

**EQcorrscan**

*Correlation for earthquake  
detection in Python*

# **EQcorrscan Documentation**

***Release 0.0.8***

**Calum John Chamberlain**

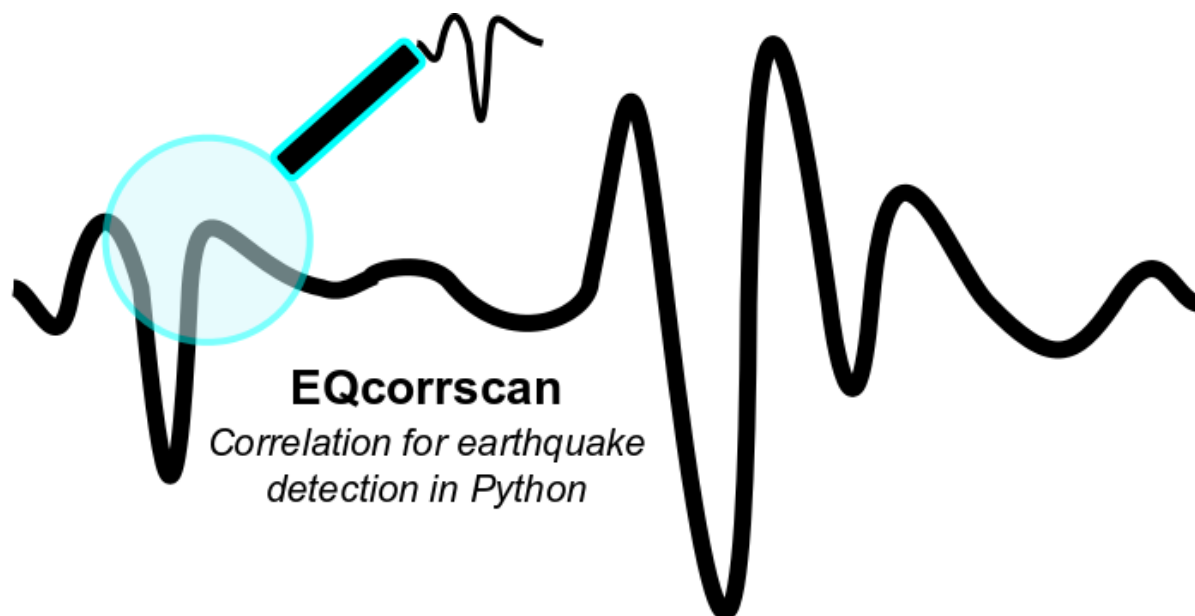
November 18, 2015



## CONTENTS

<b>1</b>	<b>EQcorrscan</b>	<b>3</b>
<b>2</b>	<b>References</b>	<b>5</b>
<b>3</b>	<b>Contents:</b>	<b>7</b>
3.1	Introduction to the EQcorrscan package . . . . .	7
3.2	EQcorrscan tutorial . . . . .	8
3.3	Core . . . . .	12
3.4	Utils . . . . .	22
	<b>Python Module Index</b>	<b>41</b>







## EQCORRSCAN

A python package to conduct match-filter earthquake detections. Codes are stored on github, the bleeding edge master is [here](#), or the latest stable(ish) release can be found [here](#)

This package contains routines to enable the user to conduct match-filter earthquake detections using [Obspy](#) bindings when reading and writing seismic data, and the correlation routine in [openCV](#). Neither of these packages are installed by this software, due to a range of licences being implimented. However, both are open-source and should be installed before using this package. This package was written to impliment the matlab routines used by Chamberlain et al. (2014) for the detection of low-frequency earthquakes.

Also within this package are:

- Clustering routines for seismic data;
- Peak finding algorithm (basic);
- Automatic amplitude picker for local magnitude scale;
- [Seisan](#) S-file integration for database management and routine earthquake location;
- Stacking routines including phase-weighted stacking based on Thurber et al. (2014);
- Brightness based template creation based on the work of Frank et al. (2014)

This package is written by Calum Chamberlain of Victoria University of Wellington, and is distributed under the LGPL GNU Licence, Copyright Calum Chamberlain 2015.





## REFERENCES

- CJ Chamberlain, DR Shelly, J Townend, TA Stern (2014) [Lowfrequency earthquakes reveal punctuated slow slip on the deep extent of the Alpine Fault, New Zealand](#), *G-cubed*, doi:10.1002/2014GC005436
- Thurber, C. H., Zeng, X., Thomas, A. M., & Audet, P. (2014). [PhaseWeighted Stacking Applied to LowFrequency Earthquakes](#), *BSSA*, doi:10.1785/0120140077.
- Frank, W. B., & Shapiro, N. M. (2014). [Automatic detection of low-frequency earthquakes \(LFEs\) based on a beamformed network response](#), *Geophysical Journal International*, 197(2), 1215-1223, doi:10.1093/gji/ggu058.



