noisi_inv

The *noisi_inv* code is an extension of *noisi* (https://github.com/lermert/noisi). Besides being able to simulate noise cross-correlations and source sensitivity kernels for any noise source distribution and includes an inversion code with other additions to allow for an efficient inversion of secondary microseismic sources. It is a combination of various different codes that were included to automate the entire workflow. Hence, an inversion can be run by only adapting the *inversion_config.yml* file. In the following, parts of the inversion code will be explained briefly to give a user a slightly better idea of the different aspects of the code.

/noisi_inv

This is the main directory. It contains a tutorial in Jupyter Notebook and script form and the main inversion script with a configuration file. To run the inversion script after changing the inversion_config.yml file on one core, run “python run_inversion.py inversion_config.yml” or multiple cores with “mpirun -np N_CORES python run_inversion.py inversion_config.yml” where N_CORES is the number of cores you want to run it on.

```
├── LICENSE
│           → Licensing details
├── README.md
│           → Readme file that gives details about the installation.
├── Tutorial_inversion_setup.ipynb
│           → Jupyter notebook with a tutorial on how to setup the
│           inversion_config file for the inversion.
├── Tutorial_inversion_setup.py
│           → Above notebook in script form.
├── inversion_config.yml
│           → Configuration file for the inversion with various
│           different parameters.
├── noisi
│           → Directory with the code.
├── noisi_inv.yml
│           → Environment file that could be used for installation
├── noisi_inv_doc.docx
│           → This file.
├── requirements.txt
│           →Package requirements for this code.
├── run_inversion.py
│           → Main inversion script that calls various scripts
│           and mostly deals with I/O, copying files, performing the
│           inversion, plotting the output and things like that.
├── setup.py
            → Script needed to install the package.
```

/noisi_inv/noisi/ants

The ants package is taken from https://github.com/lermert/ants_2 and is used to process and cross-correlate seismic ambient noise data. To avoid the introduction of numerical artefacts we stick to simple processing and stacking of the data. This code was written by Laura Ermert and slightly adapted for noisi_inv. More details about the configuration files and what the parameters do can be found in the documentation of the ants_2.

```
├── classes
│   ├── corrblock_noisi.py
│           → class used to correlate and stack data.
|   ├── corrtrace_noisi.py
│           → class to deal with correlation I/O and metadata.
|   └── prepstream_noisi.py
            → class to process the raw data.
├── config_noisi.py
            → script to make noisi_inv communicate with ants and create
            config files that can be used to process and correlate
            seismic noise data automatically.
├── scripts
│   ├── ant_correlation_noisi.py
            → main script that calls classes and functions to create
            cross-correlations.
|   └── ant_preprocess_noisi.py
            → main script that calls classes and functions to
            pre-process the raw data.
└── tools
    ├── bookkeep.py
            → file management for ants.
    ├── correlations.py
            → functions needed to cross-correlate the data.
    ├── data_splitter.py
            → not used in noisi_inv.
    ├── download_staxml.py
            → not used in noisi_inv.
    ├── geo.py
            → geographical functions.
    ├── measurements.py
            → different measurements that could be taken on the
            correlations.
    ├── plot.py
            → Plotting functions.
    ├── prepare.py
            → functions to prepare the raw data to be processed.
    ├── treatment.py
            → functions to treat the processed data for the
            correlating.
    ├── util.py
            → not used in noisi_inv.
    ├── util_noisi.py
            → function to get geographic information from stationxml
            files.
    └── windows.py
            → functions to calculate windows for correlations.
```

/noisi_inv/noisi/borrowed_functions

These scripts are necessary to compute the Voronoi cells and surface area. The first script is part of the NSF DIBBS Project "An Infrastructure for Computer Aided Discovery in Geoscience" (PI: V. Pankratius) and NASA AIST Project "Computer-Aided Discovery of Earth Surface Deformation Phenomena" (PI: V. Pankratius). The second one is taken from https://github.com/tylerjereddy/spherical-SA-docker-demo/blob/master/docker_build/demonstration.py

```
├── voronoi_polygons.py
            → functions to compute the Voronoi polygons for any source
            grid
└── voronoi_surface_area.py
            → functions to calculate the surface area of a Voronoi
            polygon
```

