4.2.3 Moving-Window Cross-Spectral method ........................................... 53
4.2.4 WLS ................................................................................................. 54
4.3 Table Definitions ................................................................................. 55
4.4 Extending MSNoise with Plugins ....................................................... 57
  4.4.1 What is a Plugin and how to declare it in MSNoise ............. 57
  4.4.2 Plugin minimal structure ................................................................. 58
  4.4.3 Declaring Job Types - Hooking ..................................................... 59
  4.4.4 Plugin’s own config table ............................................................... 61
  4.4.5 Adding Web Admin Pages ............................................................. 63
  4.4.6 Uninstalling Plugins ..................................................................... 64
  4.4.7 Download Amazing Plugin ........................................................... 64
4.5 How To’s .............................................................................................. 65
  4.5.1 Reprocess data ............................................................................... 65
  4.5.2 Define one's own data structure of the waveform archive ...... 66
  4.5.3 How to have MSNoise work with 2+ data structures at the same time 66
  4.5.4 How to duplicate/dump the MSNoise configuration ............. 66
  4.5.5 Testing the Dependencies ............................................................. 66
4.6 About Databases and Performances .................................................... 67
4.7 References ........................................................................................... 68
4.8 Contributors ........................................................................................ 69
4.9 Help on the msnoise commands .......................................................... 69
  4.9.1 admin ............................................................................................. 69
  4.9.2 bugreport ....................................................................................... 69
  4.9.3 compute_cc .................................................................................... 70
  4.9.4 compute_dtt .................................................................................. 70
  4.9.5 compute_mwcs ............................................................................. 70
  4.9.6 compute_stretching ...................................................................... 70
  4.9.7 config ............................................................................................ 71
  4.9.8 info ................................................................................................ 71
  4.9.9 install ............................................................................................. 71
  4.9.10 ipython ........................................................................................ 71
  4.9.11 new_jobs ...................................................................................... 72
  4.9.12 p .................................................................................................. 72
  4.9.13 plot ............................................................................................... 72
  4.9.14 plugin .......................................................................................... 76
  4.9.15 populate ....................................................................................... 76
  4.9.16 reset .............................................................................................. 76
  4.9.17 scan_archive ................................................................................ 76
  4.9.18 stack ............................................................................................. 77
  4.9.19 test ............................................................................................... 77
  4.9.20 upgrade_db ................................................................................... 77
5 Release Notes ............................................................................................ 79
  5.1 MSNoise 1.5 ....................................................................................... 79
    5.1.1 Introduction .................................................................................. 80
    5.1.2 Requirements .............................................................................. 81
    5.1.3 Web-based Admin Interface Changes ................................... 81
    5.1.4 Populate Station Table and Scan Archive ........................... 82
    5.1.5 New Jobs ..................................................................................... 83
    5.1.6 Preprocessing and Cross-Correlation ................................... 83
    5.1.7 Command Line changes ............................................................. 84
Originally, MSNoise was a “Python Package for Monitoring Seismic Velocity Changes using Ambient Seismic Noise”. With the release of MSNoise 1.4, and because of the Plugin Support, we could call MSNoise: “Measuring with Seismic Noise”. The current release version of MSNoise is MSNoise 1.5.

The standard MSNoise workflow is designed to go from seismic data archives to $dv/v$ curves. The monitoring is achieved by computing the cross-correlation of continuous seismic records for each pair of a network and by studying the changes in the crosscorrelation function relative to a reference.

The goal of the “suite” is to provide researchers with an efficient processing tool, while keeping the need for coding to a minimum and avoiding being a black box. Moreover, as long as the inputs and outputs of each step are respected, they can easily be replaced with one’s own codes! (See Workflow (page 11)).

Plugins can be added and extend the standard workflow from any steps, e.g. using MSNoise as a cross-correlation toolbox until the stack step, and then branching to the workflow provided by one’s plugin.


This documentation is also available in PDF format on the MSNoise Website (PDF).
1.1 Installation

MSNoise is a python package that uses a database (sqlite or MySQL) for storing station and files metadata together with jobs. When installed, it provides a top level command `msnoise` in the console.

To run MSNoise, you need:

- A recent version of Python (3.x recommended). We suggest using Anaconda with a few extra modules. MSNoise is tested “continuously” by automatic build systems (TravisCI and Appveyor) for **Python 2.7 and Python 3.5**, on **Windows, Linux and MacOSX 64 bits systems**!
  
  - Those modules are already distributed with Anaconda:
    * setuptools
    * numpy
    * scipy
    * pandas
    * matplotlib
    * statsmodels
    * sqlalchemy
    * click
    * flask
    * pymysql
    * wtforms
  
  - Not shipped with Anaconda:
    * obspy
    * flask-admin
    * markdown
    * folium
    * flask-wtf
• MySQL: if you want to use MySQL, you need to install and configure a MySQL Server beforehand. This is not needed for sqlite. Read About Databases and Performances (page 67) for more information. We recommend using MySQL.

1.2 Full Installation

1. Download and install Anaconda for your machine, make sure Anaconda’s Python is the default python for your user
2. Execute the following command to install the missing packages:

   ```
   pip install flask-admin flask-wtf markdown folium
   conda install -c conda-forge obspy
   ```

3. Install a MySQL server and MySQL Workbench:

   Download and install MySQL Community Server (MySQLs ) and MySQL Workbench (MySQLw ) ; On Windows one can also use the MySQL installer (MySQLi ).

   On Linux, the MySQL server can also be installed using the following command:

   ```
   sudo apt-get install mysql-server
   ```

4. Create a privileged user and a database:

   • Start MySQL Workbench and connect to the local database
   • Click on “Privileges” and create a new user, with all privileges (Select all). Ideally, create user “msnoise” with password “msnoise”.

5. Install the latest release version of MSNoise:

   ```
   pip install msnoise
   ```

   Power user could install the development version too, but it is not recommended.

6. Check which required packages you are still missing by executing the msnoise bugreport command. (See Testing the Dependencies (page 66))

7. To be sure all is running OK, one could start the msnoise test command in an empty directory. This will start the standard MSNoise test suite, which should end with a “Ran xx tests in yy seconds : OK”.

8. Proceed to the Workflow (page 11) description to start MSNoise!

Done !

1.3 MySQL Server and Workbench

Using the MySQL Server and Workbench is fairly easy and lots of tutorials are available online as text or videos.

Once both are installed, start Workbench and you should see the local MySQL server automatically identified:
And by clicking on “Local Instance ...” another tab should open, connected to the local database.

1.3.1 Create a msnoise user

Select “Users and Privileges” in the left sidebar, then “Add Account”. Define the username and the password (msnoise:msnoise could do, although “weak”):
Then, under “Administrative Roles”, grant this user the *DBA* mode (user can perform all tasks on the database server) and click “Apply”. 
1.3.2 Create an empty database

Each “project” needs a database. That is, if one has two different volcanoes and wants to run MSNoise the two datasets, one needs to create two empty databases.

Click on the “Create new schema” button in the taskbar:

![Create new schema](image)

and give the database a name (for example msnoise; or msnoise_project1, or project1, or else, you choose); and click “Apply”:  

---

1.3. MySQL Server and Workbench
and click “Apply” again and it should state all is OK:
When done, the database we created is present in the left sidebar:

And you’re ready to start your first project: *Workflow* (page 11).
When moving your project to a larger server, HPC or else, just add the connection to this server in Workbench and you’re good to go with the very same interface/tool!

1.4 Database Structure - Tables

MSNoise will create the tables automatically upon running the installer script (see Workflow (page 11)).

1.5 Building this documentation

To build this documentation, some modules are required:

```
pip install sphinx
pip install sphinx_bootstrap_theme
```

Then, this should simply work:

```
make html
```

It will create a .build folder containing the documentation.

You can also build the doc to Latex and then use your favorite Latex-to-PDF tool.

1.6 Using the development version

This is not recommended, but users willing to test the latest development (hopefully stable) version of MSNoise can:

```
pip uninstall msnoise
pip install http://msnoise.org/master.zip
```

Please note this version most probably uses the very latest version of every package: Release versions of numpy, scipy, etc obtained from conda-forge and “master” version of obspy. The development version (master) of obspy can be installed from github:

```
pip uninstall obspy
pip install https://github.com/obspy/obspy/archive/master.zip
```

If you are using the master version, please use the issue tracker of github to communicate about bugs and not the mailing list, preferably used for Releases.
This section only presents the “installation” and configuration of MSNoise (read “the first startup of MSNoise”), not the installation of the required software, which is described in *Installation* (page 3).

### 2.1.1 Installer (initialize Project)

This console script is responsible asking questions about the database connection, to create the db.ini file in order to store the answers and to create the tables in the database.

Questions are:

- What database technology do you want to use?
  - sqlite: this will create a file in the current folder and use it as DB
  - mysql: this will connect to a local or remote mysql server, additional information is then required:
    * hostname: of the mysql server, defaults to 127.0.0.1
    * database: must already exist on *hostname*
    * username: as registered in the privileged users of the mysql server
    * password: his password

The SQLite choice will create a xxx.sqlite file in the current (project) folder, while, for MySQL, one has to create an empty database first on the mysql server, see *how to do this* (page 7).

To run this script:
msnoise install --help

Usage: [OPTIONS]

This command launches the installer.

Options:
--help Show this message and exit.

Warning: The credentials will be saved in a flat text file in the current directory. It’s not very safe, but until now we haven’t thought of another solution.

2.1.2 MSNoise Admin (Web Interface)

MSNoise Admin is a web interface that helps the user define the configuration for all the processing steps, it also allows configuring the stations and filters to be used in the processes. It gives a view on the database tables.

To start the admin:

```
$ msnoise admin
```

Which, by default, starts a web server listening on all interfaces on port 5000. This can be overriden by passing parameters to the command, e.g. for port 5099:

```
$ msnoise admin -p 5099
```

Next step is to open a web browser and open the ip address of the machine, by default on the current machine, it’ll be http://localhost:5000/ or http://127.0.0.1:5000/.

The top level menu shows four items:

Home

The index page shows

- The project location and its database
- Stats of the Data Availability, the CC jobs and the DTT jobs

The name and the logo of the page can be overiden by setting an environment variable with a name and the HTML tag of the logo image:
and then starting msnoise admin:

ROB Dashboard

Project Folder: C:\tmp
Project Database: SQLite: msnoise.sqlite

Configuration

Station

Stations appear as a table and are editable.
Stations are defined as:

```python
class msnoise.msnoise_admin.Station(*args)
    Station Object

    Parameters
    
    - ref (int) – The Station ID in the database
    - net (str) – The network code of the Station
    - sta (str) – The station code
    - X (float) – The X coordinate of the station
    - Y (float) – The Y coordinate of the station
    - altitude (float) – The altitude of the station
    - coordinates (str) – The coordinates system. “DEG” is WGS84 latitude/ longitude in degrees. “UTM” is expressed in meters.
    - instrument (str) – The instrument code, useful with PAZ correction
    - used (bool) – Whether this station must be used in the computations.
```

Filter

Filters appear as a table and are editable. The filter parameters are validated before submission, so no errors should happen. Note: by default, the used parameter is set to False, don’t forget
Filters are defined as:

class msnoise.msnoise_admin.Filter(**kwargs)

Filter base class.

Parameters

- `ref (int)` — The id of the Filter in the database
- `low (float)` — The lower frequency bound of the Whiten function (in Hz)
- `high (float)` — The upper frequency bound of the Whiten function (in Hz)
- `mwcs_low (float)` — The lower frequency bound of the linear regression done in MWCS (in Hz)
- `mwcs_high (float)` — The upper frequency bound of the linear regression done in MWCS (in Hz)
- `rms_threshold (float)` — Not used anymore
- `mwcs_wlen (float)` — Window length (in seconds) to perform MWCS
- `mwcs_step (float)` — Step (in seconds) of the windowing procedure in MWCS
- `used (bool)` — Is the filter activated for the processing

Config

All configuration bits appear as a table and are editable. When editing one configuration item, the Edit tab shows extra information about the parameter, where it is used and its default value. Most of the configuration bits are case-sensitive!

Example view:
The table below repeats this

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_folder</td>
<td>Data Folder</td>
<td></td>
</tr>
<tr>
<td>output_folder</td>
<td>CC Output Folder</td>
<td>CROSS_CORRELATIONS</td>
</tr>
<tr>
<td>data_structure</td>
<td>Either a predefined acronym [SDS]/BUD/IDDS,&lt;br&gt; or /-separated path (e.g. NET/STA/YEAR/NET.STA.YEAR.DAY.MSEED).</td>
<td>SDS</td>
</tr>
<tr>
<td>network</td>
<td>Network to analyse [*]</td>
<td></td>
</tr>
<tr>
<td>channels</td>
<td>Channels need to match the value (ex: [*], <em>Z, BH</em>, HHZ, ...)</td>
<td></td>
</tr>
<tr>
<td>startdate</td>
<td>Start Date to process: [1970-01-01]=’since beginning of the archive’</td>
<td>1970-01-01</td>
</tr>
<tr>
<td>enddate</td>
<td>End Date to process: [2100-01-01]=’No end’</td>
<td>2018-01-01</td>
</tr>
<tr>
<td>analysis_duration</td>
<td>Duration of the Analysis (total in seconds : 3600, [86400])</td>
<td>86400</td>
</tr>
<tr>
<td>cc_sampling_rate</td>
<td>Sampling Rate for the Cross-Correlation [20.0]</td>
<td>20.0</td>
</tr>
<tr>
<td>resampling_method</td>
<td>Resampling method Resample/Decimate/[Lanczos]</td>
<td>Lanczos</td>
</tr>
<tr>
<td>preprocess_lowpass</td>
<td>Preprocessing Low-pass value in Hz [8.0]</td>
<td>8.0</td>
</tr>
<tr>
<td>preprocess_highpass</td>
<td>Preprocessing High-pass value in Hz [0.01]</td>
<td>0.01</td>
</tr>
<tr>
<td>remove_response</td>
<td>Remove instrument response Y/[N]</td>
<td>N</td>
</tr>
<tr>
<td>response_format</td>
<td>Remove instrument file format [dataless]/inventory/paz/resp</td>
<td>dataless</td>
</tr>
<tr>
<td>response_path</td>
<td>Instrument correction file(s) location (path relative to db.ini), defaults to ‘./inventory’, i.e. a subfolder in the current project folder.&lt;br&gt; All files in that folder will be parsed.</td>
<td>inventory</td>
</tr>
<tr>
<td>response_prefilt</td>
<td>Remove instrument correction pre-filter (0.005, 0.006, 30.0, 35.0)</td>
<td>(0.005, 0.006, 30.0, 35.0)</td>
</tr>
<tr>
<td>maxlag</td>
<td>Maximum lag (in seconds) [120.0]</td>
<td>120.</td>
</tr>
</tbody>
</table>
Table 2.1 – continued from previous page

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>corr_duration</td>
<td>Data windows to correlate (in seconds) [1800.]</td>
<td>1800.</td>
</tr>
<tr>
<td>overlap</td>
<td>Amount of overlap between data windows [0:1] [0.]</td>
<td>0.0</td>
</tr>
<tr>
<td>windsorizing</td>
<td>Windsorizing at N time RMS, 0 disables windsorizing, -1 enables 1-bit normalization [3]</td>
<td>3</td>
</tr>
<tr>
<td>whitening</td>
<td>Whiten Traces before cross-correlation: All (except for autocorr). None, or only if components are different [A]/N/C</td>
<td>A</td>
</tr>
<tr>
<td>stack_method</td>
<td>Stack Method: Linear Mean or Phase Weighted Stack: [linear]/pws</td>
<td>linear</td>
</tr>
<tr>
<td>pws_timegate</td>
<td>If stack_method='pws', width of the smoothing in seconds : 10.0</td>
<td>10.0</td>
</tr>
<tr>
<td>pws_power</td>
<td>If stack_method='pws', Power of the Weighting: 2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>crondays</td>
<td>Number of days to monitor with cron [-1], must be a negative number</td>
<td>-1</td>
</tr>
<tr>
<td>components_to_compute</td>
<td>List (comma separated) [ZZ]</td>
<td>ZZ</td>
</tr>
<tr>
<td>autocorr</td>
<td>Compute Auto correlation [Y]/N</td>
<td>N</td>
</tr>
<tr>
<td>keep_all</td>
<td>Keep all cross-corr (length: corr_duration) [Y]/N</td>
<td>N</td>
</tr>
<tr>
<td>keep_days</td>
<td>Keep all daily cross-corr [Y]/N</td>
<td>Y</td>
</tr>
<tr>
<td>ref_begin</td>
<td>Beginning or REF stacks. Can be absolute (2012-01-01) or relative (-100) days</td>
<td>1970-01-01</td>
</tr>
<tr>
<td>ref_end</td>
<td>End or REF stacks. Same as ref_begin</td>
<td>2018-01-01</td>
</tr>
<tr>
<td>mov_stack</td>
<td>Number of days to stack for the Moving-window stacks ([5]= [day-4:day]), can be a comma-separated list 1,2,5,10</td>
<td>5</td>
</tr>
<tr>
<td>export_format</td>
<td>Export stacks in which format(s) ? SAC/MSEED/[BOTH]</td>
<td>MSEED</td>
</tr>
<tr>
<td>sac_format</td>
<td>Format for SAC stacks ? [doublets]/clarke</td>
<td>doublets</td>
</tr>
</tbody>
</table>

Continued on next page
Table 2.1 – continued from previous page

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dtt_lag</code></td>
<td>How is the lag window defined [dynamic]/static</td>
<td>static</td>
</tr>
<tr>
<td><code>dtt_v</code></td>
<td>If <code>dtt_lag=dynamic</code>: what velocity to use to avoid ballistic waves [1.0]km/s</td>
<td>1.0</td>
</tr>
<tr>
<td><code>dtt_minlag</code></td>
<td>If <code>dtt_lag=static</code>: min lag time</td>
<td>5.0</td>
</tr>
<tr>
<td><code>dtt_width</code></td>
<td>Width of the time lag window [30]s</td>
<td>30.0</td>
</tr>
<tr>
<td><code>dtt_sides</code></td>
<td>Which sides to use [both]/left/right</td>
<td>both</td>
</tr>
<tr>
<td><code>dtt_mincoh</code></td>
<td>Minimum coherence on dt measurement, MWCS points with values lower than that will not be used in the WLS</td>
<td>0.65</td>
</tr>
<tr>
<td><code>dtt_maxerr</code></td>
<td>Maximum error on dt measurement, MWCS points with values larger than that will not be used in the WLS</td>
<td>0.1</td>
</tr>
<tr>
<td><code>dtt_maxdt</code></td>
<td>Maximum dt values, MWCS points with values larger than that will not be used in the WLS</td>
<td>0.1</td>
</tr>
<tr>
<td><code>plugins</code></td>
<td>Comma separated list of plugin names. Plugins names should be importable Python modules.</td>
<td></td>
</tr>
</tbody>
</table>

**Database**

**Data Availability**

Gives a view of the `data_availability` table. Allows to bulk edit/select rows. Its main goal is to check that the `scan_archive` procedure has successfully managed to list all files from one’s archive.