## CONTENTS:

### 3.1 Introduction to the EQcorrscan package

This document is designed to give you an overview of the capabilities and implementation of the EQcorrscan python module.

#### 3.1.1 Why EQcorrscan?

EQcorrscan is designed to compute matched-filter detections of earthquakes, or any seismic signal (explosions work *really* well) by comparing templates with continuous data. The main benefit of EQcorrscan is the level of parallelisation that can be achieved. By exploiting the fact that each template does not rely on any other template, detections from a single template through a day of seismic data can be computed in parallel. By computing these in parallel rather than a single template through multiple days we reduce IO load. At a low level, each time-step is computed in parallel by using the openCV matchTemplate function. The net result is that these functions are *very* scalable, we have obtained a speed-up from 2 months to 10 hours by migrating from a small cluster to a large one (for a 6.5 year long continuous dataset and 800 templates).

The authors of EQcorrscan foresee this project as an open repository for the development of software for the detection and analysis of repeating and near-repeating earthquakes. This repository will continue to grow and develop and any and all help/criticism will be appreciated.

We have a long way to go with this project - if you want to get involved the best place to start, and the most valuable thing for your understanding, and for the health of this repository would be to contribute tests and documentation. Ideally we would like to have one test for every function!

#### 3.1.2 Installation

A fresh install should be as simple as:

##### **pip install eqcorrscan**

Most codes should work without any effort on your part. However you may need to install the openCV-python package yourself.

This install has only been tested on Linux and OSX machines. You should be prepared for small differences in the results of your correlations relating to floating-point truncation differences between 32 and 64-Bit machines.

If you plan to run the `bright_lights` or generating a synthetic grid of templates you will need to have grid csv files, which the authors have previously used NonLinLoc to generate. This is not provided here and should be sourced from [NonLinLoc](#). This will provide the `Grid2Time` routine which is required to set-up a lag-time grid for your velocity model. You should read the NonLinLoc documentation for more information regarding how this process works and the input files you are required to give.

### 3.1.3 Functions

This package is divided into sub-directories of *core* and *utils*. The *utils* directory contains simple functions for integration with *seisan*, these are in the *Sfile\_util.py* module and functions therein which are essentially barebones and do not have the full functionality that *seisan* can handle. *utils* also contains a simple peak-finding algorithm *find\_peaks.py* which looks for peaks within noisy data above a certain threshold and within windows. Many other functions have been added to this module to handle the analysis of repeating and near-repeating earthquakes, including stacking routines, clustering algorithms, magnitude calculation both by amplitude picking and by singular value decomposition. I recommend you take a look in here to see if any of it is useful. There are also some plotting routines that make handling large datasets a little simpler. Most recently I have added a simple synthetic seismogram generator, which is currently my main project focus.

Since earlier versions the *core* modules have moved away from using parameter files, and instead rely on explicit argument calls. The parameter files are still included but not documented here (see inside the *par* files), and remain useful when generating batch scripts (see the scripts in the github repo).

Within *core* you will find the core routines to generate templates, (*template\_gen*) search for likely templates (*bright\_lights*) and compute cross-channel correlations from these templates (*match\_filter*). The *bright\_lights* and *match\_filter* submodules have been designed with parallel computing in mind, to the extent that the more cores and machines you have running them the better. These rely on the python multiprocessing module to handle parallelisation at lower-levels. You can also do some 'brute-force' parallelisation on a day level when computing detections over multiple days. I tend to run one day per node of a cluster computer, with each day running templates in parallel.

## 3.2 EQcorrscan tutorial

Welcome to EQcorrscan - this package is designed to compute earthquake detections using a paralleled matched-filter network cross-correlation routine. The inner loop of this package is the cross-correlation of templates of seismic data with day-long seismic data. This inner function is the `openCV.match_template` function - this appears to be a well optimized cross-correlation function, and is written in c++. Cross-correlations are computed in the frequency domain for large datasets, for which a day of seismic data usually qualifies.

Before continuing with this tutorial please check that you have installed all the pre-requisite modules, as not all will be installed by the *setup.py* file. The list of these is in the Introduction section of this documentation.

As you will see, this package is divided into two main sub-modules, the Core and Utils sub-modules. The Core sub-module contains the main, high-level functions:

**bright\_lights** A brightness based template detection routine;

**template\_gen** A series of routines to generate templates for match-filter detection from continuous or cut data, with pick-times defined either manually, or from a *Seisan* s-file;

**match\_filter** The main matched-filter routines, this is split into several smaller functions to allow python based parallelisation;

**lag\_calc** Routines for calculating optimal lag-times for events detected by the match-filter routine, these lags can then be used to define new picks for high accuracy relocations.