/noisi_inv/noisi/mfp

This package uses a Matched Field Processing method which can then be implemented as initial model in a noisi_inv inversion. The standalone package can be found here: [https://github.com/jigel/mfp_surf_body.](https://github.com/jigel/mfp_surf_body) Several parts were adapted to allow for the automatic computation of an MFP model within noisi_inv.

```
├── mfp_code
│   ├── scripts
│   │   ├── create_sourcegrid.py
            → function that takes a config file and creates a source
            grid. Not needed for noisi_inv.
│   │   ├── create_stat_phase_synthetics.py
            → script to create synthetic correlations for a stationary
            phase analysis. Not needed for noisi_inv.
│   │   └── mfp_main.py
            → main MFP function that creates the noise source maps for
            different measurements.
│   └── util
│       ├── plot.py
            → script to plot the noise source maps.
│       └── source_grid_svp.py
            → function to create a spatially variable grid.
├── mfp_config.yml
            → configuration file
└── run_mfp.py
            → main script used to run the MFP. This is called by
            noisi_inv to create the initial model.
```

/noisi_inv /noisi/my_classes

Classes that are used within noisi_inv to deal with the hdf5 files.

```
├── noisesource.py
            → class to deal with the noise source hdf5 files.
└── wavefield.py
            → class to deal with the wavefield hdf5 files.
```

/noisi_inv /noisi/scripts

This directory contains all the main scripts that are needed to create a new project and forward model cross-correlations and sensitivity kernels for different measurements.

```
├── adjnt_functs.py
            → functions to compute the adjoint sources for several
            different measurements.
├── assemble_gradient.py
            → function to assemble the gradient, i.e. sum of the
            sensitivity kernels.
├── correlation.py
            → functions to forward model cross-correlations.
├── kernel.py
            → functions to compute the source sensitivity kernels.
├── measurements.py
            → functions for the difference measurements.
├── rotate_kernel.py
            → functions to rotate the kernels if a horizontal component
            is involved.
├── run_measurement.py
            → functions to take the different measurements on the
            correlations.
├── run_sourcesetup.py
            → functions to setup the noise source distribution
├── run_wavefieldprep.py
            → class and functions to pre-compute the wavefield
└── source_grid.py
            → functions to create different source grids
```

/noisi_inv /noisi/test

These script are used to test the code. These can be run with the “pytest” command in the terminal.

```
├── check_adjstf.py
            → checks the adjoint source functions.
├── check_gradient.py
            → checks the gradient.
├── test_scripts.py
            → main test script.
└── testdata_v1
            → directory with test data in it.
```

/noisi_inv /noisi/util

These script contain several functions that are needed to support the main code in the /scripts directory.

```
├── add_metadata.py
            → function to assign geographic metadata to correlations.
├── ants_crosscorrelate.py
            → functions to setup the configuration files for the ants
            processing and correlating. The parameters for these are
            set in this script.
├── auto_data_grid.py
            → function to automatically setup a spatially variable
            grid based on some input data.
├── compress_files.py
            → script to convert sac to asdf and npy to h5.
├── corr_add_noise.py
            → function to add random noise to synthetic correlations.
├── corr_obs_copy.py
            → function to copy and select correlations based on a
            signal to noise ratio.
├── corr_pairs.py
            → function to define all possible correlation pairs and
            checks which ones already exist.
├── filter.py
            → different filter functions.
├── geo.py
            → geographical functions.
├── inv_step_test.py
            → script to perform a step length test for the inversion.
├── obspy_mass_download.py
            → obspy mass downloader to automatically download data for
            certain time ranges and station lists.
├── output_copy.py
            → copy the output of the inversion into one folder.
├── output_plot.py
            → plot the most important output of the inversion.
├── plot.py
            → plot functions for the source grid.
├── rotate_horizontal_components.py
            → functions to rotate the horizontal components.
├── setup_new.py
            → script to setup a new project.
├── smoothing.py
            → functions to spatially smooth the distributions.
├── source_grid_svp.py
            → main code to create a spatially variable grid.
├── syngine_download.py
            → script to download a pre-computed wavefield from syngine.
└── windows.py
            → functions to calculate windows for correlations.
```

