# MSNoise How to- A practical guide to getting up and running.

This guide is based from my own experiences setting up MSNoise and is a result of weeks bashing my head against a wall. This guide isn’t an in-depth explanation of what all the functions and do and why parameters are defined as they are, hopefully those documents will come late. This is a basic starting point aimed to get the reader to a point with MSnoise where plots and other outputs can be done. I have done my best to record errors and how I got around them, so your journey through MSNoise can be a little smoother.

This guide assumes the reader has the following se-up: A linux machine with MSNoise and Obspy packages installed. If you don’t have the last two, they are done in the terminal using:

$conda install -c conda-forge obspy –y

$pip install msnoise

Other packages may need to be installed, but using the command:

$msnoise bugreport

It should give a list of required packages and with ticks to indicate whether or not they are on your computer. If not, google is your best friend.

This guide also runs under the assumption that the reader already has seismic data saved in 1 day long mseed format, located and saved into the correct format, or at least one of the data structures outlined in the MSNoise documentation. This data structure needs to be defined in the MSNoise GUI under configuration, but I will be included in the guide later on. Depending on the server you are using this process will be different and will require a more specialised script to account for different stations, channels etc. I used a prefilter when downloading my data although I don’t know how important this is as msnoise offers a prefilter option.

## Setting up sqlite Database

Msnoise gives two options for setting up the database, one using mysql and sql lite. After trial and error using the test data we found that using sql lite allowed us to progress further through the MSNoise workflow documentation as it gave fewer errors when scanning the archive later on. Hence it will be implemented for our project.

## Steps for Creating a Database

To create a database, you must be a 'root'user. This is done by running in the command line:

$sudo mysql -u root -p

This will prompt you for sudo password (UPI one) and then another password, which is the root one (u7$Xe<{d3c9G!]Tr). This then opens up the mysql monitor.To give root permission run:

GRANT ALL PRIVILEGES ON \*.\* TO 'username'@'localhost' IDENTIFIED BY 'password';

To create a new database:

>CREATE DATABASE databasenameyouwant;

For the database I used I called it “msnoise” You can then leave the mysql monitor terminal by typing quit. Now you have created an empty database!

## Installing msnoise

In a new terminal use the cd command to navigate to the folder where your data is saved, mine was called MSNoise, and my data was in a folder in this MSNoise folder. Then enter the command:

$msnoise install

then select option 1 and enter the name of the database you just created.

## MSnoise admin

The msnoise admin GUI offers and interface for configuring settings, setting up filters and checking/scan for new jobs and data. To open the admin, enter into the terminal (in the msnoise directory):

$msnoise admin

Then open your web browser and navigate to <http://localhost:5000/admin/>

An error I got was “this address is already in use”. This normally meant that I had left a MSnoise admin tab open in my browser, so closing it would work. If not I used the following in the terminal to check if an admin process was running:

$ps -ef | grep admin which would give something like the following:

USER1 2325 19762 0 12:07 pts/16 00:00:00 something\_not\_msnoise\_related

USER2 4534 17650 1 0 Dec17 ? 00:00:55 /something/msnoise/related

We can see that the second line is something msnoise related so I can kill it by using the process number (highlighted for convenience):

$kill -9 17650

Alternatively, you want to open admin at a different location by using :

$ msnoise admin -p 5099

and then <http://localhost:5099/admin/> for example. This however can get tedious if you keep swithcing locations, so it is easier to use the same location and grep to kill old admin processes

## Configuring MSNoise

Before being able to populate the station table, MSnoise needs to know what to look for. At a minimum, the follow must be set under configuration in the MSNoise GUI:

* data\_folder set to where ever your data is stored (/home/msnoise/data for example)
* data\_structure set to the type used when naming and storing files (PDF)
* Network: NZ
* Channels: EHZ

We found this if we attempt to populate the table before configuring the location then the location will be set as \* for all the stations, indicating msnoise doesn't get this information from the files, rather from our configuration settings.

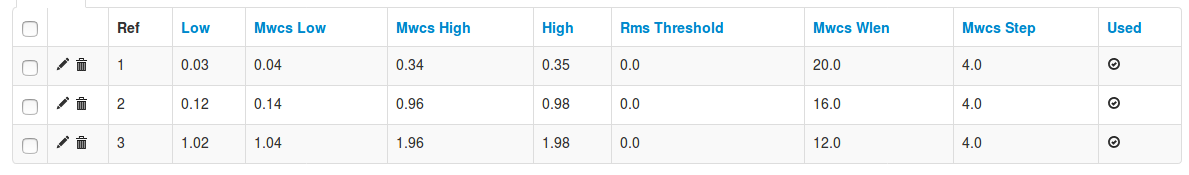
## Msnoise Filters

Before any cross correlations can be run, filters must be set. There are no default settings so I used the filters used by A S Yates in his thesis, SEISMIC VELOCITY CHANGES AT WHITE ISLAND VOLCANO, NEW ZEALAND, USING TEN YEARS OF AMBIENT NOISE INTERFEROMETRY.

Filter 1-Station pair cross-correlation

Filter 2-Auto correlation 0.1-1.0 hz

Filter 3- Auto Correlation 1.0-2.0 hz



In a separate piece I aim to explain what these filters do, how the values can be determined and what impact different values have on the results.