The Utils sub-module contains useful, but small functions. These functions are rarely cpu intensive, but perform vital operations, such as reading *Seisan* s-files, finding peaks in noisy data, converting a *seisan* database to *hypoDD* formatted files and computing cross-correlations between detections for *hypoDD* (a double difference relocation software), calculating magnitudes, clustering detections, stacking detections, making pretty plots, and processing seismic data in the same way repeatedly using *Obspy*'s functionality.

### 3.2.1 Matched-filter detection

In this section we will discuss generating a pair of templates from two *Seisan* s-files before using these templates to scan for similar earthquakes within a day of data. This small example does not truly exploit the parallel operations within this package however, so you would be encouraged to think about where parallel operations occur (*hint*,

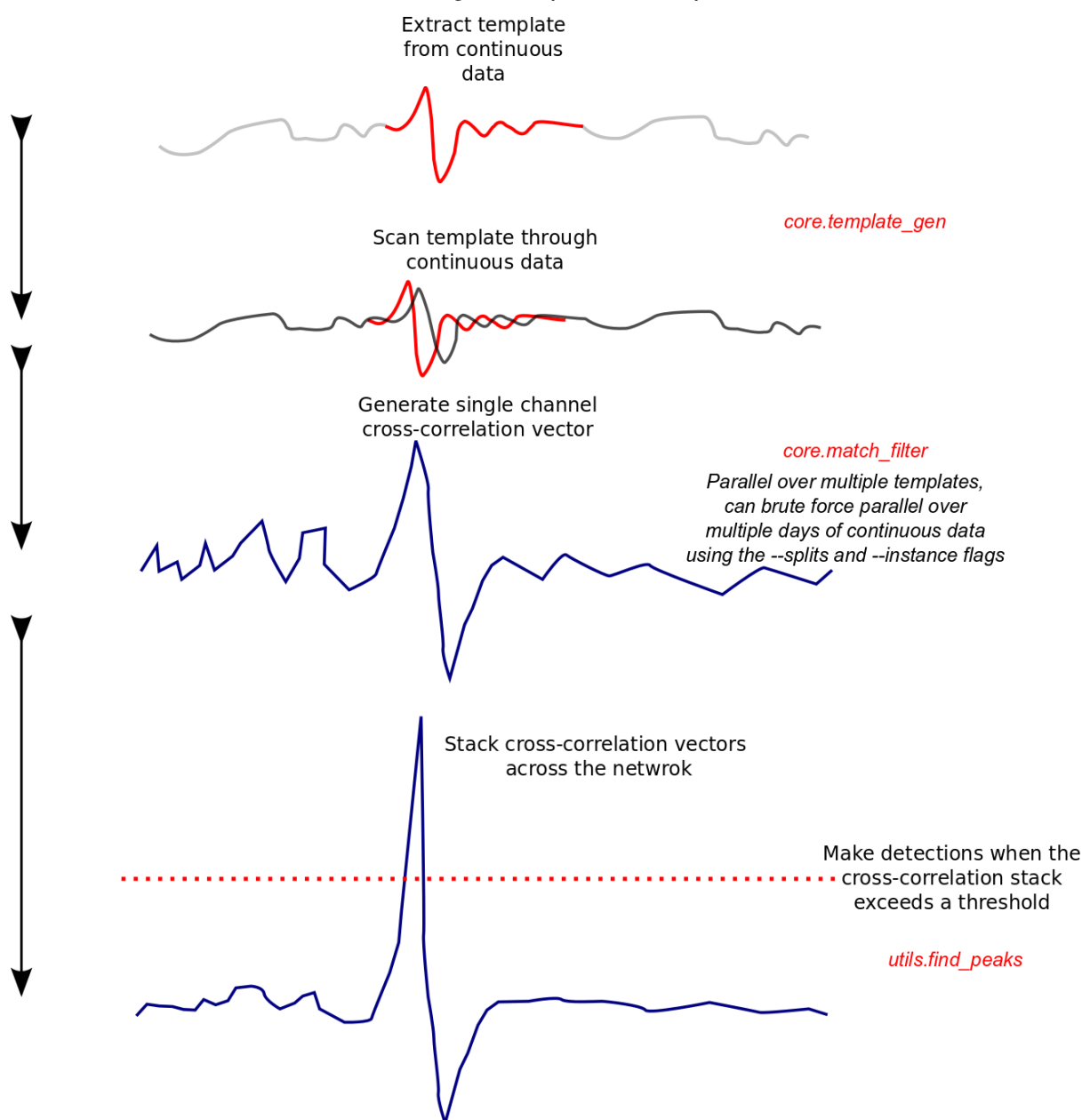
the code can run one template per cpu), and why there are `--instance` and `--splits` flags in the other scripts in the github repository (\*hint, if you have heaps of memory and cpus

you can do some brute force day parallelisation!\*)).