inversion_config.yml

To perform an inversion, the configuration file has to be adapted. An example of what the different parameters do can be found in the "Tutorial_inversion_setup.ipynb" Jupyter Notebook. Here we will go through all parameters and provide a little more detail.

main:

output_folder: ./noisi_output

→ Path to where project should be saved. Does not have to exist.

stationlist: ./noisi/examples_inv/stationlist_swiss_6.csv

→ Path to a stationlist. This should be a .csv file with columns: net, sta, lat, lon. If not stationlist is given and download_data is true, a stationlist will be created from the available data.

add_metadata: false

→ Set this to true if only a forward simulation is being run and the metadata should be added to the cross-correlations files.

add_noise: 1.5

→ This adds random noise to the cross-correlation if not set to 0, 'False', or 'null'. It does this by creating a random time series the same length as the cross-correlation, multiplying it with maximum amplitude of the cross-correlation and the scaling term given here, filtering it with the bandpass filter given for the measurement, and finally adding it to the cross-correlation.

output_plot: true

→ Set this to true if the output should be plotted automatically at the end of the inversion.

project_config:

project_name: noisi_inv_example

→ Give the project a name.

synt_data: DIS

→ What type of data should the synthetics be? Can be either 'DIS', 'VEL', or 'ACC'.

verbose: false

→ Set this to 'true' if you want print a lot of information.

load_to_memory: true

→ This loads the wavefields to memory which can speed up the modelling signficiantly if the memory is large enough to store the wavefield.

compress_output_files: false

→ If set to 'true' this will combine the .sac files into one .asdf file and the .npy files into one .h5 file. If set to 'delete' it will delete the .sac and .npy files after each iteration to save space.

data_download:

→ Parameters to download data within noisi_inv

download_data: true

→ Set this to true if you want to use the implemented obspy mass downloader to download data.

download_data_date: yesterday

→ Pick the date for which you want to download data. This can either be 'yesterday' or a date in the format '2022-11-08'.

download_data_days: 1

→ Number of days that the data should be downloaded for after the date chosen in download_data_date.

download_data_channels: ['BHZ']

→ Channels that should be downloaded. Can include wildcards, i.e. ['*Z','*E','*N'].

process_data_win_len: 7200

→ Window length in seconds that the data will be split into for the stacking and correlating.

process_data_overlap: 100

→ Overlap of the above set windows in seconds.

gcmt_exclude: False

→ If set to True all windows that include an earthquake in the GCMT catalogue above a magnitude of 5.6 will be excluded.

NOTE: The parameters below will be used if station list is None and download_data is True to create a station list from all the available data within the given domain.

min_station_dist: 0

→ Minimum interstation distance in degrees for

domain_type: circular

→ Domain to be used to look for stations and data. One of the following: circular, rectangular, global.

circ_lat_center: 46
circ_lon_center: 8
circ_radius_min: 0
circ_radius_max: 2

→ Parameters for a circular domain with the latitude/longitude of the centre, and minimum/maximum radius from the centre.

rect_lat_min: 30
rect_lat_max: 50
rect_lon_min: -10
rect_lon_max: 30

→ Parameters for a rectangular domain with the minimum/maximum latitude and longitude.

inversion_config:

→ Parameters for the inversion

observed_corr: null

→ Path to the observed correlations. Set to null if the data is being downloaded here.

opt_statpair: null

→ Path to a csv file that contains the optimal station pairs.

snr_thresh: 0

→ Signal-to-noise ratio threshold above which correlations will be used for the inversion. Set to 0 or null to include all correlations.

corr_max_dist: null

→ Maximum distance between station pairs in degrees. Set to null if this should be ignored.