## Populating the station table

Once admin has been configured, the data has been saved in the right place with the right name, we can attempt to populate the station table. First, navigate to the directory where the data is saved using the cd command, then:

$msnoise populate

To check this worked, open the configuration/station tab in the admin interface. This should now have the names and network of the stations we are wanting to analyse.

## Scan-Archive

This command looks for data and tells MSNoise where it is so it can be processed in later steps. Once again, in the MSNoise directory, for the first run use:

$ msnoise -t 2 scan-archive --init or otherwise:

$ msnoise -t 2 scan-archive

For some reason, my computer has the quirk where I need to replace underscore with dashes, so if you get the error "command does not exist", try swapping the two.

If you have downloaded more data to update a preexisiting archive but scan-archive isnt picking up on it, adjust the “crondays” setting in the configuration to add data for x number of days before today to now.

## New Jobs

Once msnoise has scanned the archive it can then look for new data on which to perform cross correlations on. This is achieved using the command (in the msnoise directory of course):

$ msnoise new-jobs

If this worked, you should be able to see jobs listed in the database-job section of MSNoise Admin GUI.

## Compute Cross Correlations

Before any cross correlations can be calculate filters must first be applied. I don't really understand why certain filters would be used over others, other than looking at different frequencies based on what part of the frequency spectrum you are looking at. For a starting point is used the filters from the thesis mentioned above, as there was a range, so I hoped that later on I could compare the outputs and see where some are better than others and what the filter parameters do.

To get the cross correlations:

$msnoise compute-cc

The other error I got when running $ compute-cc, was the error saying something along the lines of something being greater than RMS. this is confusing as according to the latest version of MSNoise, the RMS options doesn't do anything, yet if I add a value for RMS as 0 it removes this error.

Other error- no response to remove- fixed this by selecting setting remove instrument response to N in the configuration tab

## Stacks

Stacking adds up the correlations for the set length for defined periods, ie 1, 2, 5, 10 days. This helps to remove errors as they will add destructively, whereas trends will add up constructively. The length of the stacks will influence the resolution, with short stacks showing rapidly fluctuations trends whereas longer windows will show more stable long term trends.

The length of the stacks is defined in the configuration tab of the msnoise GUI. To stack:

$ msnoise stack -r -m -i10

The -r refers to reference, -m refers to moving and the -i 10 tells the compute to look for data added in the last 10 days. This process is working if job titles pop up in the terminal. For some time following the $msnoise stack (-r –m) would say ‘lets stack’ and immediately after say “finished”, however there was no evidence of it doing anything. This is because we didn’t define the interval term, which tells msnoise how old data or jobs should be that it can stack, ie i-10 would stack jobs completed from the last 10 days. To reset/recalculate all stacks:

$msnoise stack–r –m –I 999 can be used.

## Moving Window Cross Section

$ msnoise compute-mwcs

MWCS technique compares the moving window stacks with the reference stacks to obtain the central time lag, the measured delay, its error and the mean coherence of the segment.

## Compute DTT

This function computes the change in lag time from the reference as a timeseries. In order to run this command, the MWCS step must have been run beforehand. To compute DTT:

$msnoise compute\_dtt -i \*some integer\*

Like the stacking command, if I just ran $msnoise compute\_dtt, I would get a message saying “computing dtt’ and then immediately after “finished:compute dtt”, with no files changed or added. This is from the lack of interval term, so I normally add -I 10 and this seems to solve the problem.

If using the dynamic over the static lag window method, station coordinates must be known. These can be added into the station table in the MSNoise admin GUI using latitude and longitude coordinates.

An error I occurred was the compute-dtt would run into an error along the liens of “no component reshape”, when I was only dealing with data from one station. By downloading another station’s data for the same period and it fixed this error.

## Incorporating new data

Once you have downloaded new data, it can be added and computed like the original data by following the workflow, as summarized:

* $ msnoise -t 2 scan-archive
* $msnoise new-jobs
* $msnoise compute-cc
* $msnoise msnoise stack –r –m –i10
* $msnoise compute-mwcs
* $msnoise compute-dtt –i10 (number indictes how many days prior the job was marked to do, ie –i10 would indicates all jobs that were marked to do in the last 10 days)
* and now you are ready to plot!

## Adding New Filters

To avoid redoing preexisting filters, set all old filters used value to false and the new filters used value to true in te filters tab in the msnoise admin GUI. From there:

* $msnoise reset CC –all
* $msnoise compute-cc
* $msnoise stack –r –m –i999
* $msnoise compute-mwcs
* $msnoise compute-dtt –i999

Then set all the other used value back to true and you are good to go!

Updating REF stacks

Done simply using:

$msnoise stack -r -i 999

# Changing Configuration Parameters

Depending on what stage of the workflow a certain parameter is used, that stage and then the following stages need to be reset. Resetting can be done with the following command:

$msnoise reset CC/DTT/MWCS --all

Selecting what job type you want to reset.

Ie if a parameter is used in the compute-cc step, then you would need to reset CC, MWCS and DTT and then follow the work flow beginning at compute-cc through.

The following lists the parameters used in each step, descriptions are from the MSNoise Documentation:

## Compute-cc:

* 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.<br>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

## Stack:

* 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)
* 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)

## Compute-MWCS-all defined under filter tab in GUI

* 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

## Compute-DT/T

* 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)

## 