**Jobs**

Gives a view of the `jobs` table. Allows to bulk edit/select rows. Its main goal is to check the `new_jobs` or any other workflow step (or Plugins) successfully inserted/updated jobs.
Help

About

Shows some links and information about the package. Mostly the information present on the github readme file.

Bug Report

Web view of the msnoise bugreport -m, allows viewing if all required python modules are properly installed and available for MSNoise.

2.1.3 Populate Station Table

This script is responsible for rapidly scanning the data archive and identifying the Networks/Stations and insert them in the stations table in the database.

The data_folder (as defined in the config) is scanned expecting the data_structure and possible values are defined in data_structures.py:

```python
data_structure['SDS'] = "YEAR.NET/STA.CHAN.TYPE.NET.STA.LOC.CHAN.TYPE.YEAR.DAY"
data_structure['BUD'] = "NET/STA.NET.LOC.CHAN.YEAR.DAY"
data_structure['IDDS'] = "YEAR/NET/STA.CHAN.TYPE.DAY/NET.STA.LOC.CHAN.TYPE.YEAR.DAY.
˓→HOUR"
data_structure['PDF'] = "YEAR/STA.CHAN.TYPE.NET.STA.LOC.CHAN.TYPE.YEAR.DAY"
```

If one's data structure is one of those, then the data_structure configuration bit needs to be set to the acronym (SDS, BUD, IDDS or PDF).

More info on the recommended SDS (“SeisComP Data Structure”) can be found here: https://www.seiscomp3.org/wiki/doc/applications/slarchive/SDS For other simple structures, one has to edit the data_structure configuration (see below).

By default, station coordinates are initialized at 0.

To run this script:

```bash
$ msnoise populate
```

Custom data structure & station table population

If one’s data structure is not one of the pre-defined, it can be defined directly in the data_structure configuration bit using forward slashes, e.g.:

```
data_structure = "NET/STA/YEAR/NET.STA.YEAR.DAY.MSEED"
```

MSNoise expects to find a file named custom.py in the current folder. This python file will contain a function called populate which will accept one argument and return a station dictionary with keys of the format NET_STA , and fields for the stations table in the database: Net,Sta,X,Y,Altitude, Coordinates(UTM/DEG),Instrument.
import os, glob

def populate(data_folder):
    datalist = sorted(glob.glob(os.path.join(data_folder, "*", "*")))
    stationdict = {}
    for di in datalist:
        tmp = os.path.split(di)
        sta = tmp[1]
        net = os.path.split(tmp[0])[1]
        stationdict[net + "+" + sta] = [net, sta, 0.0, 0.0, 0.0, 'UTM', 'N/A']
    return stationdict

Expert (lazy) mode:

If the DataAvailability has already been filled in by another process, for example using the “scan from path” (page 19) procedure, the network/station names can be “populated” from the DataAvailability table automatically. To do this, simply run:

```
msnoise populate --fromDA
```

and MSNoise will insert the unique NET.STA in the Stations table.

2.1.4 Scan Archive

One advantage of MSNoise is its ability to be used as an automated monitoring tool. To run every night on the data acquired during the previous day, MSNoise needs to check the data archive for new or modified files. Those files could have been acquired during the last day, but be data of a previously offline station and contain useful information for, say, a month ago. The time to search for is defined in the config. The scan_archive script uses the find command (gnufind on Windows) with the -mtime argument to locate new or modified files. Once located, they are inserted (if new) or updated (if modified) in the data availability table.

To run the code on two Process, execute the following in console:

```
$ msnoise -t 2 scan_archive
```

Special case: first run

This script is the same as for the routine, but one has to pass the init argument:

```
$ msnoise -t 2 scan_archive --init
```

This will scan the data_archive folder the configured stations and will insert all files found in the data_availability table in the database. As usual, calling the script with a –help argument will show its usage.

Expert (lazy) mode:

Sometimes, you only want to scan a few files and run MSNoise on them. To do this simply run:

```
$ msnoise -t 2 scan_archive --init
```
and MSNoise will read anything ObsPy can (provided the files have a proper header (network code, station code and channel code). Then, once done, simply run the "populate from DataAvailability" (page 19) procedure.

2.1.5 New Jobs

This script searches the database for files flagged “N”ew or “M”odified. For each date in the configured range, it checks if other stations are available and defines the new jobs to be processed. Those are inserted in the jobs table of the database.

To run it from the console:

$ msnoise new_jobs

**Warning:** Upon first run, if you expect the number of jobs to be large (many days, many stations), pass the --init parameter to optimize the insert. Only use this flag once, otherwise problems will arise from duplicate entries in the jobs table.

2.1.6 Compute Cross-Correlations

This code is responsible for the computation of the cross-correlation functions.

This script will group jobs marked “T”odo in the database by day and process them using the following scheme. As soon as one day is selected, the corresponding jobs are marked “I”n Progress in the database. This allows running several instances of this script in parallel.

**Configuration Parameters**

- **cc_sampling_rate**: Sampling Rate for the CrossCorrelation [20.0] (default=20.0)
- **analysis_duration**: Duration of the Analysis (total in seconds : 3600, [86400]) (default=86400)
- **overlap**: Amount of overlap between data windows [0:1] [0.] (default=0.0)
- **maxlag**: Maximum lag (in seconds) [120.0] (default=120.)
- **corr_duration**: Data windows to correlate (in seconds) [1800.] (default=1800.)
- **windsorizing**: Windsorizing at N time RMS , 0 disables windsorizing, -1 enables 1-bit normalization [3] (default=3)
- **resampling_method**: Resampling method Resample/Decimate/[Lanczos] (default=Lanczos)
- **remove_response**: Remove instrument response Y/[N] (default=N)
- **response_format**: Remove instrument file format [dataless]/inventory/paz/resp (default=dataless)
- `response_path`: Instrument correction file(s) location (path relative to db.ini), defaults to './inventory', i.e. a subfolder in the current project folder. All files in that folder will be parsed. (default=inventory)

- `response_prefilt`: Remove instrument correction pre-filter (0.005, 0.006, 30.0, 35.0) (default=(0.005, 0.006, 30.0, 35.0))

- `preprocess_lowpass`: Preprocessing Low-pass value in Hz [8.0] (default=8.0)

- `preprocess_highpass`: Preprocessing High-pass value in Hz [0.01] (default=0.01)

- `keep_all`: Keep all cross-corr (length: corr_duration) [Y]/N (default=N)

- `keep_days`: Keep all daily cross-corr [Y]/N (default=Y)

- `stack_method`: Stack Method: Linear Mean or Phase Weighted Stack: [linear]/pws (default=linear)

- `pws_timegate`: If stack_method='pws', width of the smoothing in seconds : 10.0 (default=10.0)

- `pws_power`: If stack_method='pws', Power of the Weighting: 2.0 (default=2.0)

- `whitening`: Whiten Traces before cross-correlation: All (except for autocorr), None, or only if components are different [A]/N/C (default=A) | new in 1.5

### Waveform Pre-processing

Pairs are first split and a station list is created. The database is then queried to get file paths. For each station, all files potentially containing data for the day are opened. The traces are then merged and splitted, to obtain the most continuous chunks possible. The different chunks are then demeaned, tapered and merged again to a 1-day long trace. If a chunk is not aligned on the sampling grid (that is, start at a integer times the sample spacing in s) , the chunk is phase-shifted in the frequency domain. This requires tapering and fft/ifft. If the gap between two chunks is small, compared to a currently hard-coded value (10 samples), the gap is filled with interpolated values. Larger gaps will not be filled with interpolated values.

**Warning:** As from MSNoise 1.5, traces are no longer padded by or merged with 0s.

Each 1-day long trace is then low-passed (at `preprocess_lowpass` Hz), high-passed (at `preprocess_highpass` Hz), then if needed, decimated/downsampled. Decimation/Downsampling are configurable (resampling_method) and users are advised testing Decimate. One advantage of Downsampling over Decimation is that it is able to downsample the data by any factor, not only integer factors. Downsampling can be achieved with the new ObsPy Lanczos resampler, giving results similar to those by scikits.samplerate.

**Note:** Python 3 users will most probably struggle installing scikits.samplerate, and therefore will have to use either Decimate or Lanczos instead of Resample. This is not a problem because the Lanczos resampling gives results similar to those by scikits.samplerate.

If configured, each 1-day long trace is corrected for its instrument response. Currently, only dataless seed and inventory XML are supported.
As from MSNoise 1.5, the preprocessing routine is separated from the compute_cc and can be used by plugins with their own parameters. The routine returns a Stream object containing all the traces for all the stations/components.

**Processing**

Once all traces are preprocessed, station pairs are processed sequentially. If a component different from ZZ is to be computed, the traces are first rotated. This supposes the user has provided the station coordinates in the station table. The rotation is computed for Radial and Transverse components.

Then, for each corr_duration window in the signal, and for each filter configured in the database, the traces are clipped to windsorizing times the RMS (or 1-bit converted) and then whitened in the frequency domain (see Whitening (page 52)) between the frequency bounds. The whitening procedure can be skipped by setting the whitening configuration to None. The two other whitening modes are “[A]ll except for auto-correlation” or “Only if [C]omponents are different”. This allows skipping the whitening when, for example, computing ZZ components for very close by stations (much closer than the wavelength sampled), leading to spatial autocorrelation issues.

When both traces are ready, the cross-correlation function is computed (see Correlation (page 53)). The function returned contains data for time lags corresponding to maxlag in the acausal (negative lags) and causal (positive lags) parts.

**Stacking and Saving Results**

If configured (setting keep_all to ‘Y’), each corr_duration CCF is saved to the hard disk. By default, the keep_days setting is set to True and so “N = 1 day / corr_duration” CCF are stacked and saved to the hard disk in the STACKS/001_DAYS folder.

**Note:** Currently, the keep-all data (every CCF) are not used by next steps.

If stack_method is ‘linear’, then a simple mean CCF of all windows is saved as the daily CCF. On the other hand, if stack_method is ‘pws’, then all the Phase Weighted Stack (PWS) is computed and saved as the daily CCF. The PWS is done in two steps: first the mean coherence between the instataneous phases of all windows is calculated, and eventually serves a weighting factor on the mean. The smoothness of this weighting array is defined using the pws_timegate parameter in the configuration. The weighting array is the power of the mean coherence array. If pws_power is equal to 0, a linear stack is done (then it’s faster to do set stack_method = ‘linear’). Usual value is 2.

**Warning:** PWS is largely untested, not cross-validated. It looks good, but that doesn’t mean a lot, does it? Use with Caution! And if you cross-validate it, please let us know!!


Once done, each job is marked “D”one in the database.

To run this script:
$ msnoise compute_cc

This step also supports parallel processing/threading:

$ msnoise -t 4 compute_cc

will start 4 instances of the code (after 1 second delay to avoid database conflicts). This works both with SQLite and MySQL but be aware problems could occur with SQLite.

New in version 1.4: The Instrument Response removal & The Phase Weighted Stack & Parallel Processing

New in version 1.5: The Obspy Lanczos resampling method, gives similar results as the scikit-its.samplerate package, thus removing the requirement for it. This method is defined by default.

New in version 1.5: The preprocessing routine is separated from the compute_cc and can be called by external plugins.

2.1.7 Stack

MSNoise is capable of using a reference function defined by absolute or relative dates span. For example, an absolute range could be “from 1 January 2010 to 31 December 2011” and a relative range could be “the last 200 days”. In the latter case, the REF will need to be exported at every run, meaning the following steps (MWCS and DTT) will be executed on the whole configured period. If the REF is defined between absolute dates, excluding “today”, the MWCS and DTT will only be calculated for new data (e.g. “yesterday” and “today”). The corresponding configuration bits are ref_begin and ref_end. In the future, we plan on allowing multiple references to be defined.

Only data for new/modified dates need to be exported. If any CC-job has been marked “Done” within the last day, the stacks will be calculated and a new DTT job will be inserted in the database. For dates in the period of interest, the moving-window stack will only be exported if new/modified CCF is available. The export directory are “REF/” and “DAY%03i/” where %03i will be replaced by the number of days stacked together (DAYS_005 for a 5-days stack, e.g.).

Please note that within MSNoise, stacks are always inclusive of the time/day mentionned. For example, a 5-days stack on January 10, will contain cross-correlation functions computed for January 6, 7, 8, 9 AND 10! The graphical representation centered on a “January 10” tick might then display changes in the CCF that occurred on the 10th!

Moving-window stack length(s) are configured using the mov_stack bit.

If stack_method is ‘linear’, then a simple mean CFF of all daily is saved as the mov or ref CCF. On the other hand, if stack_method is ‘pws’, then all the Phase Weighted Stack (PWS) is computed and saved as the mov or ref CCF. The PWS is done in two steps: first the mean coherence between the instantaneous phases of all windows is calculated, and eventually serves a weighting factor on the mean. The smoothness of this weighting array is defined using the pws_timegate parameter in the configuration. The weighting array is the power of the mean coherence array. If pws_power is equal to 0, a linear stack is done (then it’s faster to do set stack_method = ‘linear’). Usual value is 2.
**Warning:** PWS is largely untested, not cross-validated. It looks good, but that doesn’t mean a lot, does it? Use with Caution! And if you cross-validate it, please let us know!!


### Configuration Parameters

- **ref_begin**: Beginning or REF stacks. Can be absolute (2012-01-01) or relative (-100) days (default=1970-01-01)
- **ref_end**: End or REF stacks. Same as ref_begin (default=2018-01-01)
- **mov_stack**: Number of days to stack for the Moving-window stacks ([5]=[day-4:day]), can be a comma-separated list 1,2,5,10 (default=5)
- **stack_method**: Stack Method: Linear Mean or Phase Weighted Stack: [linear]/pws (default=linear) | new in 1.4
- **pws_timegate**: If stack_method=’pws’, width of the smoothing in seconds : 10.0 (default=10.0) | new in 1.4
- **pws_power**: If stack_method=’pws’, Power of the Weighting: 2.0 (default=2.0) | new in 1.4

### Usage:

The best way to call this code is to start it from the console (-h shows the help)

```bash
$ msnoise stack --help
```

Usage: msnoise-script.py stack [OPTIONS]

Stacks the [REF] and/or [MOV] windows

Options:

- `-r`, `--ref` Compute the REF Stack
- `-m`, `--mov` Compute the MOV Stacks
- `-s`, `--step` Compute the STEP Stacks
- `-i`, `--interval INTEGER` Number of days before now to search for modified Jobs
- `--help` Show this message and exit.

On a routine basis, one should thus run the following to compute REF and MOV stacks:

```bash
$ msnoise stack -r -m
```

While, when playing around with data, and surely on the first run, one should define the `-i INTERVAL`, as jobs might have been marked “Done” more than 24 hours before running the stack. This, for example, will tell the code to search for jobs marked in the last 10 days:

```bash
$ msnoise stack -r -m -i 10
```

New in version 1.4: The Phase Weighted Stack.
2.1.8 Compute MWCS

**Warning:** if using only mov_stack = 1, no DTT jobs is inserted in the database and consequently, no MWCS calculation will be done! FIX!

Following Clarke et al (2011), we apply the *Moving-Window Cross-Spectral method* (page 53) to study the relative dephasing between Moving-Window stacks (“Current”) and a Reference using Moving-Window Cross-Spectral analysis. The jobs “T”o do have been inserted in the database during the stack procedure.

**Filter Configuration Parameters**

- **mwcs_low**: The lower frequency bound of the linear regression done in MWCS (in Hz)
- **mwcs_high**: The upper frequency bound of the linear regression done in MWCS (in Hz)
- **mwcs_wlen**: Window length (in seconds) to perform MWCS
- **mwcs_step**: Step (in seconds) of the windowing procedure in MWCS

In short, both time series are sliced in several overlapping windows and preprocessed. The similarity of the two time-series is assessed using the cross-coherence between energy densities in the frequency domain. The time delay between the two cross correlations is found in the unwrapped phase of the cross spectrum and is linearly proportional to frequency. This “Delay” for each window between two signals is the slope of a weighted linear regression (WLS) of the samples within the frequency band of interest.

For each filter, the frequency band can be configured using `mwcs_low` and `mwcs_high`, and the window and overlap lengths using `mwcs_wlen` and `mwcs_overlap`.

The output of this process is a table of delays measured at each window in the functions. The following is an example for lag times between -115 and -90. In this case, the window length was 10 seconds with an overlap of 5 seconds.

<table>
<thead>
<tr>
<th>LAG_TIME</th>
<th>DELAY</th>
<th>ERROR</th>
<th>MEAN</th>
<th>COHERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.1500000000e+02</td>
<td>-1.4781146383e-01</td>
<td>5.3727119135e-02</td>
<td>2.7585243911e-01</td>
<td></td>
</tr>
<tr>
<td>-1.1000000000e+02</td>
<td>-6.8207526992e-02</td>
<td>2.054644311e-02</td>
<td>3.16209352e-01</td>
<td></td>
</tr>
<tr>
<td>-1.0500000000e+02</td>
<td>-1.0337029577e-01</td>
<td>8.6645155402e-03</td>
<td>4.239269880e-01</td>
<td></td>
</tr>
<tr>
<td>-1.0000000000e+02</td>
<td>-2.8668775696e-02</td>
<td>6.252215988e-03</td>
<td>5.7159849528e-01</td>
<td></td>
</tr>
<tr>
<td>-9.5000000000e+01</td>
<td>4.1803941008e-02</td>
<td>1.5102285789e-02</td>
<td>4.1238577892e-01</td>
<td></td>
</tr>
<tr>
<td>-9.0000000000e+01</td>
<td>4.8139400233e-02</td>
<td>3.2700657018e-02</td>
<td>3.0586187792e-01</td>
<td></td>
</tr>
</tbody>
</table>

This process is job-based, so it is possible to run several instances in parallel.

