` - A list of DASpy event objects] - not yet implemented.

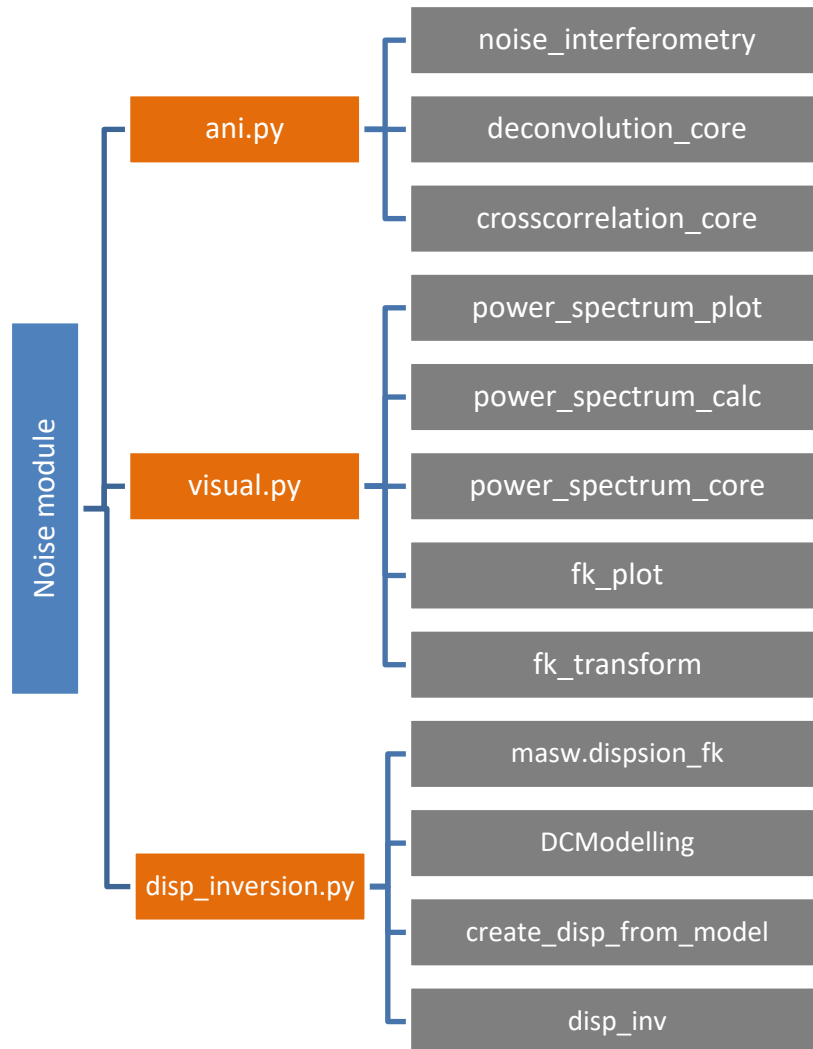
4.7 active module

Functions for processing active-source DAS data.



4.8 Noise module

The Noise analysis module contains 6 python script files. Core functions are defined in 3 files, named `ani.py`, `visual.py` and `disp_inversion.py`. Functions defined in each script are listed in the chart below.



4.8.1 ani.py

Ambient Noise Interferometry (ANI) functions including deconvolution and cross correlation. Data entry and interferometry parameters are defined in the function of:

```
noise_interferometry (stream, source_trace_num,  
                        sliding_window_length=1.0, overlap = 0.9,  
                        water_level=0.01, alpha = 100, core='deconvolution', onebit=False)
```

Apply interferometry to sliding windows over all traces in the Obspy Stream object.

stream <obspy stream>: multi channel data

source_trace_num <int>: The channel treated as virtual source (B in AB^{})*

sliding_window_length <float>: sliding window length in second

overlap <float>: Overlap of sliding windows, default 0.9 (90%)

water_level <float>: water level friction of maximal power spectrum (only) applied to deconvolution, default 0.01(1%)

alpha <float>: gaussian low pass filter based on $G = \exp(-f^2/\alpha^2)$, f (unit: Hz)

The first interferometry method implemented is water-levelled deconvolution in frequency domain.

deconvolution (x, y, dt, wl=0.001, alpha=20)

Deconvolution to x by y in frequency domain:

$$d(w) = \frac{x(w)y(w)^*G(w)}{\max(y(w)y(w)^*, \sigma \max(y(w)y(w)^*))}$$

0, preprocessing: demean, detrend, tapering

1, Fourier transform

2, water leveled deviation

3, gaussian low pass filter

x <numpy.ndarray>: space-time domain seismic data, 2D matrix

y <numpy.ndarray>: time series seismic data from certain channel, 1D

dt <float>: time domain sampling step = 1/sampling rate

wl <float>: water level friction of maximal power spectrum (only) applied to deconvolution, default 0.01(1%)

alpha <float>: gaussian low pass filter based on $G = \exp(-f^2/\alpha^2)$, f (unit: Hz)

The most widely used interferometry method is cross correlation. To improve convergency of interferograms, continuous noise are normalized in time and smoothed in frequency domain. We implement 1-bit time domain normalization and keep it as an option for user to determine. Frequency domain smoothing is implemented over a 21 points sliding window smoother.