The following script is included in the top-level directory alongside the full-scripts used by the author to generate a 6.5 year long catalogue of low-frequency earthquakes for the central Southern Alps of New Zealand.

This tutorial script highlights the ability of the match-filter method in detecting earthquakes of near-repeating nature. The dataset is a day of data taken from the New Zealand national database, and the Southern Alp Microearthquake Borehole Array (SAMBA) network (Boese et al. 2012). This day was found to contain a swarm of earthquakes, as published by Boese et al. (2014), the s-files provided are two of these events.

The main processing flow is outlined in the figure below, note the main speedups in this process are achieved by running multiple templates at once, however this increases memory usage. If memory is a problem there are flags (`mem_issue`) in the `match_filter.py` source that can be turned on - the codes will then write temporary files, which is slower, but can allow for more data crunching at once, your trade-off, your call.



## 3.2.2 References

- CM Boese, J Townend, E Smith, T Stern (2012). Microseismicity and stress in the vicinity of the Alpine Fault, central Southern Alps, New Zealand, *JGR*, doi:10.1029/2011JB008460
- CM Boese, KM Jacobs, EGC Smith, TA Stern, J Townend (2014). Background and delayed-triggered swarms in the central Southern Alps, South Island, New Zealand, *G-cubed*, doi:10.1002/2013GC005171

```
#!/usr/bin/env python
r"""Tutorial This script is designed as a tutorial to highlight how to\
call the main functions within the EQcorrscan module. In this tutorial
we will see how to generate a template and run this through the
matched-filter routine.
The template will be generated from a pre-picked earthquake, however there
are other ways to generate templates, for example this package also contains
a simple brightness function that is designed to scan continuous seismic
data and look for impulsive energy originating from a discrete point in a
seismic velocity model.

This package is distributed under the LGPL v3.0, by using this script and the
functions contained within the EQcorrscan package you implicitly accept the
licence. For the full wording of the licence refer to the licence.txt file.

Copyright 2015 Calum Chamberlain

This file is part of EQcorrscan.

EQcorrscan is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

EQcorrscan is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with EQcorrscan. If not, see <http://www.gnu.org/licenses/>.

"""

# First we import the required modules:
import os
from obspy import read
from obspy import Stream
from eqcorrscan.core import template_gen, match_filter
# Before calling these module imports for parameter files you should insert
# your own path into sys.path so that we find your parameter files.
from eqcorrscan.utils import pre_processing, Sfile_util
import glob

# Set up the default parameters - these used to be stored in parameter files
debug = 3 # High debug level should output lots to keep you informed
threshold = 8.0 # Threshold level as MAD multiplier
threshtype = 'MAD' # Threshold type, in this case Median Absolute Deviation
trig_int = 6.0 # Minimum trigger interval for one template in seconds

# Now we find the s-file we want to use to generate a template from
data_directory = os.path.join('test_data', 'tutorial_data')
sfiles = glob.glob(os.path.join(data_directory, '*.L.S*'))
print sfiles
```

```

templates = []
template_names = []
for i, sfile in enumerate(sfiles):
    # Read in the picks from the S-file, note, in the full case one fo the main\
    # functions in template_gen would be used rather than this, but for\
    # the tutorial we will read in the data here - also note that this\
    # template generation is inefficient for multiple templates, if using\
    # daylong data for multiple templates you would want to only read\
    # the seismic data once and cut it multiple times.
    picks=Sfile_util.readpicks(sfile)
    for pick in picks:
        print pick
        if not 'wavefiles' in locals():
            wavefiles = glob.glob(os.path.join(data_directory,
                                                '._'.join([pick.station, '*'])))
        else:
            wavefiles += glob.glob(os.path.join(data_directory,
                                                '._'.join([pick.station, '*'])))

    wavefiles = list(set(wavefiles))
    for wavefile in wavefiles:
        print '._'.join(['Reading data from', wavefile])
        if 'st' not in locals():
            st = read(wavefile)
        else:
            st += read(wavefile)

    st = st.merge(fill_value='interpolate')
    day = st[0].stats.starttime.date

    # Process the data with our required parameters
    for tr in st:
        tr = pre_processing.dayproc(tr, 1.0, 20.0, 3, 100.0,\
                                   debug, day)

    # Use the template generation function to cut our templates
    template = template_gen._template_gen(picks, st, length=1.0, swin='all',
                                          prepick=0.1, plot=True)

    # This will generate an obspy.Stream object
    # Append this Stream to the list of templates
    templates += [template]
    template_names.append('._'.join(['tutorial', str(i)]))

    # Save template for later
    template.write(os.path.join(data_directory, '._'.join([template_names[i],
                                                           'template.ms'])),
                  format='MSEED')

    # Delete excess information from memory If you are re-using this script
    # with the same templates you should be able to comment out this loop
    # once you have generated your templates once.
    del template, st

# Extract the stations from the templates
for template in templates:
    if not 'stachans' in locals():
        stachans = [(tr.stats.station, tr.stats.channel) for tr in template]
    else:
        stachans += [(tr.stats.station, tr.stats.channel) for tr in template]

# Make this a unique list
stachans = list(set(stachans))

# Read in the continuous data for these station, channel combinations
for stachan in stachans:

```

```
data_file = ''.join([stachan[0], '.*.*', stachan[1][-1], '.*'])
data_file = os.path.join(data_directory, data_file)
print ' '.join(['Reading data from:', data_file])
# Generate a new stream object and add to it
if 'st' not in locals():
    st = read(data_file)
else:
    st += read(data_file)

# Merge the data to account for miniseed files being written in chunks
# We need continuous day-long data, so data are padded if there are gaps
st = st.merge(fill_value='interpolate')

# Work out what day we are working on, required as we will pad the data to be daylong
day = st[0].stats.starttime.date

# Process the data in the same way as the template
for tr in st:
    tr = pre_processing.dayproc(tr, 1.0, 20.0, 3, 100.0, \
                               debug, day)

# Compute detections
detections = match_filter.match_filter(template_names, templates, st,
                                       threshold, threshtype, trig_int,
                                       plotvar=True, cores=2, tempdir=False,
                                       debug=debug, plot_format='pdf')

# We now have a list of detections! We can output these to a file to check later
f = open('tutorial_detections.csv', 'w')
for detection in detections:
    line = ', '.join([detection.template_name, str(detection.detect_time),
                     str(detection.detect_val), str(detection.threshold),
                     str(detection.no_chans)])
    f.write(line)
    print line
    f.write(os.linesep)
f.close()
```

## 3.3 Core

Core programs for the EQcorrscan project.

### 3.3.1 bright\_lights

Code to determine the brightness function of seismic data according to a three-dimensional travel-time grid. This travel-time grid should be generated using the `grid2time` function of the `NonLinLoc` package by Anthony Lomax which can be found here: <http://alomax.free.fr/nlloc/> and is not distributed within this package as this is a very useful stand-alone library for seismic event location.

This code is based on the method of Frank & Shapiro 2014.

Code generated by Calum John Chamberlain of Victoria University of Wellington, 2015.

#### Note

#### Pre-requisites:

- gcc - for the installation of the openCV correlation routine



- python-cv2 - Python bindings for the openCV routines
- python-joblib - used for parallel processing
- **python-obspy - used for lots of common seismological processing**
  - **requires:**
    - \* numpy
    - \* scipy
    - \* matplotlib
- NonLinLoc - used outside of all codes for travel-time generation

Copyright 2015 Calum Chamberlain

This file is part of EQcorrscan.

EQcorrscan is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

EQcorrscan is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with EQcorrscan. If not, see <<http://www.gnu.org/licenses/>>.

`bright_lights._cum_net_resp (node_lis, instance=0)`

Function to compute the cumulative network response by reading the saved energy .npy files.

#### Parameters

- **node\_lis** (*np.ndarray*) – List of nodes (ints) to read from
- **instance** (*Int*) – Instance flag for parallelisation, defaults to 0.

**Returns** *np.ndarray* cum\_net\_resp, list of indices used

`bright_lights._find_detections (cum_net_resp, nodes, threshold, thresh_type, samp_rate, realstations, length)`

Function to find detections within the cumulative network response according to Frank et al. (2014).

#### Parameters

- **cum\_net\_resp** (*np.ndarray*) – Array of cumulative network response for nodes
- **nodes** (*list of tuples*) – Nodes associated with the source of energy in the

`cum_net_resp`: type `threshold`: float :param `threshold`: Threshold value :type `thresh_type`: str :param `thresh_type`: Either MAD (Median Absolute Deviation) or abs (absolute) or RMS (Root Mean Squared) :type `samp_rate`: float :param `samp_rate`: Sampling rate in Hz :type `realstations`: list of str :param `realstations`: List of stations used to make the cumulative network response, will be reported in the DETECTION :type `length`: float :param `length`: Maximum length of peak to look for in seconds

**Returns** detections as :class: DETECTION

`bright_lights._node_loop (stations, lags, stream, clip_level, i=0, mem_issue=False, instance=0, plot=False)`

Internal function to allow for parallelisation of brightness.