To run this step:

```
$ msnoise compute_mwcs
```

This step also supports parallel processing/threading:

```
$ msnoise -t 4 compute_mwcs
```

will start 4 instances of the code (after 1 second delay to avoid database conflicts). This works both with SQLite and MySQL but be aware problems could occur with SQLite.
New in version 1.4: Parallel Processing

2.1.9 Compute dt/t

This code is responsible for the calculation of dt/t using the result of the MWCS calculations.

**Warning:** Previously, all pairs were analysed using the same parameters, which were hard-coded in the s06compute_dtt.py file. This has changed now, and MSNoise uses parameters set in the database via the configurator. Pre-1.3 users should upgrade their database using the "$ msnoise upgrade db" command.

**Configuration Parameters**

- **dtt_lag**: How is the lag window defined [dynamic]/static (default=static)
- **dtt_v**: If dtt_lag=dynamic: what velocity to use to avoid ballistic waves [1.0]km/s (default=1.0)
- **dtt_minlag**: If dtt_lag=static: min lag time (default=5.0)
- **dtt_width**: Width of the time lag window [30]s (default=30.0)
- **dtt_sides**: Which sides to use [both]/left/right (default=both)
- **dtt_mincoh**: Minimum coherence on dt measurement, MWCS points with values lower than that will not be used in the WLS (default=0.65)
- **dtt_maxerr**: Maximum error on dt measurement, MWCS points with values larger than that will not be used in the WLS (default=0.1)
- **dtt_maxdt**: Maximum dt values, MWCS points with values larger than that will not be used in the WLS (default=0.1)

The dt/t is determined as the slope of the delays vs time lags. The slope is calculated a weighted linear regression (WLS) through selected points.

1. The selection of points is first based on the time lag criteria. The minimum time lag can either be defined absolutely or dynamically. When dtt_lag is set to “dynamic” in the database, the inter-station distance is used to determine the minimum time lag. This lag is calculated from the distance and a velocity configured (dtt_v). The velocity is determined by the user so that the minlag doesn’t include the ballistic waves. For example, if ballistic waves are visible with a velocity of 2 km/s, one could configure dtt_v=1.0. This way, if stations are located 15 km apart, the minimum lag time will be set to 15 s. The dtt_width determines the width of the lag window used. A value of 30.0 means the process will use time lags between 15 and 45 s in the example above, on both sides if configured (dtt_sides), or only causal or acausal parts of the CCF. The following figure shows the static time lags of dtt_width = 40s starting at dtt_minlag = 10s and the dynamic time lags for a dtt_v = 1.0 km/s for the Piton de La Fournaise network (including stations not on the volcano).

**Note:** It seems obvious that these parameters are frequency-dependent, but they are currently common for all filters!
In order to use the dynamic time lags, one has to provide the station coordinates.

2. Using example values above, we chose to use only 15-45 s coda part of the signal, neglecting direct waves in the 0-15 seconds range. We then select data which match three other thresholds: `dtt_mincoh`, `dtt_maxerr` and `dtt_maxdt`.

Each of the 4 left subplot of this figure shows a colormapper matrix of which each row corresponds to the data of 1 station pair and each column corresponds to different time lags. The cells are then colored using, from left to right: Delays, Errors, Phase Coherence and Data Selection.

Once data (cells) have been selected, they are analyzed two times: first using a WLS that is forced to pass the origin (0,0) and second when a constant is added to allow for the WLS to be offset from the origin. For each value, the error is computed and stored. M0 and EM0 are the slope and its error for the first WLS, and M, EM together with A and EA are the slope, its error, the constant and its error for the second WLS. The output of this calculation is a table, with one row for each station pair.

<table>
<thead>
<tr>
<th>Date</th>
<th>A</th>
<th>EA</th>
<th>EM</th>
<th>EMO</th>
<th>M</th>
<th>MO</th>
<th>Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-01-06</td>
<td>-0.1683728</td>
<td>0.0526606</td>
<td>0.00208377</td>
<td>0.00096521</td>
<td>0.00682021</td>
<td>0.00037757</td>
<td>BE_GES_→BE_HOU</td>
</tr>
<tr>
<td>2013-01-06</td>
<td>-0.0080464</td>
<td>0.0577936</td>
<td>0.00291327</td>
<td>0.00097298</td>
<td>-0.00226910</td>
<td>-0.00264354</td>
<td>BE_GES_→BE_MEM</td>
</tr>
<tr>
<td>2013-01-06</td>
<td>0.1007472</td>
<td>0.0144648</td>
<td>0.00179566</td>
<td>0.00454172</td>
<td>-0.00145738</td>
<td>0.00741478</td>
<td>BE_GES_→BE_RCHB</td>
</tr>
<tr>
<td>2013-01-06</td>
<td>-0.0556811</td>
<td>0.0098926</td>
<td>0.00057839</td>
<td>0.00108102</td>
<td>-0.00328965</td>
<td>-0.00136075</td>
<td>BE_GES_→BE_SKQ</td>
</tr>
<tr>
<td>2013-01-06</td>
<td>0.0150866</td>
<td>0.0202243</td>
<td>0.00096543</td>
<td>0.00089832</td>
<td>0.00083714</td>
<td>0.00104507</td>
<td>BE_GES_→BE_STI</td>
</tr>
<tr>
<td>2013-01-06</td>
<td>0.0268309</td>
<td>0.0328997</td>
<td>0.00153137</td>
<td>0.00150261</td>
<td>0.00302331</td>
<td>0.00302451</td>
<td>BE_GES_→BE_UCC</td>
</tr>
<tr>
<td>2013-01-06</td>
<td>-0.0121293</td>
<td>0.0043351</td>
<td>0.00039019</td>
<td>0.00041347</td>
<td>0.00025836</td>
<td>-0.00042709</td>
<td>BE_HOU_→BE_MEM</td>
</tr>
<tr>
<td>2013-01-06</td>
<td>0.1076247</td>
<td>0.0188662</td>
<td>0.00076824</td>
<td>0.00216383</td>
<td>-0.00030791</td>
<td>0.00112692</td>
<td>BE_HOU_→BE_RCHB</td>
</tr>
<tr>
<td>2013-01-06</td>
<td>-0.0468485</td>
<td>0.0194492</td>
<td>0.00069968</td>
<td>0.00078207</td>
<td>-0.00066133</td>
<td>0.00027102</td>
<td>BE_HOU_→BE_SKQ</td>
</tr>
<tr>
<td>2013-01-06</td>
<td>0.0203057</td>
<td>0.0161316</td>
<td>0.00131522</td>
<td>0.00131182</td>
<td>0.00051626</td>
<td>-3.10306611</td>
<td>BE_HOU_→BE_STI</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2013-01-06</td>
<td>-0.0022588</td>
<td>0.00037141</td>
<td>0.000010340</td>
<td>9.1996e-05</td>
<td>0.00073635</td>
<td>0.00076238</td>
<td>ALL</td>
</tr>
</tbody>
</table>
To run this script:

```
msnoise compute_dtt
```

**Grouping Station Pairs**

Although not clearly visible on the figure above, the very last row of the matrix doesn’t contain information about one station pair, but contains a weighted mean of all delays (from all pairs) for each time lag. For each time lag, delays from each pair is taken into account if it satisfies the same criteria as for the individual data selection. Once the last row (the ALL line) has been calculated, it goes through the normal process of the double WLS and is saved to the output file, as visible above. In the future, MSNoise will be able to treat as many groups as the user want, allowing, e.g. a “crater” and a “slopes” groups.

**Mean of All Pairs vs Mean Pair**

The dt/t calculated using the mean pair (ALL, in red on subplots 4 and 5) and by calculating the weighted mean of the dt/t of all pairs (in green) don’t show a significant difference. The standard deviation around the latter is more spread than on the former, but this has to be investigated.

**Forcing vs No Forcing through Origin**

The reason for allowing the WLS to cross the axis elsewhere than on (0,0) is, for example, to study the potential clock drifts or noise source position variations.
3.1 Plotting

MSNoise comes with some default plotting tools.

All plotting commands accept the `--outfile` argument. If provided, the figure will be saved to the disk. Names can be explicit, or tell the code to generate the filename automatically (using the `?` question mark), for example:

```
# automatic naming, save to PNG
msnoise plot dvv -o ?.png

# automatic naming, save to PDF
msnoise plot dvv -o ?.pdf

# explicit naming, save to JPG
msnoise plot dvv -o mydvv.jpg
```

3.2 Customizing Plots

All plots commands can be overridden using a `-c` argument *in front of the plot command* !!

Examples:

- `msnoise -c plot distance`
- `msnoise -c plot ccftime YA.UV02 YA.UV06 -m 5`
- etc.

To make this work, one has to copy the plot script from the msnoise install directory to the project directory (where your db.ini file is located, then edit it to one's desires. The first thing to edit in the code is the import of the *MSNoise API* (page 41):

```
from ..api import *
```

to

```
from msnoise.api import *
```

and it should work.

New in version 1.4.
3.3 Data Availability Plot

Plots the data availability, as contained in the database. Every day which has a least some data will be coloured in red. Days with no data remain blank.

```bash
msnoise plot data_availability --help
Usage: [OPTIONS]

Plots the Data Availability vs time

Options:
- s, --show BOOLEAN Show interactively?
- o, --outfile TEXT Output filename (=auto)
--help Show this message and exit.
```

Example:

```bash
msnoise plot data_availability:
```

3.4 Station Map

This plots a very raw station map (needs improvement). This plot requires cartopy!

```bash
msnoise plot station_map --help
Usage: [OPTIONS]

Plots the station map (very very basic)
```
Options:
-s, --show BOOLEAN Show interactively?
-o, --outfile TEXT Output filename (=?=auto)
--help Show this message and exit.

Example:

msnoise plot station_map:

![Station map]

It will also generate a HTML file showing the stations on the Leaflet Mapping Service:

New in version 1.4: | Thanks to A. Mordret!

### 3.5 Interferogram Plot

This plot shows the cross-correlation functions (CCF) vs time in a very similar manner as on the `ccftime` plot above, but shows an image instead of wiggles. The parameters allow to plot the daily or the mov-stacked CCF. Filters and components are selectable too. Passing `--refilter` allows to bandpass filter CCFs before plotting (new in 1.5).

```bash
msnoise plot interferogram --help
```

Usage: [OPTIONS] STA1 STA2

Plots the interferogram between sta1 and sta2 (parses the CCFs)
STA1 and STA2 must be provided with this format: NET.STA!

Options:
- `f, --filterid INTEGER` Filter ID
- `c, --comp TEXT` Components (ZZ, ZR,...)
- `m, --mov_stack INTEGER` Mov Stack to read from disk
- `s, --show BOOLEAN` Show interactively?
- `o, --outfile TEXT` Output filename (=auto)
- `r, --refilter TEXT` Refilter CCFs before plotting (e.g. 4:8 for filtering CCFs between 4.0 and 8.0 Hz. This will update the plot title.
- `--help` Show this message and exit.

Example:

msnoise plot interferogram YA.UV06 YA.UV10 -m5 will plot the ZZ component (default), filter 1 (default) and mov_stack 5:

This plot shows the cross-correlation functions (CCF) vs time. The parameters allow to plot the daily or the mov-stacked CCF. Filters and components are selectable too. The `--ampli` argument allows to increase the vertical scale of the CCFs. The `--seismic` shows the up-going wiggles with a black-filled background (very heavy!). Passing `--refilter` allows to bandpass filter CCFs before plotting (new in 1.5).

3.6 CCF vs Time

Usage: `[OPTIONS] STA1 STA2`

Plots the ccf vs time between sta1 and sta2 (parses the dt/t results)
STA1 and STA2 must be provided with this format: NET.STA

Options:
- \(-f\), --filterid INTEGER  Filter ID
- \(-c\), --comp TEXT  Components (ZZ, ZR,...)
- \(-m\), --mov_stack INTEGER  Mov Stack to read from disk
- \(-a\), --ampli FLOAT Amplification
- \(-S\), --seismic Seismic style
- \(-s\), --show BOOLEAN Show interactively?
- \(-o\), --outfile TEXT Output filename (=auto)
- \(-e\), --envelope Plot envelope instead of time series
- \(-r\), --refilter TEXT Refilter CCFs before plotting (e.g. 4:8 for filtering CCFs between 4.0 and 8.0 Hz. This will update the plot title.

--help  Show this message and exit.

Example:

msnoise plot ccftime ID.KWUI ID.POSI will plot all defaults:

3.7 MWCS Plot

This plot shows the result of the MWCS calculations in two superposed images. One is the dt calculated vs time lag and the other one is the coherence. The image is constructed by horizontally stacking the MWCS of different days. The two right panels show the mean and standard deviation per time lag of the whole image. The selected time lags for the dt/t calculation are presented with green horizontal lines, and the minimum coherence or the maximum dt are in red.

The filterid, comp and mov_stack allow filtering the data used.
msnoise plot mwcs --help

Usage: [OPTIONS] STA1 STA2

Plots the mwcs results between sta1 and sta2 (parses the CCFs)

STA1 and STA2 must be provided with this format: NET.STA

Options:
- -f, --filterid INTEGER  Filter ID
- -c, --comp TEXT  Components (ZZ, ZR,...)
- -m, --mov_stack INTEGER  Mov Stack to read from disk
- -s, --show BOOLEAN  Show interactively?
- -o, --outfile TEXT  Output filename (=auto)
- --help  Show this message and exit.

Example:

msnoise plot mwcs ID.KWUI ID.POSI -m 3 will plot all defaults with the mov_stack = 3:

3.8 Distance Plot

Plots the REF stacks vs interstation distance. This could help deciding which parameters to use in the dt/t calculation step. Passing --refilter allows to bandpass filter CCFs before plotting (new in 1.5). It is also possible to only draw CCFs for pairs including one station by passing --virtual-pair followed by the desired NET.STA (new in 1.5).
msnoise plot distance --help

Usage: [OPTIONS]

Plots the REFs of all pairs vs distance

Options:
- `--filterid INTEGER` Filter ID
- `--comp TEXT` Components (ZZ, ZR,...)
- `--ampli FLOAT` Amplification
- `--show BOOLEAN` Show interactively?
- `--outfile TEXT` Output filename (?=auto)
- `--refilter TEXT` Refilter CCFs before plotting (e.g. 4:8 for filtering CCFs between 4.0 and 8.0 Hz. This will update the plot title.
- `--virtual-source TEXT` Use only pairs including this station. Format must be NET.STA
- `--help` Show this message and exit.

Example:

msnoise plot distance will plot all defaults:

![dv/v Plot](image)

3.9 dv/v Plot

This plot shows the final output of MSNoise.

msnoise plot dvv --help
Usage: [OPTIONS]

Plots the dv/v (parses the dt/t results)

Individual pairs can be plotted extra using the -p flag one or more times.

Example: msnoise plot dvv -p ID_KWUI_ID_POSI

Example: msnoise plot dvv -p ID_KWUI_ID_POSI -p ID_KWUI_ID_TRWI

Remember to order stations alphabetically!

Options:
- `f`, `--filterid` INTEGER Filter ID
- `c`, `--comp` TEXT Components (ZZ, ZR,...)
- `m`, `--mov_stack` INTEGER Plot specific mov stacks
- `p`, `--pair` TEXT Plot a specific pair
- `A`, `--all` Show the ALL line?
- `M`, `--dttname` TEXT Plot M or M0?
- `s`, `--show` BOOLEAN Show interactively?
- `o`, `--outfile` TEXT Output filename (=auto)
- `--help` Show this message and exit.

Example:

msnoise plot dvv will plot all defaults:
3.10 dt/t Plot

This plots dt (delay time) against t (time lag). It shows the results from the MWCS step, plus the calculated regression lines M0 and M. The errors in the regression lines are also plotted as fainter lines. The time lags used to calculate the regression are shown in blue.

```
msnoise plot dtt --help

Usage: [OPTIONS] STA1 STA2 DAY

Plots a graph of dt against t

STA1 and STA2 must be provided with this format: NET.STA

DAY must be provided in the ISO format: YYYY-MM-DD

Options:
-f, --filterid INTEGER    Filter ID
-c, --comp TEXT           Components (ZZ, ZR,...)
-m, --mov_stack INTEGER   Mov Stack to read from disk
-s, --show BOOLEAN        Show interactively?
-o, --outfile TEXT        Output filename (=auto)
--help                    Show this message and exit.
```

Example

```
msnoise plot dtt Z7.HRIM Z7.LIND 2014-08-10 -f 14 -m 20 will plot:
```

![Graph of dt/t Plot](image)
New in version 1.4: (Thanks to C.G. Donaldson)
4.1 MSNoise API

**msnoise.api.get_tech()**
Returns the current DB technology used (reads from the db.ini file)

- **Return type**: int
- **Returns**: The database technology used: 1=sqlite 2=mysql

**msnoise.api.get_engine(inifile=None)**
Returns the a SQLAlchemy Engine

- **Parameters**
  - `inifile (str)` – The path to the db.ini file to use. Defaults to `os.getcwd() + db.ini`

- **Return type**: sqlalchemy.engine.Engine
- **Returns**: An Engine Object

**msnoise.api.connect(inifile=None)**
Establishes a connection to the database and returns a Session object.

- **Parameters**
  - `inifile (str)` – The path to the db.ini file to use. Defaults to `os.getcwd() + db.ini`

- **Return type**: sqlalchemy.orm.session.Session
- **Returns**: A Session object, needed for many of the other API methods.

**msnoise.api.create_database(inifile=tech, hostname, database, username, password)**
Creates the db.ini file based on supplied parameters.

- **Parameters**
  - `tech (int)` – The database technology used: 1=sqlite 2=mysql
  - `hostname (str)` – The hostname of the server (if tech=2) or the name of the sqlite file if tech=1
  - `database (str)` – The database name
  - `username (str)` – The user name
  - `password (str)` – The password of user

- **Returns**: None
msnoise.api.read_database_inifile(inifile=None)
Reads the parameters from the db.ini file.

**Parameters**

- `inifile (string)` – The path to the db.ini file to use. Defaults to os.getcwd() + db.ini

**Return type** tuple

**Returns** tech, hostname, database, username, password

msnoise.api.get_config(session, name=None, isbool=False, plugin=None)
Get the value of one or all config bits from the database.