crosscorrelation_core(x, y, dt, alpha=200, onebit=False)

Cross correlation of x and y

$$c(w) = \frac{x(w)y(w)^*G(w)}{\sqrt{(x(w)x(w)^*y(w)y(w)^*)}}$$

0, preprocessing: demean, detrend, tapering

1, Fourier transform

2, cross correlation (normalized) = cross coherence

x <numpy.ndarray>: space-time domain seismic data, 2D matrix

y <numpy.ndarray>: time series seismic data from certain channel, 1D

dt <float>: time domain sampling step = 1/sampling rate

alpha <float>: gaussian low pass filter based on $G = \exp(-f^2/\alpha^2)$, f (unit: Hz)

t_nor <str> : time domain normalization: '1-bit' or None

4.8.2 visual.py

The visualization of continuous noise is implemented in plot.py and described in previous sections. This script instead visualize noise in frequency and frequency-wavenumber domain.

power_spectrum_plot(stream, ax=None)

Plot averaged noise spectrum on log-log scale curve of.

Input -

stream: Obspy stream or trace

ax: matplotlib axes object

Output -

ax

power_spectrum_calc(stream)

Calculate power spectrum of each trace in an obspy stream object.

Input -

stream: Obspy stream or trace

Output -

Px: numpy array (2D [tr, nfft]), power spectrum density (c^2/Hz), 'c' is the unit of input data.

freq: numpy array (1D), frequency in Hz

power_spectrum_core(x, dt, npts=None)

Calculate power spectrum for an array x [tr, npts].

Input -

x: numpy array (2D, [tr, npts])

dt: float, sampling step in second

npts: int, number of sampling points, npts should $\geq x.\text{shape}[1]$

Output -

Px: numpy array (2D [tr, nfft]), power spectrum density (c^2/Hz), 'c' is the unit of input data.

freq: numpy array (1D), frequency in Hz

fk_plot(st, flim=[-100,100], dt=0.001, dx=1)

Plot fk transform with all traces in the st (NOTE: trace distance must be CONSTANT)

st could be an obspy stream or a numpy array.

only if st is a numpy array:

provide sampling information: dt, and dx

fk_transform(data, dt=0.001, dx=1, pad_x=0, pad_t=0)

fk spectrum of a seismic section

data_fk = output after fk-filter

f = frequency axis

kx = wave number axis

data [n_x, n_t]

*pad_x (0 -)= pad zeros on x (space) domain, 0= no padding, 1 = n_x, 2 = 2*n_x,...*

4.8.3 disp_inversion.py

dispersion_fk(self, st, freqmin=5, freqmax=30, kmin=0., kmax=0.1, vmin=500, vmax = 3000, a=50)

Pick fundamental mode surface wave dispersion curve from FK domain of DAS data.

Input -

st <Obspy Stream>: multi channel waveform

freqmin <float>: minimal frequency to be picked

freqmax <float>: maximal frequency to to picked

kmin <float>: minimal wavenumber

kmax <float>: maximal wavenumber

vmin <float>: minimal velocity

vmax <float>: maximal velocity

a <float>: a factor to suppress picking of higher mode surface wave.

*maximal velocity is defined: $v_{max} - a * freq$*

class **DCModelling**(pg.Modelling)

A class made for pygimli inversion.

1, contains elastic model information

2, calculating dispersion of the corresponding elastic model

create_disp_from_model(mat, freqs, disp_obs, poisson=0, plot=True):

Forward dispersion curve from a given model and plot with observed data.

Input -

disp_obs: observed Rayleigh wave velocities.

freqmin: minimum frequency of rayleigh waves.

freqmax: maximum frequency of rayleigh waves.

mat: initial velocity model

Output -

disp_model: Contains model dimension, the initial model and forwarding function.

disp_inv(disp_model, disp_obs)

Invert for the best fit dispersion curve.

Input -

disp_model: DCModelling class, contains model dimension, the initial model and forwarding function.

disp_obs: masw class, contains observed dispersion curve

Output:

coeff: final 1D Vs model

inv: pygimli inversion operator

5 References

- Hudson, T. S., Baird, A. F., Kendall, J. M., Kufner, S. K., Brisbourne, A. M., Smith, A. M., et al. (2021). Distributed Acoustic Sensing (DAS) for natural microseismicity studies: A case study from Antarctica. *Journal of Geophysical Research: Solid Earth*, 126, e2020JB021493. <https://doi.org/10.1029/2020JB021493>
- Krischer, L., Megies, T., Barsch, R., Beyreuther, M., Lecocq, T., Caudron, C., & Wassermann, J. (2015). ObsPy: A bridge for seismology into the scientific Python ecosystem. *Computational Science and Discovery*, 8(1). <https://doi.org/10.1088/1749-4699/8/1/014003>
- Lomax, A., Satriano, C., & Vassallo, M. (2012). Automatic Picker Developments and Optimization: FilterPicker--a Robust, Broadband Picker for Real-Time Seismic Monitoring and Earthquake Early Warning. *Seismological Research Letters*, 83(3), 531–540. <https://doi.org/10.1785/gssrl.83.3.531>