nr_iterations: 1

→ Number of iterations for the inversion. Set to 0 if only a forward model should be run.

nr_step_tests: 5

→ Number of step tests for each iteration.

step_test_smoothing: false

→ Tests if the misfit of the updated model is below the previous misfit. If not, smoothing is reduced in steps of 0.5 degrees.

frac_corr_slt: 1

→ Fraction of correlations that is used for the step length test, e.g. 2 means 1/2 of the correlations are used Set to 1 to use all correlations.

step_length_min: 0.05

→ Minimum step length that will be tested.

step_length_max: 3.0

→ Maximum step length that will be tested.

step_smooth: [[2,3],[4,2],[6,1.5],[8,1]]

→ list of iterations and smoothing in degrees. [[iteration1,smoothing1],[iteration2,smoothing2],..]. Up to iteration1 (could be iteration number 2 or 3), smoothing1 will be used. Afterwards smoothing2 will be used up to iteration2 etc.

smoothing_cap: 95

→ Percentage at which the distribution will be capped when it is being smoothed.

grid_config:

→ Parameters to setup a rectangular grid

grid_dx_in_m: 35000
grid_lat_max: 65
grid_lat_min: 30
grid_lon_max: 25
grid_lon_min: -15

→ Grid point distance in metres and minimum/maximum latitude/longitude. These parameters will be used if the svp_grid parameter below is set to False.

svp_grid_config:

→ Parameters to setup a spatially variable grid.

svp_grid: true

→ Set to true if a spatially variable grid should be used.

svp_dense_antipole: false

→ This creates a variable grid which is mirrored at the equator before being rotated to its centre, i.e. the antipole will also have a denser grid.

svp_only_ocean: true

→ Set to true if all grid points on land should be removed.

svp_voronoi_area: true

→ Set to true if the Voronoi cell surface areas should be calculated. This is necessary to forward model correlations and kernels properly.

svp_station_remove: 1

→ Radius in degrees around stations where any grid points will be removed.

svp_plot: false

→ Set to true if the grid should be plotted.

NOTE: The following parameters are lists because multiple values can be given for multiple grids. If this is the case, the svp_gamma parameters sets the radius around the centre where the new grid will be included.

svp_beta: [7]
→ Steepness of the drop from dense to sparse grid. The higher the number, the steeper the grid point distance increase.
svp_gamma: [0]
→ Radius in degrees where a new grid will replace the old one. Only necessary if multiple grids are given.
svp_lat_0: [47]
→ Latitude of the spatially variable grid centre.
svp_lon_0: [8]
→ Longitude of the spatially variable grid centre.
svp_phi_min: [2]
→ Minimum grid point distance in degrees.
svp_phi_max: [5]
→ Maximum grid point distance in degrees.
svp_sigma: [10]
→ Radius in degrees around centre where grid point distance is svp_phi_min.

NOTE: The automatic grid is only recommended for forward modelling of cross-correlations for a given data set, not for inversions. It is best to play around with this in the Jupyter Notebook to see how many grid points there would be.

auto_data_grid_config:
→ These parameters can be set to automatically create a spatially variable grid based on input data.
auto_data_grid: false
→ Set to true if you want an automatic spatially variable grid.
auto_data_path: null
→ Path to the data the grid should be based on. The data should be a .npy file with latitude, longitude, data.
auto_data_thresh: 0.5
→ Dense grids will be added for data point above this threshold for normalised data.
auto_station_remove: 1
→ Radius around stations in degrees where grid is removed.
auto_back_grid_centre: stations
→ Can be either 'data', 'stations', or [longitude, latitude]. Sets the centre of the background grid.
auto_back_grid_phi_min: 2
→ Minimum grid point distance in degrees for the background grid.
auto_back_grid_phi_max: 4
→ Maximum grid point distance in degrees for the background grid.
auto_data_grid_phi: 1
→ Grid point distance in degrees of the additional grids that are added to the background grid.
auto_extent: [-90,0,0,90]
→ Area where denser grids will be added.
auto_gamma_thresh: 5
→ Radius in degrees of the new denser grids that are added.