#### Parameters

- **stations** (*list*) – List of stations to use.
- **lags** (*np.ndarray*) – List of lags where lags[i:] are the lags for stations[i].
- **stream** – Data stream to find the brightness for.
- **clip\_level** (*float*) – Upper limit for energy as a multiplier to the mean

energy. :type i: int :param i: Index of loop for parallelisation. :type mem\_issue: bool :param mem\_issue: If True will write to disk rather than storing data in RAM. :type instance: int :param instance: instance for bulk parallelisation, only used if mem\_issue=true. :type plot: bool :param plot: Turn plotting on or off, defaults to False.

**Returns** (i, energy (np.ndarray))

`bright_lights._read_tt (path, stations, phase, phaseout='S', ps_ratio=1.68, lags_switch=True)`

Function to read in .csv files of slowness generated from Grid2Time(part of NonLinLoc by Anthony Lomax) and convert this to a useful format here.

It should be noted that this can read either P or S travel-time grids, not both at the moment.

#### Parameters

- **path** (*str*) – The path to the .csv Grid2Time outputs
- **stations** (*list*) – List of station names to read slowness files for.
- **phaseout** (*str*) – What phase to return the lagtimes in
- **ps\_ratio** (*float*) – p to s ratio for conversion
- **lags\_switch** (*Bool*) – Return lags or raw travel-times, if set to true will return lags.

**Returns** list stations, list of lists of tuples nodes,

**Class** 'numpy.array' lags station[1] refers to nodes[1] and

lags[1] nodes[1][1] refers to station[1] and lags[1][1]nodes[n][n] is a tuple of latitude, longitude and depth

`bright_lights._resample_grid (stations, nodes, lags, mindepth, maxdepth, corners, resolution)`

Function to resample the lagtime grid to a given volume. For use if the grid from Grid2Time is too large or you want to run a faster, downsampled scan.

**Parameters** **stations** (*list*) – List of station names from in the form where stations[i]

refers to nodes[i][:] and lags[i][:] :type nodes: list, tuple :param nodes: List of node points where nodes[i] refers to stations[i] and nodes[:][0] is latitude in degrees, nodes[:][1] is longitude indegrees, nodes[:][2] is depth in km. :type lags: :class: 'numpy.array' :param lags: Array of arrays where lags[i][:] refers to stations[i].lags[i][j] should be the delay to the nodes[i][j] for stations[i] in seconds. :type mindepth: float :param mindepth: Upper limit of volume :type maxdepth: float :param maxdepth: Lower limit of volume :type corners: matplotlib.Path :param corners: matplotlib path of the corners for the 2D polygon to cut to in lat and long

**Returns** list stations, list of lists of tuples nodes, :class:

'numpy.array' lags station[1] refers to nodes[1] and lags[1]nodes[1][1] refers to station[1] and lags[1][1]nodes[n][n] is a tuple of latitude, longitude and depth.

`bright_lights._rm_similarlags (stations, nodes, lags, threshold)`

Function to remove those nodes that have a very similar network moveout to another lag.

Will, for each node, calculate the difference in lagtime at each station at every node, then sum these for each node to get a cumulative difference in network moveout. This will result in an array of arrays with zeros on the diagonal.

**Parameters** **stations** (*list*) – List of station names from in the form where stations[i]

refers to nodes[i][:] and lags[i][:] :type nodes: list, tuple :param nodes: List of node points where nodes[i] refers to stations[i] and nodes[:][0] is latitude in degrees, nodes[:][1] is longitude indegrees, nodes[:][2] is depth in km. :type lags: :class: 'numpy.array' :param lags: Array of arrays where lags[i][:] refers to stations[i].lags[i][j] should be the delay to the nodes[i][j] for stations[i] in seconds :type threshold: float :param threshold: Threshold for removal in seconds

**Returns** list stations, list of lists of tuples nodes, :class:

'numpy.array' lags station[1] refers to nodes[1] and lags[1]nodes[1][1] refers to station[1] and lags[1][1]nodes[n][n] is a tuple of latitude, longitude and depth.

`bright_lights._rms (array)`  
Calculate RMS of array

`bright_lights.brightness (stations, nodes, lags, stream, threshold, thresh_type, template_length, template_saveloc, coherence_thresh, coherence_stations=['all'], coherence_clip=False, gap=2.0, clip_level=100, instance=0, pre_pick=0.2, plotsave=True, cores=1)`

Function to calculate the brightness function in terms of energy for a day of data over the entire network for a given grid of nodes.

Note data in stream must be all of the same length and have the same sampling rates.