**Parameters**

- `session (sqlalchemy.orm.session.Session)` – A Session object, as obtained by `connect()` (page 41)
- `name (str)` – The name of the config bit to get. If omitted, a dictionary with all config items will be returned
- `isbool (bool)` – if True, returns True/False for config name. Defaults to False
- `plugin (str)` – if provided, gives the name of the Plugin config to use. E.g. if “Amazing” is provided, MSNoise will try to load the “AmazingConfig” entry point. See *Extending MSNoise with Plugins* (page 57) for details.

**Return type** str, bool or dict

**Returns** the value for name or a dict of all config values

msnoise.api.update_config(session, name, value)
Update one config bit in the database.

**Parameters**

- `session (sqlalchemy.orm.session.Session)` – A Session object, as obtained by `connect()` (page 41)
- `name (str)` – The name of the config bit to set.
- `value (str)` – The value of parameter name. Can also be NULL if you don’t want to use this particular parameter.

msnoise.api.get_filters(session, all=False)
Get Filters from the database.

**Parameters**

- `session (sqlalchemy.orm.session.Session)` – A Session object, as obtained by `connect()` (page 41)
- `all (bool)` – Returns all filters from the database if True, or only filters where used = 1 if False (default)

**Return type** list of Filter (page 55)

**Returns** a list of Filter
msnoise.api.update_filter(session, ref, low, mwcs_low, high, mwcs_high,
                         rms_threshold, mwcs_wlen, mwcs_step, used)
Updates or Insert a new Filter in the database.

See also:
msnoise.msnoise_table_def.Filter (page 55)

Parameters

- **session** *(sqlalchemy.orm.session.Session)* – A Session object, as obtained by connect() (page 41)
- **ref** *(int)* – The id of the Filter in the database
- **low** *(float)* – The lower frequency bound of the Whiten function (in Hz)
- **high** *(float)* – The upper frequency bound of the Whiten function (in Hz)
- **rms_threshold** *(float)* – Not used anymore
- **mwcs_wlen** *(float)* – Window length (in seconds) to perform MWCS
- **mwcs_step** *(float)* – Step (in seconds) of the windowing procedure in MWCS
- **used** *(bool)* – Is the filter activated for the processing

msnoise.api.get_networks(session, all=False)
Get Networks from the database.

Parameters

- **session** *(sqlalchemy.orm.session.Session)* – A Session object, as obtained by connect() (page 41)
- **all** *(bool)* – Returns all networks from the database if True, or only networks at least one station has used = 1 if False (default)

Return type list of str

Returns a list of network codes

msnoise.api.get_stations(session, all=False, net=None)
Get Stations from the database.

Parameters

- **session** *(sqlalchemy.orm.session.Session)* – A Session object, as obtained by connect() (page 41)
- **all** *(bool)* – Returns all stations from the database if True, or only stations where used = 1 if False (default)
- **net** *(str)* – if set, limits the stations returned to this network

Return type list of msnoise.msnoise_table_def.Station (page 56)

Returns list of Station (page 56)
msnoise.api.get_station(session, net, sta)
Get one Station from the database.

Parameters
- session (sqlalchemy.orm.session.Session) – A Session object, as obtained by `connect()` (page 41)
- net (str) – the network code
- sta (str) – the station code

Return type msnoise.msnoise_table_def.Station (page 56)

Returns a `Station` Object

msnoise.api.update_station(session, net, sta, X, Y, altitude, coordinates='UTM', instrument='N/A', used=1)
Updates or Insert a new Station in the database.

See also:
msnoise.msnoise_table_def.Station (page 56)

Parameters
- session (sqlalchemy.orm.session.Session) – A Session object, as obtained by `connect()` (page 41)
- net (str) – The network code of the Station
- sta (str) – The station code
- X (float) – The X coordinate of the station
- Y (float) – The Y coordinate of the station
- altitude (float) – The altitude of the station
- coordinates (str) – The coordinates system. “DEG” is WGS84 latitude/ longitude in degrees. “UTM” is expressed in meters.
- instrument (str) – The instrument code, useful with PAZ correction
- used (bool) – Whether this station must be used in the computations.

msnoise.api.get_station_pairs(session, used=None, net=None)
Returns an iterator over all possible station pairs. If auto-correlation is configured in the database, returns N*N pairs, otherwise returns N*(N-1)/2 pairs.

Parameters
- session (sqlalchemy.orm.session.Session) – A Session object, as obtained by `connect()` (page 41)
- used (bool, int) – Select only stations marked used if False (default) or all stations present in the database if True
- net (str) – Network code to filter for the pairs.

Return type iterable

Returns An iterable of `Station` (page 56) object pairs
msnoise.api.get_interstation_distance(station1, station2, coordinates='DEG')

Returns the distance in km between station1 and station2.

**Warning:** Currently the stations coordinates system have to be the same!

**Parameters**
- station1 (`Station` (page 56)) – A Station object
- station2 (`Station` (page 56)) – A Station object
- coordinates (`str`) – The coordinates system. “DEG” is WGS84 latitude/ longitude in degrees. “UTM” is expressed in meters.

**Return type** float

**Returns** The interstation distance in km

msnoise.api.update_data_availability(session, net, sta, comp, path, file, starttime, endtime, data_duration, gaps_duration, samplerate)

Updates a DataAvailability object in the database

**Parameters**
- session (`sqlalchemy.orm.session.Session`) – A Session object, as obtained by `connect()` (page 41)
- net (`str`) – The network code of the Station
- sta (`str`) – The station code
- comp (`str`) – The component (channel)
- path (`str`) – The full path to the folder containing the file
- file (`str`) – The name of the file
- starttime (`datetime.datetime`) – Start time of the file
- endtime (`datetime.datetime`) – End time of the file
- data_duration (`float`) – Cumulative duration of available data in the file
- gaps_duration (`float`) – Cumulative duration of gaps in the file
- samplerate (`float`) – Sample rate of the data in the file (in Hz)

msnoise.api.get_new_files(session)

Returns the files marked “N”ew or “M”odified in the database

**Parameters**
- session (`sqlalchemy.orm.session.Session`) – A Session object, as obtained by `connect()` (page 41)

**Return type** list

**Returns** list of `DataAvailability` (page 56)
msnoise.api.get_data_availability(session, net=None, sta=None, comp=None, starttime=None, endtime=None)

Returns the *DataAvailability* (page 56) objects for specific net, sta, starttime or endtime.

**Parameters**

- **session** *(sqlalchemy.orm.session.Session)* – A Session object, as obtained by *connect()* (page 41)
- **net** *(str)* – Network code
- **sta** *(str)* – Station code
- **starttime** *(datetime.datetime, datetime.date)* – Start time of the search
- **endtime** *(datetime.datetime, datetime.date)* – End time of the search

**Return type** list

**Returns** list of *DataAvailability* (page 56)

msnoise.api.mark_data_availability(session, net, sta, flag)

Updates the flag of all *DataAvailability* (page 56) objects matching *net.sta* in the database.

**Parameters**

- **session** *(sqlalchemy.orm.session.Session)* – A Session object, as obtained by *connect()* (page 41)
- **net** *(str)* – Network code
- **sta** *(str)* – Station code
- **flag** *(str)* – Status of the DataAvailability object: New, Modified or Archive. Values accepted are {'N', 'M', 'A'}

msnoise.api.count_data_availability_flags(session)

Count the number of *DataAvailability* (page 56), grouped by flag.

**Parameters**

- **session** *(sqlalchemy.orm.session.Session)* – A Session object, as obtained by *connect()* (page 41)

**Return type** list

**Returns** list of [count, flag] pairs

msnoise.api.update_job(session, day, pair, jobtype, flag, commit=True, return_job=True)

Updates or Inserts a new *Job* (page 55) in the database.

**Parameters**

- **day** *(str)* – The day in YYYY-MM-DD format
- **pair** *(str)* – the name of the pair (EXAMPLE?)
- **jobtype** *(str)* – CrossCorrelation (CC) or dt/t (DTT) Job?
• commit (bool) – Whether to directly commit (True, default) or not (False)
• returnjob (bool) – Return the modified/inserted Job (True, default) or not (False)

Return type Job (page 55) or None

Returns If returnjob is True, returns the modified/inserted Job.

msnoise.api.massive_insert_job(jobs)
Routine to use a low level function to insert much faster a list of Job (page 55). This method uses the Engine directly, no need to pass a Session object.

Parameters jobs (list) – a list of Job (page 55) to insert.

msnoise.api.is_next_job(session, flag='T', jobtype='CC')
Are there any Job (page 55) in the database, with flag='flag' and jobtype='type'

Parameters
• session (sqlalchemy.orm.session.Session) – A Session object, as obtained by connect() (page 41)
• jobtype (str) – CrossCorrelation (CC) or dt/t (DTT) Job?

Return type bool

Returns True if at least one Job (page 55) matches, False otherwise.

msnoise.api.get_next_job(session, flag='T', jobtype='CC')
Get the next Job (page 55) in the database, with flag='flag' and jobtype='jobtype'. Jobs of the same type are grouped per day. This function also sets the flag of all selected Jobs to “T”n progress.

Parameters
• session (sqlalchemy.orm.session.Session) – A Session object, as obtained by connect() (page 41)
• jobtype (str) – CrossCorrelation (CC) or dt/t (DTT) Job?

Return type list

Returns list of Job (page 55)

msnoise.api.is_dtt_next_job(session, flag='T', jobtype='DTT', ref=False)
Are there any DTT Job (page 55) in the database, with flag='flag' and jobtype='jobtype'. If ref is provided, checks if a DTT “REF” job is present.

Parameters
• session (sqlalchemy.orm.session.Session) – A Session object, as obtained by connect() (page 41)
• jobtype (str) – CrossCorrelation (CC) or dt/t (DTT) Job?
- **ref (bool)** – Whether to check for a REF job (True) or not (False, default)

**Return type** bool

**Returns** True if at least one Job matches, False otherwise.

```python
msnoise.api.get_dtt_next_job(session, flag='T', jobtype='DTT')
```

Get the next DTT Job (page 55) in the database, with flag='flag' and jobtype='jobtype'. Jobs are then grouped per station pair. This function also sets the flag of all selected Jobs to “T”n progress.

**Parameters**

- **session (sqlalchemy.orm.session.Session)** – A Session object, as obtained by `connect()` (page 41)
- **jobtype (str)** – CrossCorrelation (CC) or dt/t (DTT) Job?

**Return type** tuple

**Returns** (pairs, days, refs): List of station pair names - Days of the next DTT jobs - Job IDs (for later being able to update their flag).

```python
msnoise.api.reset_jobs(session, jobtype, alljobs=False, rule=None)
```

Sets the flag of all jobtype Jobs to “T”odo.

**Parameters**

- **session (sqlalchemy.orm.session.Session)** – A Session object, as obtained by `connect()` (page 41)
- **jobtype (str)** – CrossCorrelation (CC) or dt/t (DTT) Job?
- **alljobs (bool)** – If True, resets all jobs. If False (default), only resets jobs “I”n progress.

```python
msnoise.api.reset_dtt_jobs(session, pair)
```

Sets the flag of all DTT Jobs of one pair to “T”odo.

**Parameters**

- **session (sqlalchemy.orm.session.Session)** – A Session object, as obtained by `connect()` (page 41)
- **pair (str)** – The pair to update

```python
msnoise.api.get_job_types(session, jobtype='CC')
```

Count the number of Jobs of a specific type, grouped by flag.

**Parameters**

- **session (sqlalchemy.orm.session.Session)** – A Session object, as obtained by `connect()` (page 41)
- **jobtype (str)** – CrossCorrelation (CC) or dt/t (DTT) Job?

**Return type** list

**Returns** list of [count, flag] pairs
msnoise.api.get_jobs_by_lastmod(session, jobtype='CC', last-mod=datetime.datetime(2017, 4, 28, 13, 50, 51, 308000))

msnoise.api.export_allcorr(session, ccfid, data)

msnoise.api.add_corr(session, station1, station2, filterid, date, time, duration, components, CF, sampling_rate, day=False, ncorr=0)

Adds a CCF to the data archive on disk.

Parameters

- **session** (sqlalchemy.orm.session.Session) – A Session object, as obtained by `connect()` (page 41)
- **station1** (str) – The name of station 1 (formatted NET.STA)
- **station2** (str) – The name of station 2 (formatted NET.STA)
- **filterid** (int) – The ID (ref) of the filter
- **date** (datetime.date or str) – The date of the CCF
- **time** (datetime.time or str) – The time of the CCF
- **duration** (float) – The total duration of the exported CCF
- **components** (str) – The name of the components used (ZZ, ZR, ...)
- **sampling_rate** (float) – The sampling rate of the exported CCF
- **day** (bool) – Whether this function is called to export a daily stack (True) or each CCF (when `keep_all` parameter is set to True in the configuration). Defaults to True.
- **ncorr** (int) – Number of CCF that have been stacked for this CCF.

msnoise.api.export_sac(db, filename, pair, components, filterid, corr, ncorr=0, sac_format=None, maxlag=None, cc_sampling_rate=None)

msnoise.api.export_mseed(db, filename, pair, components, filterid, corr, ncorr=0, maxlag=None, cc_sampling_rate=None)

msnoise.api.stack(session, data)

msnoise.api.get_results(session, station1, station2, filterid, components, dates, mov_stack=1, format='stack')

msnoise.api.get_results_all(session, station1, station2, filterid, components, dates, mov_stack=1, format='stack')

msnoise.api.get_maxlag_samples(session)

Returns the length of the CC functions. Gets the maxlag and sampling rate from the database.

Parameters

- **session** (sqlalchemy.orm.session.Session) – A Session object, as obtained by `connect()` (page 41)

Return type int

Returns the length of the CCF
msnoise.api.get_t_axis(session)

Returns the time axis (in seconds) of the CC functions. Gets the maxlag from the database and uses get_maxlag.samples function.

Parameters

- **session** (sqlalchemy.orm.session.Session) – A Session object, as obtained by connect() (page 41)

Return type

numpy.array

Returns the time axis

msnoise.api.get_components_to_compute(session, plugin=None)

Returns the components configured in the database.

Parameters

- **session** (sqlalchemy.orm.session.Session) – A Session object, as obtained by connect() (page 41)

Return type

list of str

Returns a list of components to compute

msnoise.api.build_ref_datelist(session)

Creates a date array for the REF. The returned tuple contains a start and an end date, and a list of individual dates between the two.

Parameters

- **session** (sqlalchemy.orm.session.Session) – A Session object, as obtained by connect() (page 41)

Return type

tuple

Returns (start, end, datelist)

msnoise.api.build_movstack_datelist(session)

Creates a date array for the analyse period. The returned tuple contains a start and an end date, and a list of individual dates between the two.

Parameters

- **session** (sqlalchemy.orm.session.Session) – A Session object, as obtained by connect() (page 41)

Return type

tuple

Returns (start, end, datelist)

msnoise.api.updated_days_for_dates(session, date1, date2, pair, jobtype='CC', interval=datetime.timedelta(1), returndays=False)

Determines if any Job of jobtype='jobtype' and for pair='pair', concerning a date between date1 and date2 has been modified in the last interval='interval'.

Parameters

- **session** (sqlalchemy.orm.session.Session) – A Session object, as obtained by connect() (page 41)
- **date1** (datetime.datetime) – Beginning of the period of interest
- **date2** (datetime.datetime) – End of the period of interest
- **pair** (str) – Pair of interest
- **jobtype** (str) – CrossCorrelation (CC) or dt/t (DTT) Job?
- **interval** (datetime.timedelta) – Interval of time before now to search for updated days
- `returndays (bool)` – Whether to return a list of days (True) or not (False, default)

**Return type** list or bool

**Returns** List of days if returndays is True, only “True” if not. (not clear!)

```python
msnoise.api.azimuth(coordinates, x0, y0, x1, y1)
```

Returns the azimuth between two coordinate sets.

**Parameters**

- `coordinates (str)` – {‘DEG’, ‘UTM’, ‘MIX’}
- `x0 (float)` – X coordinate of station 1
- `y0 (float)` – Y coordinate of station 1
- `x1 (float)` – X coordinate of station 2
- `y1 (float)` – Y coordinate of station 2

**Return type** float

**Returns** The azimuth in degrees

```python
msnoise.api.nextpow2(x)
```

Returns the next power of 2 of `x`.

**Parameters** `x (int)` – any value

**Return type** int

**Returns** the next power of 2 of `x`

```python
msnoise.api.check_and_phase_shift(trace)
```

```python
msnoise.api.getGaps(stream, min_gap=None, max_gap=None)
```

```python
msnoise.api.make_same_length(st)
```

This function takes a stream of equal sampling rate and makes sure that all channels have the same length and the same gaps.

```python
msnoise.api.clean_scipy_cache()
```

This function wraps all destroy scipy cache at once. It is a workaround to the memory leak induced by the “caching” functions in scipy fft.

```python
msnoise.api.linear_regression(xdata, ydata, weights=None, p0=None, intercept_origin=True, **kwargs)
```

Use linear least squares to fit a function, `f`, to data. This method is a generalized version of `scipy.optimize.minpack.curve_fit()`; allowing for Ordinary Least Square and Weighted Least Square regressions:

- OLS through origin: `linear_regression(xdata, ydata)`
- OLS with any intercept: `linear_regression(xdata, ydata, intercept_origin=False)`
- WLS through origin: `linear_regression(xdata, ydata, weights)`
- WLS with any intercept: `linear_regression(xdata, ydata, weights, intercept_origin=False)`
If the expected values of slope (and intercept) are different from 0.0, provide the p0 value(s).

**Parameters**

- **xdata** – The independent variable where the data is measured.
- **ydata** – The dependent data - nominally f(xdata, ...)
- **weights** – If not None, the uncertainties in the ydata array. These are used as weights in the least-squares problem. If None, the uncertainties are assumed to be 1. In SciPy vocabulary, our weights are 1/sigma.
- **p0** – Initial guess for the parameters. If None, then the initial values will all be 0 (Different from SciPy where all are 1)
- **intercept_origin** – If True: solves y=a*x (default); if False: solves y=a*x+b.

**Return type** tuple

**Returns** (slope, std_slope) if intercept_origin is True; (slope, intercept, std_slope, std_intercept) if False.

**msnoise.api.preload_instrument_responses(session)**

This function preloads all instrument responses from response format and stores the seed ids, start and end dates, and paz for every channel in a DataFrame.