wavefield_config:

→ Parameters for the wavefield computation.

wavefield_channel: Z

→ Channels that should be modelled. Can be one of 'Z','N', 'E' or a list like ['Z','N'] or 'all'.

wavefield_domain: time

→ Domain for the wavefield computation, either time or frequency.

wavefield_duration: 5000.0

→ Duration of the simulation in seconds.

wavefield_filter: null

→ Bandpass filter for the wavefield as a list,e.g. [0.01,0.2].

wavefield_path: null

→ Path to the wavefield. If wavefield_type is 'instaseis' this should be a path to an AxiSEM wavefield. If wavefield_type is 'greens' this should be a path to a folder with already pre-computed Green's functions.

wavefield_point_force: 1.0e9

→ Force of the vertical point source used for the AxiSEM simulations.

wavefield_sampling_rate: 1.0

→ Sampling rate of the wavefield in Hz.

wavefield_type: analytic

→ Wavefield type can either be 'analytic', 'instaseis', or 'greens'.

v: 3000.0

q: 100.0

rho: 3000.0

→ If the wavefield_type is 'analytic', set the velocity, attenuation, and density here.

source_config:

→ Configuration for the source.

get_auto_corr: false

→ Set to true if autocorrelation should be computed.

model_observed_only: true

→ Set to true if only observed correlations should be modelled.

max_lag: 1500

→ Maximum lag of the cross-correlation in seconds. Needs to be less than half of wavefield_duration.

diagonals: false

→ NOT IMPLEMENTED.

rotate_horizontal_components: false

→ Set to true to rotate horizontal components to Z, R, T.

source_setup_config:

→ Parameters for the source distribution. In theory,multiple distributions can be given.

- distribution: homogeneous

→ Distribution type can be one of the following: homogeneous, zero or homogeneous_0, random, ocean, gaussian_blob, mfp or matchedfieldprocessing.

mfp_smooth: 4

→ Smoothing parameter in degrees if mfp is chosen as distribution.

center_latlon: [[-50,-20],[0,0]
→ Latitude/Longitude of the centres of the gaussian blobs if distribution is gaussian_blob.
sigma_m: [1000000, 1000000]
→ Size of the gaussian blobs in metres.
mean_frequency_Hz: 0.15
→ Mean frequency of the distribution.
standard_deviation_Hz: 0.05
→ Standard deviation of the frequency spectrum of the distribution.
weight: 1.0
→ Weight of the distribution.
taper_percent: 1.0
→ Taper for the frequency spectrum.
normalize_spectrum_to_unity: true
→ Set to true if the frequency spectrum should be normalised.

measr_config:
→ Parameters for the inversion measurement.
mtype: "ln_energy_ratio"
→ Measurement type can be one of the following: ln_energy_ratio, energy_diff, square_envelope, envelope, full_waveform.
taper_perc: 0.01
→ Taper for the bandpass filter.
bandpass: [[0.01,0.2,5]]
→ Bandpass filter given is a list with [[freq_min,freq_max,corners]].
weights: [1.0]
→ Weights of the bandpass filter if multiple are given.
g_speed: 2700
→ Surface wave speed for the windowed measurements in m/s.
waterlevel_perc: 0
→ NOT IMPLEMENTED ANYMORE.
window_params_wtype: "hann"
→ Window type for the measurement can be either 'hann' or 'boxcar'.
window_params_hw: 30
→ Width of the window in seconds.
window_params_hw_variable: 100
→ Parameter for a variable window size in m/s. Creates a window with g_speed - this value - window_params_hw, g_speed + this value + window_params_hw.
window_params_win_overlap: false
→ Set to true if the windows are allowed to overlap.
window_params_sep_noise: 0.0
→ Seperation between noise and data window to calculate a signal-to-noise ratio.
window_plot_measurements: false
→ Set to true if you want to plot the measurement.
ignore_network: false
→ Set to true if the network of the stations should be ignored.