**Parameters** `stations` (*list*) – List of station names from in the form where `stations[i]`

refers to `nodes[i][:]` and `lags[i][:]` :type nodes: list, tuple :param nodes: List of node points where `nodes[i]` refers to `stations[i]` and `nodes[:,0]` is latitude in degrees, `nodes[:,1]` is longitude in degrees, `nodes[:,2]` is depth in km. :type lags: :class: 'numpy.array' :param lags: Array of arrays where `lags[i][:]` refers to `stations[i].lags[i][j]` should be the delay to the nodes `nodes[i][j]` for stations `stations[i]` in seconds. :type stream: :class: `obspy.Stream` :param data: Data through which to look for detections. :type threshold: float :param threshold: Threshold value for detection of template within the brightness function :type thresh\_type: str :param thresh\_type: Either MAD or abs where MAD is the Median Absolute Deviation and abs is an absolute brightness. :type template\_length: float :param template\_length: Length of template to extract in seconds :type template\_saveloc: str :param template\_saveloc: Path of where to save the templates. :type coherence\_thresh: tuple of floats :param coherence\_thresh: Threshold for removing incoherent peaks in the

network response, those below this will not be used as templates. Must be in the form of (a,b) where the coherence is given by:  $a \cdot kchan / b$  where `kchan` is the number of channels used to compute the coherence

#### Parameters

- **coherence\_stations** (*list*) – List of stations to use in the coherence thresholding - defaults to 'all' which uses all the stations.
- **coherence\_clip** (*Start and end in seconds of data to window around, defaults to False, which uses all the data given.*) – tuple
- **pre\_pick** (*float*) – Seconds before the detection time to include in template
- **plotsave** (*bool*) – Save or show plots, if False will try and show the plot on screen - as this is designed for bulk use this is set to True to save any plots rather than show them if you create them - changes the backend of matplotlib, so if is set to False you will see NO PLOTS!
- **core** – Number of cores to use, defaults to 1.
- **clip\_level** (*float*) – Multiplier applied to the mean deviation of the energy as an upper limit, used to remove spikes (earthquakes, lightning, electrical spikes) from the energy stack.
- **gap** (*float*) – Minimum inter-event time in seconds for detections

**Returns** list of templates as :class: `obspy.Stream` objects

`bright_lights.coherence (stream_in, stations=['all'], clip=False)`

Function to determine the average network coherence of a given template or detection. You will want your stream to contain only signal as noise will reduce the coherence (assuming it is incoherent random noise).

#### Parameters

- **stream** (*obspy.Stream*) – The stream of seismic data you want to calculate the coherence for.
- **stations** (*List of String*) – List of stations to use for coherence, default is all
- **clip** (*Tuple of Float*) – Default is to use all the data given - tuple of start and end in seconds from start of trace

**Returns** float - coherence, int number of channels used

### 3.3.2 template\_gen

Function to generate template waveforms and information to go with them for the application of cross-correlation of seismic data for the detection of

repeating events.

Part of the EQcorrscan module to read nordic format s-files. EQcorrscan is a python module designed to run match filter routines for seismology, within it are routines for integration to seisan and obspy. With obspy integration (which is necessary) all main waveform formats can be read in and output.

This main section contains a script, LFE\_search.py which demonstrates the usage of the built in functions from template generation from picked waveforms through detection by match filter of continuous data to the generation of lags to be used for relative locations.

The match-filter routine described here was used a previous Matlab code for the Chamberlain et al. 2014 G-cubed publication. The basis for the lag-time generation section is outlined in Hardebeck & Shelly 2011, GRL.

Code generated by Calum John Chamberlain of Victoria University of Wellington, 2015.

#### Note

#### Pre-requisites:

- gcc - for the installation of the openCV correlation routine
- python-cv2 - Python bindings for the openCV routines
- python-joblib - used for parallel processing
- **python-obsPy - used for lots of common seismological processing**
  - **requires:**
    - \* numpy
    - \* scipy
    - \* matplotlib
- NonLinLoc - used outside of all codes for travel-time generation

Copyright 2015 Calum Chamberlain

This file is part of EQcorrscan.

EQcorrscan is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

EQcorrscan is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with EQcorrscan. If not, see <<http://www.gnu.org/licenses/>>.

`template_gen._template_gen(picks, st, length, swin, prepick=0.05, plot=False)`

Function to generate a cut template in the obspyStream class from a given set of picks and data, also in an obspy streamclass. Should be given pre-processed data (downsampled and filtered)

#### Parameters

- **picks** – Picks to extract data around
- **st** – Stream to extract templates from

- **length** (*float*) – Length of template in seconds
- **swin** (*string*) – P, S or all
- **prepick** (*float*) – Length in seconds to extract before the pick timedefault is 0.05 seconds
- **plot** (*bool*) – To plot the template or not, default is True

```
template_gen.extract_from_stack(stack, template, length, pre_pick, pre_pad,
                                Z_include=False, pre_processed=True, samp_rate=False,
                                lowcut=False, highcut=False, filt_order=False)
```

Function to extract a new template from a stack of previous detections. Requires the stack, the template used to make the detections for the stack, and we need to know if the stack has been pre-processed.