**Warning:** This function only works for response_format being “inventory” or “dataless”.

**Parameters**

- **session** (sqlalchemy.orm.session.Session) – A Session object, as obtained by connect() (page 41)

**Return type** pandas.DataFrame

**Returns** A table containing all channels with the time of operation and poles and zeros.

### 4.2 Core Functions

#### 4.2.1 Whitening

**move2obspy.whiten(data, Nfft, delta, freqmin, freqmax, plot=False)**

This function takes 1-dimensional data timeseries array, goes to frequency domain using fft, whitens the amplitude of the spectrum in frequency domain between freqmin and freqmax and returns the whitened fft.

**Parameters**

- **data** (numpy.ndarray) – Contains the 1D time series to whiten
- **Nfft** (int) – The number of points to compute the FFT
- **delta** (float) – The sampling frequency of the data
• freqmin \((float)\) – The lower frequency bound
• freqmax \((float)\) – The upper frequency bound
• plot \((bool)\) – Whether to show a raw plot of the action (default: False)

Return type **numpy.ndarray**

Returns The FFT of the input trace, whitened between the frequency bounds

### 4.2.2 Correlation

**move2obspy.myCorr(data, maxlag, plot=False, nfft=None)**

This function takes ndimensional data array, computes the cross-correlation in the frequency domain and returns the cross-correlation function between \([-maxlag:maxlag]\).

Parameters

• **data** \(\text{numpy.ndarray}\) – This array contains the fft of each timeseries to be cross-correlated.
• **maxlag** \(\text{int}\) – This number defines the number of samples \((N=2*\text{maxlag} + 1)\) of the CCF that will be returned.

Return type **numpy.ndarray**

Returns The cross-correlation function between \([-\text{maxlag}:maxlag]\)

### 4.2.3 Moving-Window Cross-Spectral method

**move2obspy.mwcs(current, reference, freqmin, freqmax, df, tmin, window_length, step, smoothing_half_win=5)**

The current time series is compared to the reference. Both time series are sliced in several overlapping windows. Each slice is mean-adjusted and cosine-tapered (85% taper) before being Fourier-transformed to the frequency domain. \(F_{\text{current}}(\nu)\) and \(F_{\text{reference}}(\nu)\) are the first halves of the Hermitian symmetric Fourier-transformed segments. The cross-spectrum \(X(\nu)\) is defined as

\[
X(\nu) = F_{\text{reference}}(\nu)F_{\text{current}}^*(\nu)
\]

in which \(\ast\) denotes the complex conjugation. \(X(\nu)\) is then smoothed by convolution with a Hanning window. The similarity of the two time-series is assessed using the cross-coherence between energy densities in the frequency domain:

\[
C(\nu) = \frac{|X(\nu)|}{\sqrt{|F_{\text{reference}}(\nu)|^2|F_{\text{current}}(\nu)|^2}}
\]

in which the over-line here represents the smoothing of the energy spectra for \(F_{\text{reference}}\) and \(F_{\text{current}}\) and of the spectrum of \(X\). The mean coherence for the segment is defined as the mean of \(C(\nu)\) in the frequency range of interest. The time-delay between the two cross correlations is found in the unwrapped phase, \(\phi(u)\), of the cross spectrum and is linearly proportional to frequency:

\[
\phi_j = m.u_j, m = 2\pi\delta t
\]

The time shift for each window between two signals is the slope \(m\) of a weighted linear regression of the samples within the frequency band of interest. The weights are those introduced by [Clarke2011], which incorporate both the cross-spectral amplitude
and cross-coherence, unlike [Poupinet1984]. The errors are estimated using the weights (thus the coherence) and the squared misfit to the modelled slope:

\[ e_m = \sqrt{\sum_j \left( \frac{w_j \nu_j}{\sum_i w_i} \right)^2 \sigma_\phi^2} \]

where \( w \) are weights, \( \nu \) are cross-coherences and \( \sigma_\phi^2 \) is the squared misfit of the data to the modelled slope and is calculated as

\[ \sigma_\phi^2 = \frac{\sum_j (\phi_j - m \nu_j)^2}{N-1} \]

The output of this process is a table containing, for each moving window: the central time lag, the measured delay, its error and the mean coherence of the segment.

**Warning:** The time series will not be filtered before computing the cross-spectrum! They should be band-pass filtered around the freqmin-freqmax band of interest beforehand.

### Parameters

- current (`numpy.ndarray`) – The “Current” timeseries
- reference (`numpy.ndarray`) – The “Reference” timeseries
- freqmin (`float`) – The lower frequency bound to compute the dephasing (in Hz)
- freqmax (`float`) – The higher frequency bound to compute the dephasing (in Hz)
- df (`float`) – The sampling rate of the input timeseries (in Hz)
- tmin (`float`) – The leftmost time lag (used to compute the “time lags array”)
- window_length (`float`) – The moving window length (in seconds)
- step (`float`) – The step to jump for the moving window (in seconds)
- smoothing_half_win (`int`) – If different from 0, defines the half length of the smoothing hanning window.

**Return type** `numpy.ndarray`

**Returns** `[time_axis,delta_t,delta_err,delta_mcoh]`. `time_axis` contains the central times of the windows. The three other columns contain dt, error and mean coherence for each window.

### 4.2.4 WLS

`move2obspy.linear_regression(xdata, ydata, weights=None, p0=None, intercept_origin=True, **kwargs)`

Use linear least squares to fit a function, f, to data. This method is a generalized version of `scipy.optimize.minpack.curve_fit()`; allowing for Ordinary Least Square and Weighted Least Square regressions:

- OLS through origin: `linear_regression(xdata, ydata)`
- OLS with any intercept: `linear_regression(xdata, ydata, intercept-origin=False)`
• WLS through origin: `linear_regression(xdata, ydata, weights)`

• WLS with any intercept: `linear_regression(xdata, ydata, weights, intercept_origin=False)`

If the expected values of slope (and intercept) are different from 0.0, provide the p0 value(s).

Parameters

- `xdata` – The independent variable where the data is measured.
- `ydata` – The dependent data - nominally f(xdata, ...)
- `weights` – If not None, the uncertainties in the ydata array. These are used as weights in the least-squares problem. If None, the uncertainties are assumed to be 1. In SciPy vocabulary, our weights are 1/sigma.
- `p0` – Initial guess for the parameters. If None, then the initial values will all be 0 (Different from SciPy where all are 1)
- `intercept_origin` – If True: solves y=a*x (default); if False: solves y=a*x+b.

Extra keyword arguments will be passed to `scipy.optimize.minpack.curve_fit()`.

Return type `tuple`

Returns (slope, std_slope) if `intercept_origin` is True; (slope, intercept, std_slope, std_intercept) if False.

4.3 Table Definitions

class `msnoise.msnoise_table_def.Filter(**kwargs)`

Filter base class.

Parameters

- `ref (int)` – The id of the Filter in the database
- `low (float)` – The lower frequency bound of the Whiten function (in Hz)
- `high (float)` – The upper frequency bound of the Whiten function (in Hz)
- `mwcs_low (float)` – The lower frequency bound of the linear regression done in MWCS (in Hz)
- `mwcs_high (float)` – The upper frequency bound of the linear regression done in MWCS (in Hz)
- `rms_threshold (float)` – Not used anymore
- `mwcs_wlen (float)` – Window length (in seconds) to perform MWCS
- `mwcs_step (float)` – Step (in seconds) of the windowing procedure in MWCS
- `used (bool)` – Is the filter activated for the processing
**Job Object**

Class:

```python
def Job(day, pair, jobtype, flag, last-mod=datetime.datetime(2017, 4, 28, 11, 50, 51, 256000))
```

Parameters:

- ref (int) – The Job ID in the database
- day (str) – The day in YYYY-MM-DD format
- pair (str) – the name of the pair (EXAMPLE?)
- jobtype (str) – CrossCorrelation (CC) or dt/t (DTT) Job?

**Station Object**

Class:

```python
def Station(*args)
```

Parameters:

- ref (int) – The Station ID in the database
- net (str) – The network code of the Station
- sta (str) – The station code
- X (float) – The X coordinate of the station
- Y (float) – The Y coordinate of the station
- altitude (float) – The altitude of the station
- coordinates (str) – The coordinates system. “DEG” is WGS84 latitude/ longitude in degrees. “UTM” is expressed in meters.
- instrument (str) – The instrument code, useful with PAZ correction
- used (bool) – Whether this station must be used in the computations.

**Config Object**

Class:

```python
def Config(name, value)
```

Parameters:

- name (str) – The name of the config bit to set.
- value (str) – The value of parameter name

**DataAvailability Object**

Class:

```python
def DataAvailability(net, sta, comp, path, file, start-time, endtime, data-duration, gaps-duration, samplerate, flag)
```

Parameters:

- ref (int) – The Station ID in the database
- net (str) – The network code of the Station
- sta (str) – The station code
- comp (str) – The component (channel)
- path (str) – The full path to the folder containing the file
• file (str) – The name of the file
• starttime (datetime) – Start time of the file
• endtime (datetime) – End time of the file
• data_duration – Cumulative duration of available data in the file
• gaps_duration (float) – Cumulative duration of gaps in the file
• samplerate (float) – Sample rate of the data in the file (in Hz)
• flag (str) – The status of the entry: “N”ew, “M”odified or “A”rchive

4.4 Extending MSNoise with Plugins

New in version 1.4.

Starting with MSNoise 1.4 (page 86), MSNoise supports Plugins, this means the default workflow “from archive to dv/v” can be branched at any step!

• What is a Plugin and how to declare it in MSNoise (page 57)
• Plugin minimal structure (page 58)
• Declaring Job Types - Hooking (page 59)
• Plugin’s own config table (page 61)
• Adding Web Admin Pages (page 63)
• Uninstalling Plugins (page 64)
• Download Amazing Plugin (page 64)

4.4.1 What is a Plugin and how to declare it in MSNoise

A plugin is a python package, properly structured, that can be imported from msnoise, i.e. it has to be “installed” like any other python package.

After installing a plugin, its package name must be declared in the plugins parameter in the configuration. This must be done PER PROJECT. This configuration field supports a list of plugins, separated by a simple comma (!no space), e.g. msnoise_amazing,msnoise_plugin101.

Once configured in a project, the plugin should appear when calling the msnoise plugin command:

```bash
$ msnoise plugin

Usage: msnoise-script.py plugin [OPTIONS] COMMAND [ARGS]...

Runs a command in a named plugin

Options:
--help  Show this message and exit.
```
Commands:
   amazing   Example Amazing Plugin for MSNoise

4.4.2 Plugin minimal structure

A plugin is a python package, so its minimal structure is:

```
msnoise-amazingplugin
-- __init__.py
-- setup.py
-- msnoise_amazingplugin
   -- __init__.py
   -- plugin_definition.py
```

The `setup.py` declares where the plugin actually hooks into MSNoise:

```
from setuptools import setup, find_packages

setup(
   name='msnoise_amazing',
   version='0.1a',
   packages=find_packages(),
   include_package_data=True,
   install_requires=['msnoise',
                    'obspy'],

   entry_points = {
      'msnoise.plugins.commands': [
         'amazing = msnoise_amazing.plugin_definition:amazing',
      ],
   },

   author = "Thomas Lecocq & MSNoise dev team",
   author_email = "Thomas.Lecocq@seismology.be",
   description = "An example plugin",
   license = "EUPL-1.1",
   url = "http://www.msnoise.org",
   keywords="amazing seismology"
)
```

The most important line of this file is the one declaring the `amazing` entry point in `msnoise.plugins.commands` and linking it to the plugin’s `plugin_definition.py` file.

The content of `plugin_definition.py` must then provide at least one `click.Command`, or more commonly, one `click.Group` and many `click.Command`.

```
import click

@click.group()
def amazing():
    """Example Amazing Plugin for MSNoise""
    pass

@click.command()
def sayhi():
    """A Very Polite Command""
    print("Hi")
```
amazing.add_command(sayhi)

This way, once properly installed and activated (declared in the plugins config), the plugin will be callable from msnoise:

```bash
$ msnoise plugin amazing
Usage: msnoise-script.py plugin amazing [OPTIONS] COMMAND [ARGS]...
Example Amazing Plugin for MSNoise
Options:
    --help  Show this message and exit.
Commands:
    sayhi  A Very Polite Command
```

and its command too:

```bash
$ msnoise plugin amazing sayhi
Hi

Amazing, isn’t it ?
```

4.4.3 Declaring Job Types - Hooking

Plugin-based job types are defined by providing a `register_job_types` method in `plugin_definition.py`. A new job type is defined with two parameters:

- **name**: the actual job name (acronym style) used all over (example: CC2, TEST)
- **after**: when is this job added to the database.

Current supported “after” are:

- **new_files**: will be created when running the `new_jobs` command and will create a job with those parameters (nf is a new file identified in the scan_archive procedure). In this specific case, the `pair` field of the job will only be NET.STA, not a “pair”. A job will only be inserted if the station is “Used” in the configuration.

```python
all_jobs.append({
    "day": current_date,
    "pair": "%s.%s"%(nf.net,nf.sta),
    "jobtype": jobtype,
    "flag": "T",
    "lastmod": datetime.datetime.utcnow()
})
```

- **scan_archive**: will be created when running the `new_jobs` command, in parallel to CC jobs. This is, for example, useful when one wants to compute relative amplitude ratios between station pairs. In this case, the `pair` field of the job is set to the pair name.

- **refstack**: will be created when running the `stack` command and when a new REF stack needed to be calculated. This is, for example, useful when one wants to work on the REF stacks using a Ambient Seismic Noise Tomography code.

Plugin’s Job Types are first declared in `setup.py` (in Entry Points):
Defining job types in `plugin_definition.py`:

```python
def register_job_types():
    jobtypes = []
    jobtypes.append( {"name": "AMAZ1", "after": "new_files"} )
    return jobtypes
```

Then, adding a `compute` command to the `plugin_definition.py`:

```python
@click.command()
def compute():
    """Compute an Amazing Value""
    from .compute import main()
    main()
```

and creating a `compute.py` file in the plugin folder:

```python
import os
from obspy.core import UTCDateTime, read
from msnoise.api import connect, is_next_job, get_next_job, get_data_availability, get_config, update_job

def main():
    db = connect()
    while is_next_job(db, jobtype='AMAZ1'):
        jobs = get_next_job(db, jobtype='AMAZ1')
        for job in jobs:
            net, sta = job.pair.split('.')
            gd = UTCDateTime(job.day).datetime
            print("Processing %s.%s for day %s"%(net, sta, job.day))
            files = get_data_availability(
                db, net=net, sta=sta, starttime=gd, endtime=gd,
                comp="Z")
            for file in files:
                fn = os.path.join(file.path, file.file)
                st = read(fn, starttime=UTCDateTime(job.day), endtime=UTCDateTime(job.day)+86400)
                print(st)
```

Aaaand:

```
$ msnoise plugin amazing compute
Processing YA.UV05 for day 2010-09-01
1 Trace(s) in Stream:
YA.UV05.00.HHZ | 2010-09-01T00:00:00.000000Z - 2010-09-01T23:59:59.990000Z | 100.0 Hz, 8640000 samples
Processing YA.UV06 for day 2010-09-01
1 Trace(s) in Stream:
YA.UV06.00.HHZ | 2010-09-01T00:00:00.000000Z - 2010-09-01T23:59:59.990000Z | 100.0 Hz, 8640000 samples
Processing YA.UV10 for day 2010-09-01
```
Provided you have reset the DataAvailability rows with a “M” or “N” flag so that when you run new jobs it actually inserted the AMAZ1 jobs!

Because job-based stuff always requires a lot of trial-and-error, remember that the `msnoise reset` command is your best friend. In this example, we would need to `msnoise reset AMAZ1` to reset “T”n Progress jobs, or `msnoise reset AMAZ1 --all` to reset all AMAZ1 jobs to “T”o Do.

**Note:**

- Currently, not all MSNoise workflow steps use the `is_next_job - get_next_job` logic, but it’ll be the case for MSNoise 1.5
- Only three hooks are currently present, of course, more will be added in in the future.

### 4.4.4 Plugin’s own config table

Plugins can create a new table in the database, e.g. in an `install` command. First, a `amazing_table_def.py` table definition file must be created:

```python
# Table definitions for Amazing
from sqlalchemy import Column, String
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class AmazingConfig(Base):
    """
    Config Object
    """
    :type name: str
    :param name: The name of the config bit to set.

    :type value: str
    :param value: The value of parameter `name`
    """
    __tablename__ = "amazing-config"
    name = Column(String(255), primary_key=True)
    value = Column(String(255))

    def __init__(self, name, value):
        """
        self.name = name
        self.value = value
```

and a `default.py` file containing the parameters names, explanation and default value:

```python
from collections import OrderedDict
default = OrderedDict()
```

---

4.4. Extending MSNoise with Plugins
Then, the install.py file contains the method to add this table to the database:

```python
from msnoise.api import *
from .amazing_table_def import AmazingConfig
from .default import default

def main():
    engine = get_engine()
    Session = sessionmaker(bind=engine)
    session = Session()

    AmazingConfig.__table__.create(bind=engine, checkfirst=True)
    for name in default.keys():
        session.add(AmazingConfig(name=name, value=default[name][-1]))
    session.commit()
```

then add the command to the plugin_definition.py:

```python
@click.command()
def install():
    """ Create the Config table""
    from .install import main
    main()

amazing.add_command(install)
```

When all this is prepared, running the msnoise plugin amazing install command will connect to the current database, create the amazing-config table and add the parameters names and their default value.

An entry point to the setup.py file has to be defined in order to access Plugin’s config tables via the msnoise api msnoise.api.get_config() (page 42) method:

```python
'msnoise.plugins.table_def': [
    'AmazingConfig = msnoise_amazing.amazing_table_def:AmazingConfig',
],
```

Then, running a simple python command:

```python
from msnoise.api import connect, get_config

db = connect()
print(get_config(db, "parameter1", plugin="Amazing"))
print(get_config(db, "parameter2", plugin="Amazing"))
print(get_config(db, "parameter3", plugin="Amazing"))
print(get_config(db, "parameter4", plugin="Amazing"))
print(get_config(db, "question1", plugin="Amazing", isbool=True))
```
4.4.5 Adding Web Admin Pages

Plugins can also declare new pages to the Web Admin! This is simply done by, again, declaring some entry points in setup.py:

```python
'msnoise.plugins.admin_view': [ 'AmazingConfigView = msnoise_amazing.plugin_definition:AmazingConfigView', ],
```

and the corresponding object in plugin_definition.py:

```python
from flask.ext.admin.contrib.sqla import ModelView
from .amazing_table_def import AmazingConfig

class AmazingConfigView(ModelView):
    # Disable model creation
    view_title = "MSNoise Amazing Configuration"
    name = "Configuration"
    can_create = False
    can_delete = False
    page_size = 50
    # Override displayed fields
    column_list = ('name', 'value')

    def __init__(self, session, **kwargs):
        # You can pass name and other parameters if you want to
        super(AmazingConfigView, self).__init__(AmazingConfig, session,
                                             endpoint="amazingconfig",
                                             name="Config",
                                             category="Amazing", **kwargs)
```

Then (as always, after re-developing/installing the package), the magic occurs:
4.4.6 Uninstalling Plugins

Plugins can be de-activated by removing their package name from the plugins configuration parameter. Ideally, plugins should provide an uninstall command similar to the install to take care of deleting/dropping the tables in the project database.

4.4.7 Download Amazing Plugin

That’s cheating, you know? :-)

Download the Amazing Plugin
4.5 How To's

4.5.1 Reprocess data

When starting to use MSNoise, one will most probably need to re-run different parts of the Workflow more than one time. By default, MSNoise is designed to only process “what’s new”, which is antagonistic to what is wanted. Hereafter, we present cases that will cover most of the re-run techniques:

When adding a new filter

If new filter are added to the filters list in the Configurator, one has to reprocess all CC jobs, but not for filters already existing. The recipe is:

- Add a new filter, be sure to mark ‘used’=1
- Set all other filters ‘used’ value to 0
- Redefine the flag of the CC jobs, from ‘D’one to ‘T’odo with the following:
  - Run `msnoise reset CC --all`
  - Run `msnoise compute_cc`
- Run next commands if needed (stack, mwcs, dtt)
- Set back the other filters ‘used’ value to 1

The `compute_cc` will only compute the CC’s for the new filter(s) and output the results in the STACKS/ folder, in a subfolder named by a formatted integer from the filter ID. For example: STACKS/01 for ‘filter id’=1, STACKS/02 for ‘filter id’=2, etc.

When changing the REF

When changing the REF, the REF stack has to be re-computed:

`msnoise stack -r -i 999` will ensure all jobs marked done in the last 999 days are checked for modification. The REF will then be re-output.

When changing the MWCS parameters

If the MWCS parameters are changed in the database, all MWCS jobs need to be reprocessed.

`msnoise reset DTT --all`

`msnoise compute_mwcs`

should do the trick.

When changing the dt/t parameters

`msnoise compute_dtt -i 999` will ensure all MWCS jobs marked done in the last 999 days are checked for modification.
4.5.2 Define one’s own data structure of the waveform archive

The data_structure.py file contains the known data archive formats. If another data format needs to be defined, it will be done in the custom.py file in the current project folder:

See also:
Check the “Populate Station Table” step in the workflow.

4.5.3 How to have MSNoise work with 2+ data structures at the same time

Not yet implemented.

4.5.4 How to duplicate/dump the MSNoise configuration

Not yet implemented.

4.5.5 Testing the Dependencies

Once installed, you should be able to import the python packages in a python console. MSNoise comes with a little script called bugreport.py that can be useful to check if you have all the required packages (+ some extras).

The usage is such:

```
$ msnoise bugreport -h
usage: msnoise bugreport [-h] [-s] [-m] [-e] [-a]
Helps determining what didn't work

optional arguments:
  -h, --help           show this help message and exit
  -s, --sys            Outputs System info
  -m, --modules        Outputs Python Modules Presence/Version
  -e, --env            Outputs System Environment Variables
  -a, --all            Outputs all of the above
```

On my Windows machine, the execution of

```
$ msnoise bugreport -s -m
```

results in:

```
************* Computer Report *************
----------------+SYSTEM+-------------------
Windows seis31 7 6.1.7601 AMD64 Intel64 Family 6 Model 42 Stepping 7, GenuineIntel
```
The [X] marks the presence of the module. In the case above, PyQt4 is missing, but that’s not a problem because wx or PySide are present, so traitsui has a backend to render the GUI for the Configurator. The “not-required” packages are checked for information, those packages can be useful for reporting / hacking / rendering the data.

To install a missing package, for example obspy, use the pip command:

$$ \texttt{pip install obspy} $$

### 4.6 About Databases and Performances

To quote the SQLite website:

**Appropriate Uses For SQLite**

SQLite is different from most other SQL database engines in that its primary design goal is to be simple:
• Simple to administer
• Simple to operate
• Simple to embed in a larger program
• Simple to maintain and customize

Many people like SQLite because it is small and fast. But those qualities are just happy accidents. Users also find that SQLite is very reliable. Reliability is a consequence of simplicity. With less complication, there is less to go wrong. So, yes, SQLite is small, fast, and reliable, but first and foremost, SQLite strives to be simple.

Simplicity in a database engine can be either a strength or a weakness, depending on what you are trying to do. In order to achieve simplicity, SQLite has had to sacrifice other characteristics that some people find useful, such as high concurrency, fine-grained access control, a rich set of built-in functions, stored procedures, esoteric SQL language features, XML and/or Java extensions, tera- or peta-byte scalability, and so forth. If you need some of these features and do not mind the added complexity that they bring, then SQLite is probably not the database for you. SQLite is not intended to be an enterprise database engine. It is not designed to compete with Oracle or PostgreSQL.

The basic rule of thumb for when it is appropriate to use SQLite is this: Use SQLite in situations where simplicity of administration, implementation, and maintenance are more important than the countless complex features that enterprise database engines provide. As it turns out, situations where simplicity is the better choice are more common than many people realize.

Another way to look at SQLite is this: SQLite is not designed to replace Oracle. It is designed to replace fopen().

To test MSNoise, one can work with a SQLite database. SQLite communication is supported by default in Python (part of the standard library). The major drawback of SQLite is that it doesn’t support high concurrency. In the case of MSNoise, this means that only one Thread (or Process) can interact with the database “at a time”. For small batch tests or small runs, that is OK, but when processing larger archives (years of data of 5+ stations), then the implementation of a MySQL database will allow to process the jobs in parallel.

---

**Note:** I have been working on some sort of API server layer above a single SQLite database, working as a Queuing system. The API server is the only client of the database, and exchanges data with the code via json HTTP requests. Any help, idea, brainstorming on this is welcome!

---

### 4.7 References


4.8 Contributors

The following people have contributed to MSNoise (sorted alphabetically):

• Corentin Caudron
• Clare Donaldson
• Raphal De Plaen
• Robert Green
• Thomas Lecocq
• Aurlien Mordret
• Lukas E. Preiswerk
• Carmelo Sammarco

4.9 Help on the msnoise commands

This page shows all the command line interface commands

4.9.1 admin

msnoise admin --help
Usage: [OPTIONS]

Options:
-p, --port INTEGER Port to open
--help Show this message and exit.

4.9.2 bugreport

msnoise bugreport --help
Usage: [OPTIONS]

This command launches the Bug Report script.
Options:
- \texttt{-s, --sys} System Info
- \texttt{-m, --modules} Modules Info
- \texttt{-e, --env} Environment Info
- \texttt{-a, --all} All Info
- \texttt{--help} Show this message and exit.

### 4.9.3 compute\_cc

\texttt{msnoise compute\_cc --help}

Usage: [OPTIONS]

Computes the CC jobs (based on the "New Jobs" identified)

Options:
- \texttt{--help} Show this message and exit.

### 4.9.4 compute\_dtt

\texttt{msnoise compute\_dtt --help}

Usage: [OPTIONS]

Computes the dt/t jobs based on the new MWCS data

Options:
- \texttt{-i, --interval INTEGER} Number of days before now to search for modified Jobs
- \texttt{--help} Show this message and exit.

### 4.9.5 compute\_mwcs

\texttt{msnoise compute\_mwcs --help}

Usage: [OPTIONS]

Computes the MWCS based on the new stacked data

Options:
- \texttt{--help} Show this message and exit.

### 4.9.6 compute\_stretching

\texttt{msnoise compute\_stretching --help}

Usage: [OPTIONS]

[experimental] Computes the stretching based on the new stacked data
Options:
  --help  Show this message and exit.

4.9.7 config

msnoise config --help

Usage: [OPTIONS]

This command should now only be used to use the command line to set a parameter value in the data base. It used to launch the Configurator but the recommended way to configure MSNoise is to use the "msnoise admin" web interface.

Options:
  -s, --set TEXT  Modify config value: usage --set name=value
  -S, --sync  Sync station metadata from inventory/dataless
  --help  Show this message and exit.

4.9.8 info

msnoise info --help

Usage: [OPTIONS]

Outputs general information about the current install and config, plus information about jobs and their status.

Options:
  -j, --jobs  Jobs Info only
  --help  Show this message and exit.

4.9.9 install

msnoise install --help

Usage: [OPTIONS]

This command launches the installer.

Options:
  --help  Show this message and exit.

4.9.10 ipython

msnoise ipython --help

Usage: [OPTIONS]
Launches an ipython notebook in the current folder

Options:
--help  Show this message and exit.

4.9.11 new_jobs

msnoise new_jobs --help

Usage: [OPTIONS]

Determines if new CC jobs are to be defined

Options:
-i, --init First run ?  
--nocc Disable the creation of CC jobs  
--help Show this message and exit.

4.9.12 p

Will be automatically populated with the commands declared by the plugins (p is an alias for plugin)

4.9.13 plot

ccftime

msnoise plot ccftime --help

Usage: [OPTIONS] STA1 STA2

Plots the ccf vs time between sta1 and sta2 (parses the dt/t results)

STA1 and STA2 must be provided with this format: NET.STA

Options:
-f, --filterid INTEGER Filter ID  
-c, --comp TEXT Components (ZZ, ZR,...)  
-m, --mov_stack INTEGER Mov Stack to read from disk  
-a, --ampli FLOAT Amplification  
-S, --seismic Seismic style  
-s, --show BOOLEAN Show interactively?  
-o, --outfile TEXT Output filename (=auto)  
-e, --envelope Plot envelope instead of time series  
-r, --refilter TEXT Refilter CCFs before plotting (e.g. 4:8 for filtering CCFs between 4.0 and 8.0 Hz. This will update the plot title.

--help Show this message and exit.
4.9. Help on the msnoise commands

**data availability**

```
msnoise plot data_availability --help
```

Usage: [OPTIONS]

Plots the Data Availability vs time

Options:

- `-s, --show BOOLEAN` Show interactively?
- `-o, --outfile TEXT` Output filename (?,=auto)
- `--help` Show this message and exit.

**distance**

```
msnoise plot distance --help
```

Usage: [OPTIONS]

Plots the REFs of all pairs vs distance

Options:

- `-f, --filterid INTEGER` Filter ID
- `-c, --comp TEXT` Components (ZZ, ZR,...)
- `-a, --ampli FLOAT` Amplification
- `-s, --show BOOLEAN` Show interactively?
- `-o, --outfile TEXT` Output filename (?,=auto)
- `-r, --refilter TEXT` Refilter CCFs before plotting (e.g. 4:8 for filtering CCFs between 4.0 and 8.0 Hz. This will update the plot title.
- `--virtual-source TEXT` Use only pairs including this station. Format must be NET.STA
- `--help` Show this message and exit.

**dtt**

```
msnoise plot dtt --help
```

Usage: [OPTIONS] STA1 STA2 DAY

Plots a graph of dt against t

STA1 and STA2 must be provided with this format: NET.STA !

DAY must be provided in the ISO format: YYYY-MM-DD

Options:

- `-f, --filterid INTEGER` Filter ID
- `-c, --comp TEXT` Components (ZZ, ZR,...)
- `-m, --mov_stack INTEGER` Mov Stack to read from disk
- `-s, --show BOOLEAN` Show interactively?
- `-o, --outfile TEXT` Output filename (?,=auto)
- `--help` Show this message and exit.
**dvv**

```
msnoise plot dvv --help
```

**Usage:** [OPTIONS]

Plots the dv/v (parses the dt/t results)

Individual pairs can be plotted extra using the -p flag one or more times.

Example: msnoise plot dvv -p ID_KWUI_ID_POSI

Example: msnoise plot dvv -p ID_KWUI_ID_POSI -p ID_KWUI_ID_TRWI

Remember to order stations alphabetically!

**Options:**

- `-f, --filterid INTEGER` Filter ID
- `-c, --comp TEXT` Components (ZZ, ZR,...)
- `-m, --mov_stack INTEGER` Plot specific mov stacks
- `-p, --pair TEXT` Plot a specific pair
- `-A, --all` Show the ALL line?
- `-M, --dttname TEXT` Plot M or M0?
- `-s, --show BOOLEAN` Show interactively?
- `-o, --outfile TEXT` Output filename (?=auto)
- `--help` Show this message and exit.

---

**interferogram**

```
msnoise plot interferogram --help
```

**Usage:** [OPTIONS] STA1 STA2

Plots the interferogram between sta1 and sta2 (parses the CCFs)

STA1 and STA2 must be provided with this format: NET.STA !

**Options:**

- `-f, --filterid INTEGER` Filter ID
- `-c, --comp TEXT` Components (ZZ, ZR,...)
- `-m, --mov_stack INTEGER` Mov Stack to read from disk
- `-s, --show BOOLEAN` Show interactively?
- `-o, --outfile TEXT` Output filename (?=auto)
- `-r, --refilter TEXT` Refilter CCFs before plotting (e.g. 4:8 for filtering CCFs between 4.0 and 8.0 Hz. This will update the plot title.
- `--help` Show this message and exit.

---

**mwcs**

```
msnoise plot mwcs --help
```

**Usage:** [OPTIONS] STA1 STA2
Plots the mwcs results between sta1 and sta2 (parses the CCFs)

STA1 and STA2 must be provided with this format: NET.STA!

Options:
- `--filterid INTEGER` Filter ID
- `--comp TEXT` Components (ZZ, ZR,...)
- `--mov_stack INTEGER` Mov Stack to read from disk
- `--show BOOLEAN` Show interactively?
- `--outfile TEXT` Output filename (?=auto)
- `-h` Show this message and exit.

```
msnoise plot station_map --help
```

Usage: `[OPTIONS]`

Plots the station map (very very basic)

Options:
- `--show BOOLEAN` Show interactively?
- `--outfile TEXT` Output filename (?=auto)
- `-h` Show this message and exit.

```
msnoise plot timing --help
```

Usage: `[OPTIONS]`

Plots the timing (parses the dt/t results)

Individual pairs can be plotted extra using the `-p` flag one or more times.

Example: `msnoise plot timing -p ID_KWUI_ID_POSI`

Example: `msnoise plot timing -p ID_KWUI_ID_POSI -p ID_KWUI_ID_TRWI`

Remember to order stations alphabetically!

Options:
- `--filterid INTEGER` Filter ID
- `--comp TEXT` Components (ZZ, ZR,...)
- `--mov_stack INTEGER` Plot specific mov stacks
- `--pair TEXT` Plot a specific pair
- `-A` Show the ALL line?
- `-M` Plot M or MO?
- `--show BOOLEAN` Show interactively?
- `--outfile TEXT` Output filename (?=auto)
- `-h` Show this message and exit.
4.9.14 plugin

Will be automatically populated with the commands declared by the plugins (p is an alias for plugin)

4.9.15 populate

msnoise populate --help

Usage: [OPTIONS]

Rapidly scan the archive filenames and find Network/Stations

Options:
--fromDA Populates the station table using network and station codes found in the data_availability table, overrides the default workflow step.
--help Show this message and exit.

4.9.16 reset

msnoise reset --help

Usage: [OPTIONS] JOBTYPE

Resets the job to "T"odo. ARG is [CC] or [DTT]. By default only resets jobs "I"n progress. --all resets all jobs, whatever the flag value

Options:
-a, --all Reset all jobs
-r, --rule TEXT Reset job that match this SQL rule
--help Show this message and exit.

4.9.17 scan_archive

msnoise scan_archive --help

Usage: [OPTIONS]

Scan the archive and insert into the Data Availability table.

Options:
-i, --init First run ?
--path TEXT Scan all files in specific folder, overrides the default workflow step.
--help Show this message and exit.
4.9.18 stack

`msnoise stack --help`

Usage: [OPTIONS]

Stacks the [REF] and/or [MOV] windows

Options:
- `-r`, `--ref` Compute the REF Stack
- `-m`, `--mov` Compute the MOV Stacks
- `-s`, `--step` Compute the STEP Stacks
- `-i`, `--interval INTEGER` Number of days before now to search for modified Jobs
- `--help` Show this message and exit.

4.9.19 test

`msnoise test --help`

Usage: [OPTIONS]

Runs the test suite, should be executed in an empty folder!

Options:
- `--help` Show this message and exit.

4.9.20 upgrade_db

`msnoise upgrade_db --help`

Usage: [OPTIONS]

Upgrade the database from previous to a new version.

This procedure adds new parameters with their default value in the config database.

Options:
- `--help` Show this message and exit.
5.1 MSNoise 1.5

Release date: 28 April 2017
Release type: major
Release notes:

- Introduction (page 80)
- Requirements (page 81)
- Web-based Admin Interface Changes (page 81)
  - Configuration Parameters (page 81)
- Populate Station Table and Scan Archive (page 82)
  - Expert/Lazy mode (page 82)
- New Jobs (page 83)
- Preprocessing and Cross-Correlation (page 83)
  - Preprocessing (page 83)
  - Cross-Correlation (page 83)
- Command Line changes (page 84)
- API Changes (page 84)
- Plugin support (page 84)
- Other Bugfixes (page 84)
- Plot Updates (page 84)
- Performance and Code improvements (page 85)
- Documentation (page 85)
- Upgrading an existing project to MSNoise 1.5 (page 85)
- A final note about development pace and choices (page 86)
5.1.1 Introduction

About 1 year after the last major release (MSNoise 1.4 (page 86)) we are proud to announce the new MSNoise 1.5 (page 79). It is a major release, with a massive amount of work since the last one: in GitHub numbers, it’s over 120 commits and over 2500 lines of code and documentation changed or added!

MSNoise 1.5 introduces a series of new features:

- We have started to move core math functions to ObsPy, currently the only one ready is linear_regression, a function I wrote to remove the dependency to statsmodels, required to move mwcs to ObsPy later.
- The preprocessing routine has been isolated, rewritten and optimized. It is now a standalone script, callable by plugins. It returns a Stream object with all the data needed for the analysis.
- This change in preprocessing was done mostly to allow cross-component, auto-correlation and cross-correlation, with or without rotation, to be done with the same code. CC, SC and AC are now supported in MSNoise with proper whitening (possible to disable spectral whitening for specific cases).
- This documentation is now available in PDF too (easier for offline usage) and it also includes a new tutorial for setting up the MySQL server and Workbench.
- Last but not least: MSNoise is “tested” automatically on Linux (thanks to TravisCI) & Windows (thanks to Appveyor), for Python versions 2.7 and 3.5. With MSNoise 1.5 we also added the MacOSX tests on TravisCI. With these tests, we can guarantee MSNoise works on different platforms and Anaconda (or miniconda) python versions.

This version has benefited from outputs/ideas/pull requests/questions from several users/friends (listed alphabetically):

- Raphael De Plaen
- Clare Donaldson
- Robert Green
- Aurelien Mordret
- Lukas Preiswerk
- The participants to the NERC MSNoise Liverpool Workshop in January 2017
- all others (don’t be mad :-)

Thanks to all for using MSNoise, and please, let us know why/how you use it (and please cite it)!

To date, we found/are aware of 25 publications using MSNoise! That’s the best validation of our project ever! See the full list on the MSNoise website.

*Thomas, Corentin and others*

---

PS: if you use MSNoise for your research and prepare publications, please consider citing it:

5.1.2 Requirements

If you have any package older than those mentioned here, some things might not work as expected.

- Pandas >= 0.18.0
- obspy >= 1.1

No longer needed:

- scikits-samplerate
- statsmodels

5.1.3 Web-based Admin Interface Changes

- Bugfix: Stations can now be added manually
- Bugfix: Bulk operations on Job based on their jobtype is now possible
- Bugfix: Set debug-mode to False to increase performance and remove the risk of Werkzeug failing.
- Feature: Calling admin/data_availability.png should serve the image directly
- Feature: It’s now possible to “Brand” MSNoise: The name and the logo of the page can be overridden by setting an environment variable with a name and the HTML tag of the logo image:

```bash
set msnoise_brand="ROB|<img src='http://www.seismologie.be/img/oma/ROB-logo.svg'|width=200 height=200>"
```

and then starting msnoise admin:

![ROB Dashboard](image.png)

**ROB Dashboard**

Project Folder: C:\\tmp
Project Database: SQLite: msnoise.sqlite

Configuration Parameters

- REMOVED: decimation_factor: it’s now computed automatically from the stream’s sampling rate. If the decimation factor is non-integer, the routine exits and the user is advised to use the Lanczos resampler.
- REMOVED: ZZ, RR, TT, etc in favor of components_to_compute.
● ADDED: `components_to_compute`: it’s accepting a comma-separated list of components.
● ADDED: whitening method for disabling spectral whitening in specific cases.

5.1.4 Populate Station Table and Scan Archive

● Because of platform-dependent issues, we dropped the need for the “find” command to be present on each machine. The “cron” `scan_archive` procedure is now a simple python routine. It might take a little longer, but much safer. This also means the `crondays` parameter can be set to a floating point value, “-1.0” meaning “files modified in the last 1 day”.
● The custom archive structure can now directly be input in the config. This is still quite preliminary and needs at least one slash in the parameter value to be identified as such by the code.
● A change in the database structure (added an index on the “path”+”file”+”net”+”sta”+”comp”) allows running `msnoise scan_archive` a little more efficiently than before.

**Warning:** The change in the database (adding an index) requires that you `msnoise upgrade_db` every project!

Expert/Lazy mode

● Added an Expert/Lazy mode to scan files directly by passing the path of their containing folder to `scan_archive --path` *(see here)* (page 19).
● Added an Expert/Lazy mode to input network/stations directly from the data scanned using the new `populate --fromDA` procedure *(see here)* (page 19).
● The `scan_archive` is now capable of handling multiplexed files and inputs one line in `DataAvailability` per unique “trace id” in files.

Practically, the three changes above allow users to:

1. Download a single or few mseed file with N stations, for example from webdc or using FDSN webservice in ObsPy or else.
2. Run `msnoise install`
3. Run `msnoise scan_archive --path /path/to/the/big/file/`
4. Run `msnoise populate --fromDA`
5. Run `msnoise new_jobs --init`
6. Run `msnoise admin`, define (pre-)processing and filter parameters
7. Run `msnoise compute_cc`
5.1.5 New Jobs

- The `msnoise new_jobs` routine has been rewritten to correct a few bugs linked to start/end dates (missed days if a file of day 2 begins on day 1 and ends on day 3, jobs were only created for day 1 and 3. This is corrected.
- A change in the database structure (added an index on the “day”+”pair”+”jobtype”) allows running `msnoise new_jobs` much faster than before. It took 5m40s for a test with 105381 jobs, while it would have taken hours before. For the first run, it is still faster (20s for the example above) to use `msnoise new_jobs --init` as this doesn’t check for existing jobs.

Warning: The change in the database (adding an index) requires that you `msnoise upgrade_db` every project!

5.1.6 Preprocessing and Cross-Correlation

Preprocessing

- A complete rewrite of the preprocessing function to avoid padding and merging with zeros. The preprocessing function is now separated from the `compute_cc` code and can be called by external plugins. It returns a Stream object that can easily be filtered or slid.
- The instrument response removal has been accelerated by doing it after decimation, on fewer data points (Thanks to Robert Green). The response removal is done without the `evalresp` stuff from ObsPy, it’s faster but potentially a little less safe.
- The default decimation tool is now Lanczos (builtin in ObsPy) and scikits.samplerate is no longer needed.

Cross-Correlation

- It is now possible to do the Cross-Correlation (classic “CC”), the Auto- Correlation (“AC”) or the Cross-Components within the same station (“SC”). To achieve this, we removed the ZZ, ZT, etc parameters from the configuration and replaced it with `components_to_compute` which takes a list: e.g. `ZZ,ZE,ZN,EZ,EE,EN,NZ,NE,NN` for the full non-rotated tensor between stations. If `autocorr` is set to “Y”, then the cross-components (SC) of each station will also be computed (of course, ZE and EZ are identical).
- The cross-correlation is done on sliding windows on the available data. If one trace contains a gap, the window is skipped. This corrects previous errors linked with gaps synchronised in time that lead to perfect sinc autocorr functions. The windows should have a duration of at least 2 times the `maxlag` configuration value.
- The whitening procedure can be skipped by setting the `whitening` configuration to `None`. The two other `whitening` modes are “[A]ll except for auto-correlation” or “Only if [C]omponents are different”. This allows skipping the whitening when, for example, computing ZZ components for very close by stations (much closer than the wavelength sampled), leading to spatial autocorrelation issues.
5.1.7 Command Line changes

- **msnoise config --sync** will try to parse the dataless files for existing stations in the Station table and, if found, will input the coordinates.

- **msnoise scan_archive --path** will only scan the path independently of its structure. It will only read files, not “walk” in subfolders.

- **msnoise populate --fromDA** will populate the station table from the existing data_availability table.

- **msnoise p** is a lazy alias to **msnoise plugin**

- Add a default delay of 1 second (customizable) to start parallel threads (using -t)

Note, all commands are documented: *Help on the msnoise commands* (page 69).

5.1.8 API Changes

- New **make_same_length** API method to return common data for two traces, this is necessary to compute the rotation of the horizontal traces to Radial/Transverse.

- New **clean_scipy_cache** API method to reduce the memory imprint caused by scipy’s automatic caching of FFT arrays.

See *MSNoise API* (page 41).

5.1.9 Plugin support

- Plugins can now declare their own templates using the **msnoise.plugins.templates** entry point.

- Plugins can override the “components_to_compute” config bit and the MSNoise API **get_components_to_compute(session, plugin=None)** works like **get_config**.

See *Extending MSNoise with Plugins* (page 57).

5.1.10 Other Bugfixes

- Removed the call to **scipy.stats.nanmean** and replaced by **numpy.nanmean**

- Better error message in compute_cc when the content of the slice is only zeros or smaller than **rms_threshold**

- Checked all “integer” - “float” warnings from numpy/scipy

- Crondays were hardcoded to -2, now taking the **crondays** value from the DB

- Py3 error when **msnoise scan_archive** in cron mode

5.1.11 Plot Updates

- **msnoise plot ccftime** now accepts -e (--envelope) and will plot the envelope of the ccfs.
• `msnoise plot ccftime`, `msnoise plot interferogram` and `msnoise plot distance` now accept `-r` (`--refilter`) to refilter the CCFs before plotting. The argument must be a column-separated string (e.g. `-r 4:8` for filtering between 4.0 and 8.0 Hz).

• `msnoise plot distance` accepts a new `--virtual-source NET.STA` parameter to only plot the pairs including this station.

• Most plots have better titles (filter details, etc).

• The dv/v plot now allows averaging over components by passing them as comma-separated values.

See *Plotting* (page 31).

### 5.1.12 Performance and Code improvements

Improvements in terms of performances have also been done for MSNoise 1.5:

• Added fftpack optimized `nfft` (scipy’s `next_fast_len`). This could lead to some small differences in the final result of the MWCS procedure, because of the number of points used for smoothing the (cross-)spectra.

• Replaced binarization (sign) and windsorizing (clip) by standard numpy functions operating directly inplace on the arrays, avoiding unnecessary copies.

• The preprocessing only reads files that should contain the right component.

• The stretching code has been improved (thanks to Clare Donaldson)

### 5.1.13 Documentation

• New tutorial for installing and configuring MySQL and MySQL workbench.

• The Workflow is separated in steps (See *Workflow* (page 11))

• The Documentation is now available in PDF format (PDF).

### 5.1.14 Upgrading an existing project to MSNoise 1.5

Some users will want to keep their current project without recomputing everything. This requires adding a few configuration parameters to the database.

Running the following command will take care of the upgrade from 1.4 to 1.5:

```
msnoise upgrade_db
```

**Warning:** Upgrading the database will not remove deprecated configuration bits, so users should remember to define, for example, the `components_to_compute` parameter if anything else than ZZ was set before.
5.1.15 A final note about development pace and choices

- MSNoise team is
  - **1 developper** (Thomas)
  - 1 dedicated debugger (Corentin)
  - less than 10 really **active** users, providing feedback and/or lines of codes (Esteban, Raphal, Aurlien, Carmelo, Clare, Rob ...)

- All software engineering ideas are coming from too infrequent beerstormings between Thomas & others

- The web-interface and the plugin support were developed during Thomas’ holidays

If you need help, please ask your questions on the mailing list. Don’t be afraid to ask. If you have ideas, please share them. If you develop codes to supplement MSNoise, please share them, even if very small, even if you don’t master gitHub. If you have complaints, post them too, but remember that the package you are using has been coded by 1 person, and that it’s not his full time job. So MSNoise is provided “as-is”, carefully written and tested, but there will be bugs, issues, incompatibility with certain python installations, OS or module versions. If you **want or need** developments made, contact Thomas via email directly. If these developments are within the focus of the developers’ research, then a collaboration, i.e. resulting in a co-authored peer reviewed publication, can be an option. Otherwise, you can contract us for paid-developments.

5.2 MSNoise 1.4

Release date: 11 April 2016
Release type: major

Release notes:

- **Introduction** (page 87)
- **Web-based Admin Interface** (page 87)
- **Plugin support** (page 88)
- **Phase Weighted Stack** (page 89)
- **Instrument Response Correction** (page 90)
- **Command Line changes** (page 90)
- **Customizing Plots** (page 91)
- **New plots** (page 91)
- **Performance improvements** (page 91)
- **Upgrading an existing project to MSNoise 1.4** (page 92)
- **A final note about development pace and choices** (page 92)
5.2.1 Introduction

Just over a year after the last major release (MSNoise 1.3 (page 93)) we are proud to announce the new MSNoise 1.4 (page 86). It is a major release, with a massive amount of work since the last one: in GitHub numbers, it’s over 125 commits and about 5500 new lines of code and documentation added!

MSNoise 1.4 introduces four major new features: a new ultra-intuitive web-based admin interface, the support for plugins and extensions, the phase weighted stack and the instrument response removal. It also brings the possibility to parallel/thread process the cross-correlation and the MWCS steps. MSNoise is now “tested” automatically on Linux (thanks to TravisCI) & Windows (thanks to Appveyor), for Python versions 2.7, 3.4 and 3.5. Yes, MSNoise is Python 3 compatible!!!

This version has benefited from outputs/ideas/pull requests/questions from several users/friends:

- Carmelo Sammarco
- Esteban Chaves
- Lion Krischer
- Tobias Megies
- Clare Donaldson
- Aurlien Mordret
- Raphal De Plaen
- Lukas E. Preiswerk
- all others (don’t be mad :-)

Thanks to all for using MSNoise, and please, let us know why/how you use it (and please cite it)!

To date, we found/are aware of 12 publications using MSNoise! That’s the best validation of our project ever! See the full list on the MSNoise website.

Thomas & Corentin

PS: if you use MSNoise for your research and prepare publications, please consider citing it:


5.2.2 Web-based Admin Interface

For this release, we have replaced the Configurator by a more intuitive web-based configuration interface. All fields present in the Configurator are present, and more!

Example view:
Pros:

- easier to customise
- modern
- less bandwidth when working remotely
- removes dependency for traits/traitui
- allows to customise “Views” to provide more information
- allows the validation of fields before saving to database
- “home” page shows the status of the Data Availability and of the Jobs
- will allow interactive plotting in the future

Cons:

- adds dependency to flask & flask-admin

See MSNoise Admin (Web Interface) (page 12) for more details!

5.2.3 Plugin support

MSNoise supports Plugins, this means the default workflow “from archive to dv/v” can be branched at any step (well, currently at 3 steps)!

A plugin is a python package, properly structured, that can be imported from msnoise, i.e. it has to be “installed” like any other python package.

After installing a plugin, its package name must be declared in the plugins parameter in the configuration. This must be done PER PROJECT.

Pythonly speaking, plugins declare entry points that MSNoise uses to define commands and job types. This way, the msnoise plugin command would populate with the plugin commands and plugin’s custom Job Types will be declared at different steps of the code. Currently, custom job types can be defined at three moments of the workflow: “when a new file is detected in the archive”, “when a new pair-job has to be done” and “when a new REF stack has been computed”. More will be added in the future releases.

See Extending MSNoise with Plugins (page 57) to learn how to create this Amazing plugin:

```
$ msnoise plugin amazing

Usage: msnoise-script.py plugin amazing [OPTIONS] COMMAND [ARGS]...

Example Amazing Plugin for MSNoise
```
5.2.4 Phase Weighted Stack

**Warning:** Largely untested, not cross-validated. It looks good, but that doesn’t mean a lot, does it? Use with Caution! And if you cross-validate it, please let us know!!

The Phase Weighted Stack has been implemented. When configured, it is applied on both the “N minutes windows to daily stacks”, on the “moving window stacks” and on the “reference stacks”. It is thus applied at the compute cc and at the stack steps.

If `stack_method` is ‘linear’, then a simple mean CFF is done. On the other hand, if `stack_method` is ‘pws’, then all the Phase Weighted Stack (PWS) is computed and saved as the mov or ref CCF. The PWS is done in two steps: first the mean coherence between the instataneous phases of all windows is calculated, and eventually serves a weighting factor on the mean. The smoothness of this weighting array is defined using the `pws_timegate` parameter in the configuration. The weighting array is the power of the mean coherence array. If `pws_power` is equal to 0, a linear stack is done (then it’s faster to do set `stack_method = 'linear'`). Usual value is 2.

**See also:**


The following example has been provided by Carmelo Sammarco (thanks for testing/debugging!, but COME ONE, why SAC ??!!). There is maybe a too narrow time-gate on the final stack, but overall, the amplitude of the non-coherent phases are greatly diminished. Each of the three rows show the “REF” stack and its FTAN image. The three rows are:

- A: Linear stack of “windows” to “daily” - Linear stack of “daily” to “ref”
- B: PWS stack of “windows” to “daily” - Linear stack of “daily” to “ref”
- C: PWS stack of “windows” to “daily” - PWS stack of “daily” to “ref”

To obtain an LIN-LIN or PWS-PWS stack, simply set `stack_method = 'linear'` or `pws`, respectively when running the `compute_cc` and `stack` steps. And for mixed cases LIN-PWS or PWS-LIN, edit the config between the two steps!
5.2.5 Instrument Response Correction

During pre-processing of the waveforms (at `compute_cc` step), the instrument response can be corrected, based on those parameters:

- **remove_response**: Remove instrument response Y/[N] (default=N)
- **response_format**: Remove instrument file format [dataless]/inventory/paz/resp (default=dataless)
- **response_path**: Instrument correction file(s) location (path relative to db.ini), defaults to './inventory', i.e. a subfolder in the current project folder. All files in that folder will be parsed. (default=inventory)
- **response_prefilt**: Remove instrument correction pre-filter (0.005, 0.006, 30.0, 35.0) (default=(0.005, 0.006, 30.0, 35.0))

Currently, only dataless seed and inventory XML are supported.

Note: Removing the instrument response is a computationally very expensive task and not useful for dv/v iff your instruments didn’t change during the analysed period. It is also not needed for tomography iff all instruments are the same, or at least have an identical phase response in the frequency band of interest.

5.2.6 Command Line changes

- **msnoise admin**: new command to start the web interface
• **msnoise config**: accepts a \texttt{--set name=value} option, to rapidly change a configuration parameter in the database.

• The **msnoise** command accepts a \texttt{-c} option that triggers the “custom” mode, currently only for plots. See below.

• **msnoise compute_cc** and **msnoise compute_mwcs**: support the \texttt{-t} threading flag and should be able to work multiple threads. Example calls: \texttt{msnoise -t 4 compute_cc} or \texttt{msnoise -t 16 compute_mwcs}. Don’t start more threads than the actual number of real cores on your machine, and take into account that if each instance loads a lot of data (stations), you should have enough RAM to store it.

• **msnoise info** now returns the location where the MSNoise package is installed (useful when developing / hacking the code). It also returns the list of configured Filters and Stations in the database.

All commands are now documented: *Help on the msnoise commands* (page 69).

5.2.7 Customizing Plots

All plots commands can be overridden using a \texttt{-c} argument \textit{in front of the plot command} !!

Examples:

- \texttt{msnoise -c plot distance}
- \texttt{msnoise -c plot ccftime YA.UV02 YA.UV06 -m 5}
- etc.

To make this work, one has to copy the plot script from the msnoise install directory to the project directory (where your db.ini file is located), then edit it to one’s desires. The first thing to edit in the code is the import of the *MSNoise API* (page 41):

from ..api import *

to

from msnoise.api import *

and it should work.

5.2.8 New plots

- plot dtt

5.2.9 Performance improvements

Improvements in terms of performances have also been done for MSNoise 1.4:

• **keep_all**: if set to \texttt{Y (=True)} in the config, all CCF are now stored in a single HDF5 file, which makes it much nicer to backup/transfer/delete. – XXX not used actually !!

• **compute_cc**: reversed the change done in 1.3, the pre-whitening of the traces is now disabled, it led to very high memory usage and needs a fresh rewrite. This doesn’t mean whitening is no longer done, but just some sort of caching of the pre-whitened traces.
5.2.10 Upgrading an existing project to MSNoise 1.4

Some users will want to keep their current project without recomputing everything. This requires adding a few configuration parameters to the database.

Running the following command will take care of the upgrade from 1.3 to 1.4:

```
msnoise upgrade_db
```

There was a bug, mainly present in MySQL, with too sharp rounding of station coordinates. The bugfix change is done automatically for MySQL databases. It is a little different if you are using SQLite as it can’t be done automatically. This is because SQLite doesn’t support “ALTER” commands. Ultimately we want the `station.X` and `station.Y` to be of type `double`. You will have to do this operation manually:

```
Warning: Do the following at your own risk. It might not be needed! From the tests we ran, it seems the coordinates rounding error was present only for MySQL databases!
```

- Open SQLite database browser (SQLiteManager extension for Firefox, for example)
- Open the msnoise.sqlite file
- Select the station table
- Edit the X field and change its type to `double`
- Edit the Y field and change its type to `double`
- Ignore the warnings (it should work, although it could fail!)
- Close the database

5.2.11 A final note about development pace and choices

- MSNoise team is
  - 1 developper (Thomas)
  - 1 dedicated debugger (Corentin)
  - less than 5 really active users, providing feedback and/or lines of codes (Esteban, Raphael, Aurlien, Carmelo, ...)
- All software engineering ideas are coming from too infrequent beerstormings between Thomas & others
- The web-interface and the plugin support were developed during Thomas’ holidays

If you need help, please ask your question on the mailing list. Don’t be afraid to ask. If you have ideas, please share them. If you develop codes to supplement MSNoise, please share them, even if very small, even if you don’t master gitHub. If you have complaints, post them too, but remember that the package you are using has been coded by 1 person, and that it’s not his full time job. So MSNoise is provided “as-is”, carefully written and tested, but there will be bugs, issues, incompatibility with certain python installations, OS or module versions. If you want or need developments made, contact Thomas via email directly. If these developments are within the focus of the developers’ research, then a collaboration, i.e. resulting in a co-authored peer reviewed publication, can be an option. Otherwise, you can contract us for paid-developments.
5.3 MSNoise 1.3.1

Release date: 26 March 2015
Release type: bugfix
Release notes:

5.3.1 BugFixes

When running the new jobs procedure, the jobs are “inserted” in the database, even if they already existed (they should be “updated”). This is because of the complete rewrite of the code to optimize the operation.

As this optimization is mostly useful upon first run, I’ve added a parameter –init to the command. If provided, the “massive insert” procedure is used, if not, then the classic “insert or update if existing” is used.

So, upon first run: `msnoise new_jobs --init` And afterwards (in cron, e.g.): `msnoise new_jobs`

Users who have already run MSNoise 1.3 on their archive need to clean the jobs table in the database. The buggy jobs are those “CC” jobs which are still marked “I”n progress after the compute_cc procedure, and with a “lastmod” = “NULL”. They can be easily identified and removed.

A classic SQL command would be:

`DELETE from jobs WHERE lastmod is NULL;`

5.4 MSNoise 1.3

Release date: 20 March 2015
Release type: major
Release notes:

- Introduction (page 94)
- MSNoise is a real Python Package (page 95)
- msnoise is now a top-level command (page 95)
- msnoise plot: Plotting made easy (page 96)
- New functionality: Dynamic time lags (page 98)
- Math updates & bugfixes (page 99)
- Performance improvements (page 99)
- MSNoise has a proper API: Hacking MSNoise (page 100)
- MSNoise is tested (page 100)
5.4.1 Introduction

8 months after the last bugfix release (MSNoise 1.2.5 (page 101)), and 17 months after the last major release (MSNoise 1.2 (page 103)) we are proud to announce the new MSNoise 1.3 (page 93). It is a major release, with a massive amount of work since the last release: in GitHub numbers, it’s over 100 commits and about 3500 new lines of code and documentation added! MSNoise 1.3 introduces a brand new way of executing the workflow. The workflow in itself doesn’t change, so experienced users as well as new users reading the SRL publication will find their way easily!

MSNoise is now a Python Package, allowing a single (and easy) install for all your projects and/or all users using pip. The new top-level msnoise command contains all the steps of the workflow, plus new additions, as the very useful reset command to easily mark all jobs “T”odo. The msnoise plot command group which includes seven plots, all directly callable from the command line, without needing to hack/edit the source codes. About hacking: MSNoise has now a proper documented API which allows pythonistas to write their own plots, computation steps, ..., while interacting with the database and the data archive! The “dynamic time lag” allows to use parts of the coda that is dependent from the interstation distance (provided station coordinates are defined). Finally, MSNoise is now tested and automatically checked by Travis-CI!

This version has benefited from outputs/ideas/pull requests/questions from several users:

- Rebecca Kramer
- Carmelo Sammarco
- Oscar Alberto Castro Artola
- Kasper van Wijk
- Kohtaro R. Araragi
- Esteban Chaves
- Adrian Shelley
- Weston Thelen
- Robert Abbott
- Jean Battaglia
- Sbastien Carniato
- Xiao Wang
- Lion Krisher
- Tobias Megies
- all participants to the 2014 Pre-AGU MSNoise workshop
- all others (don’t be mad :-) )
Thanks to all for using MSNoise, and please, let us know why/how you use it (and please cite it)!

Thomas Lecocq & Corentin Caudron

PS: if you use MSNoise for your research and prepare publications, please consider citing it:


5.4.2 MSNoise is a real Python Package

This is probably the most important change since the original release of MSNoise 1.0 (August 2013), it represents a massive amount of work since the last release (1.2.5 in June 2014), and is probably the most needed by users! In GitHub numbers, it’s over 100 commits and about 3500 new lines of code (and of documentation!) added!

In practice, what does change?

- MSNoise is installable using pip or easy_install, soon using conda
- MSNoise is installed in the common “site-package/” folder of one’s python install.
- Once installed, it is available for all users, all projects.
- It allows updating MSNoise for all projects at once.
- It removes all python files from project folders, which is much cleaner.

MSNoise being installed in the standard lib directories means it shouldn’t write or output anything in those folders. To facilitate the launch of commands a new top level msnoise command has been created, and should be available right after installing.

5.4.3 msnoise is now a top-level command

Users of MSNoise will have to change the way they call the steps, i.e.:

- python s000_installer.py becomes msnoise install
- python s001_configurator.py becomes msnoise config
- python s002_populate_station_table.py becomes msnoise populate
- python s01_scan_archive.py becomes msnoise scan_archive
- python s02_new_jobs.py becomes msnoise new_jobs
- python s03_compute_cc.py becomes msnoise compute_cc
- python s04_stack.py becomes msnoise stack
- python s05_compute_mwcs.py becomes msnoise compute_mwcs
- python s06_compute_dtt.py becomes msnoise compute_dtt

All the commands are visible using the --help argument:

5.4. MSNoise 1.3
msnoise --help

Usage: msnoise-script.py [OPTIONS] COMMAND [ARGS]...

Options:
-t, --threads INTEGER  Number of threads to use (only affects modules that are designed to do parallel processing)
--help                  Show this message and exit.

Commands:
bugreport            This command launches the Bug Report script.
compute_cc           Computes the CC jobs (based on the "New Jobs"
compute_dtt          Computes the dt/t jobs based on the new MWCS...
compute_mwcs         Computes the MWCS based on the new stacked...
compute_stretching   [experimental] Computes the stretching based...
config               This command launches the Configurator.
info                 Outputs general information about the current...
install              This command launches the installer.
ipython              Launches an ipython notebook in the current...
new_jobs             Determines if new CC jobs are to be defined
plot                 Top level command to trigger different plots
populate             Rapidly scan the archive filenames and find...
reset                Resets the job to "T"odo.
scan_archive         Scan the archive and insert into the Data...
stack                Stacks the [REF] and/or [MOV] windows
test                 Runs the test suite, should be executed in an...
upgrade_db           Upgrade the database from pre-1.3 to MSNoise...

The parameters/arguments of each command are explained using its own --help, for example:

msnoise reset --help

Usage: msnoise-script.py reset [OPTIONS] JOBTYPE

Resets the job to "T"odo. ARG is [CC] or [DTT]. By default only resets jobs "I"n progress. --all resets all jobs, whatever the flag value

Options:
-a, --all Reset all jobs
--help Show this message and exit.

The description of each step has been updated in the documentation.

### 5.4.4 msnoise plot: Plotting made easy

As explained above, msnoise is a top level command available in your command prompt. MSNoise 1.3 includes several plots which are available using the msnoise plot command. See Plotting (page 31) to view all plots!

All the available plots are listed using the --help argument:
Options:
--help   Show this message and exit.

Commands:
ccftime       Plots the dv/v (parses the dt/t results)
data_availability  Plots the Data Availability vs time
distance       Plots the REFs of all pairs vs distance
dvv            Plots the dv/v (parses the dt/t results)
interferogram   Plots the interferogram between sta1 and sta2...
mwcs           Plots the mwcs results between sta1 and sta2...
station_map    Plots the station map (very basic)

Same as above, sub-commands have their own --help:

msnoise plot interferogram --help

Usage: [OPTIONS] STA1 STA2

Plots the interferogram between sta1 and sta2 (parses the CCFs)

STA1 and STA2 must be provided with this format: NET.STA !

Options:
-f, --filterid INTEGER  Filter ID
-c, --comp TEXT         Components (ZZ, ZR,...)
-m, --mov_stack INTEGER  Mov Stack to read from disk
-s, --show BOOLEAN      Show interactively?
-o, --outfile TEXT      Output filename (?=auto)
-r, --refilter TEXT     Refilter CCFs before plotting (e.g. 4:8 for
                        filtering CCFs between 4.0 and 8.0 Hz. This will
                        update the plot title.
--help                   Show this message and exit.
5.4.5 New functionality: Dynamic time lags

As before, the dt/t is determined as the slope of the delays vs time lags. The slope is calculated using a weighted linear regression (WLS) through selected points. The selection of points is first based on the time lag criteria. The minimum time lag can either be defined absolutely or dynamically. When dtt.lag is set to "dynamic" in the database, the inter-station distance is used to determine the minimum time lag. This lag is calculated from the distance and a velocity configured (dtt.v). The velocity is determined by the user so that the minlag doesn’t include the ballistic waves. For example if ballistic waves are visible with a velocity of 2 km/s, one could configure dtt.v=1.0. This way, if stations are located 15 km apart, the minimum lag time will be set to 15 s. The dtt.width determines the width of the lag window used. A value of 30.0 means the process will use time lags between 15 and 45 s in the example above, on both sides if configured (dtt.sides), or only causal or acausal parts of the CCF.

Note: It seems obvious that these parameters are frequency-dependent, but they are currently common for all filters!

New parameters have been added to the configuration:
• **dtt_lag**: How is the lag window defined [dynamic]/static (default=static)

• **dtt_v**: If dtt_lag=dynamic: what velocity to use to avoid ballistic waves [1.0]km/s (default=1.0)

• **dtt_minlag**: If dtt_lag=static: min lag time (default=5.0)

• **dtt_width**: Width of the time lag window [30]s (default=30.0)

• **dtt_sides**: Which sides to use [both]/left/right (default=both)

• **dtt_mincoh**: Minimum coherence on dt measurement, MWCS points with values lower than that will not be used in the WLS (default=0.65)

• **dtt_maxerr**: Maximum error on dt measurement, MWCS points with values larger than that will not be used in the WLS (default=0.1)

• **dtt_maxdt**: Maximum dt values, MWCS points with values larger than that will not be used in the WLS (default=0.1)

See also:
The description of the *Compute dt/t* (page 26) step in the workflow.

### 5.4.6 Math updates & bugfixes

Some improvements to the maths have been done for MSNoise 1.3:

• **whiten**: the symmetric hermitian was not properly defined and could lead to a 1 sample shift in the negative frequencies.

• **compute_cc**: it is now possible to define an overlap of the windows

• **compute_cc**: setting windsorizing to -1 now computes the 1-bit normalization of the trace. Reminder: 0: no normalization, N: N*rms clipping.

• **mwcs**: the tapering of the windowed CCF has been improved in order to optimize the calculation for the center of the window.

### 5.4.7 Performance improvements

Improvements in terms of performances have also been done for MSNoise 1.3:

• **new_jobs**: the procedure has been completely rewritten and should be a lot faster, certainly for large (to very-large) archives and/or number of days.

• **keep_all**: if set to Y (=True) in the config, all CCF are now stored in a single HDF5 file, which makes it much nicer to backup/transfer/delete.

• **compute_cc**: if only ZZ components are to be computed, the whitened windows are pre-computed, which makes the process faster. This could lead to memory issues if the job contains a lot of stations, a lot of filters are configured and a large number of windows.

• **compute_mwcs**: The procedure updates the jobs all at once, which brings a big gain in transaction time.
5.4.8 MSNoise has a proper API: Hacking MSNoise

The former `database_tools.py` has been renamed to `api.py` and all the functions are now documented (in MSNoise API (page 41)) so they can be used from the console or from custom scripts.

Using the `msnoise ipython` command, one triggers the start of an IPython notebook in the current project folder. Once in a new notebook, one could

```python
from msnoise.api import connect, get_config
session = connect()
print get_config(session, "mov_stack")
```

to get the current configuration of the `mov_stack` parameter! Enjoy Hacking!

5.4.9 MSNoise is tested

UnitTests are now defined for some (not all, yet) MSNoise functions, and most of the workflow is tested automatically by launching `msnoise test` in a new folder.

This will allow contributors to easily know the code they want to be merged in the next versions of MSNoise doesn’t break anything. Travis-CI runs automatically on every push or pull request made on GitHub.

5.4.10 Upgrading an existing project to MSNoise 1.3

Some users will want to keep their current project without recomputing everything. This requires:

1. adding a few configuration parameters to the database
2. modifying the structure of the `jobs` table.

Running the following command will do both parts for MySQL and only the first part for SQLite:

```
msnoise upgrade_db
```

The second part is a little different if you are using SQLite as it can’t be done automatically. This is because SQLite doesn’t support “ALTER” commands. Ultimately we want the `jobs.type` to be renamed to `jobs.jobtype`. You will have to do this operation manually:

- Open SQLite database browser (`SQLiteManager` extension for Firefox, for example)
- Open the `msnoise.sqlite` file
- Select the `jobs` table
- Edit the `type` field and rename it to `jobtype`
- Ignore the warnings (it should work, although it could fail!)
- Close the database
5.4.11 A note on parallel processing

Although the `msnoise` command accepts the `-t INTEGER` argument to launch a number of threads in parallel, it currently only works with `scan_archive`: `msnoise -t 4 scan_archive` will run the scan on four folders in parallel. For the other steps, one has still to run multiple commands in a console. This should change in the future.

5.5 MSNoise 1.2.5

Release date: 16 June 2014
Release type: bugfix & improvements
Release notes:

5.5.1 BugFixes + Improvements

Some bugfixes (thanks to all reporters) and improvements for this release:

- Bugfix: station ordered by net, sta
- Bugfix: s03 -> remove explicit MSEED
- Bugfix: database_tools: add gps2DistAzimuth import
- Bugfix: s07 -> left, right are now defined from the first plot
- Improvement: Add support for manual DB.ini path
- Improvement: Cleaner s07 plot script
- Improvement: s03: add taper to each slice before FFT in whiten
- Improvement: s01: add headonly=True arg to the read function, should make the archive scan faster
- Minor Documentation update

5.6 MSNoise 1.2.4

Release date: 28 April 2014
Release type: bugfix & improvements
Release notes:

5.6.1 BugFix - Improvement of scan_archive and new_jobs

Thanks to some great early MSNoise adopters who reported problems using the mailing-list, we have identified a few bugs and tricky situations where some steps failed. This is the case for

- scan_archive: there were major issues using Threads, this step uses Process for multiprocessing, which is much safer. The only problem remaining is that there is no more console-logging of the found/identified files. To be corrected soon.
• new_jobs: mostly rewritten, all jobs are properly identified now. There should no more lost jobs!

I’ve also reflected those changes in the documentation.

## 5.7 MSNoise 1.2.3

Release date: 25 Februari 2014

Release type: bugfix

Release notes:

### 5.7.1 BugFix - Compatibility Improvement

Small bugs have been corrected Numpy 1.8.0 is supported (wait for 1.8.1 to avoid numpy/numpy#4276) Obspy 0.9.0 is supported

## 5.8 MSNoise 1.2.2

Release date: 6 November 2013

Release type: bugfix

Release notes:

### 5.8.1 BugFix in DatabaseTools

In s06compute_dtt.py, when updated_days_for_dates is called with pair as “%”, it doesn’t work in database_tools.py. There is no query result back when calling

```python
days = session.query(Job).filter(Job.pair == pair).filter(Job.day >= date1).filter(Job.day <= date2).filter(Job.type == type).filter(Job.lastmod >= lastmod).group_by(Job.day).order_by(Job.day).all().
```

It probably work under windows, but it doesn’t work in Linux.

Thanks for reporting, Xiao, it’s now corrected.

## 5.9 MSNoise 1.2.1

Release date: 23 October 2013

Release type: bugfix

Release notes:
5.9.1 BugFix in Stack

A bug in the Stack procedure has been corrected and the database tools have been updated. There should be a slight performance improvement in the search for updated days within a specified moving-window.

5.10 MSNoise 1.2

Release date: 17 October 2013
Release type: major
Release notes:

5.10.1 Major Release

This release is not backward compatible. The database scheme and the way to handle the DB have changed dramatically to make the process easier.

The installation process is now much simpler. MySQL is no longer required and the database in/outs are handled by sqlalchemy.

From now on, the documentation of the releases will be available on http://msnoise.org/doc.

The steps names have changed in order to allow the imports of the modules. This was needed for the documentation, too.

5.11 MSNoise 1.0

Release date: 25 August 2013
Release type: major
Release notes:

5.11.1 Initial MSNoise Release

MSNoise is now available on GitHub!