**Parameters** **stack** (:class:obspy.Stream) – Waveform stack from detections. Can be of any length and

can have delays already included, or not. :type template: :class:obspy.Stream :param template: Template used to make the detections in the stack. Will use the delays of this for the new template. :type length: float :param length: Length of new template in seconds :type pre\_pick: float :param pre\_pick: Extract additional data before the detection, seconds :type pre\_pad: float :param pre\_pad: Pad used in seconds when extracting the data, e.g. the time before the detection extracted. If using clustering.extract\_detections this half the length of the extracted waveform. :type Z\_include: bool :param Z\_include: If True will include any Z-channels even if there is no template for this channel, as long as there is a template for this station at a different channel. If this is False and Z channels are included in the template Z channels will be included in the new template anyway. :type pre\_processed: bool :param pre\_processed: Have the data been pre-processed, if True (default)

then we will only cut the data here.

#### Parameters

- **samp\_rate** (*float*) – If pre\_processed=False then this is required, desired sampling rate in Hz, defaults to False.
- **lowcut** (*float*) – If pre\_processed=False then this is required, lowcut in Hz, defaults to False
- **highcut** (*float*) – If pre\_processed=False then this is required, highcut in

Hz, defaults to False :type filt\_order: int :param filt\_order: If pre\_processed=False then this is required, filter

order, defaults to False

**Returns** :class:obspy.Stream Newly cut template

```
template_gen.from_contbase(sfile, contbase_list, lowcut, highcut, samp_rate, filt_order, length,
                           prepick, swin, debug=0)
```

Function to read in picks from sfile then generate the template from the picks within this and the wavefiles from the continuous database of day-long files. Included is a section to sanity check that the files are daylong and that they start at the start of the day. You should ensure this is the case otherwise this may alter your data if your data are daylong but the headers are incorrectly set.

#### Parameters

- **sfile** (*string*) – sfilename must be the path to a seisan nordic type s-file containing waveform and pick information, all other arguments can be numbers save for swin which must be either P, S or all (case-sensitive).
- **contbase\_list** (*List of tuple of string*) – List of tuples of the form

['path', 'type', 'network']. Where path is the path to the continuous database, type is the directory structure, which can be either Yyyy/Rjjj.01, which is the standard IRIS Year, julian day structure, or, yyyymmdd

which is a single directory for every day. :type lowcut: float :param lowcut: Low cut (Hz), if set to None will look in template

defaults file

#### Parameters

- **lowcut** – High cut (Hz), if set to None will look in templatedefaults file
- **samp\_rate** (*float*) – New sampling rate in Hz, if set to None will look in template defaults file
- **filt\_order** (*int*) – Filter level, if set to None will look in template defaults file
- **length** (*float*) – Extract length in seconds, if None will look in templatedefaults file.
- **prepick** (*float*) – Pre-pick time in seconds
- **swin** (*str*) – Either ‘all’, ‘P’ or ‘S’, to select which phases to output.
- **debug** (*int*) – Level of debugging output, higher=more

`template_gen.from_sfile(sfile, lowcut, highcut, samp_rate, filt_order, length, swin, debug=0)`

Function to read in picks from sfile then generate the template from the picks within this and the waveform found in the pick file.

**Parameters** **sfile** (*string*) – sfilename must be the

path to a seisan nordic type s-file containing waveform and pick information. :type lowcut: float :param lowcut: Low cut (Hz), if set to None will look in template

defaults file

#### Parameters

- **lowcut** – High cut (Hz), if set to None will look in templatedefaults file
- **samp\_rate** (*float*) – New sampling rate in Hz, if set to None will look in template defaults file
- **filt\_order** (*int*) – Filter level, if set to None will look in template defaults file
- **swin** (*str*) – Either ‘all’, ‘P’ or ‘S’, to select which phases to output.
- **length** (*float*) – Extract length in seconds, if None will look in templatedefaults file.
- **debug** (*int*) – Debug level, higher number=more output.

### 3.3.3 match\_filter

Function to cross-correlate templates generated by template\_gen function with data and output the detections. The main component of this script is the `normxcorr2` function from the openCV image processing package. This is a highly optimized and accurate normalized cross-correlation routine. The details of this code can be found here: \* [http://www.cs.ubc.ca/research/deaton/remarks\\_ncc.html](http://www.cs.ubc.ca/research/deaton/remarks_ncc.html) The cpp code was first tested using the Matlab mex wrapper, and has since been ported to a python callable dynamic library.

Part of the EQcorrscan module to integrate seisan nordic files into a full cross-channel correlation for detection routine. EQcorrscan is a python module designed to run match filter routines for seismology, within it are routines for integration to seisan and obspy. With obspy integration (which is necessary) all main waveform formats can be read in and output.

This main section contains a script, `LFE_search.py` which demonstrates the usage of the built in functions from template generation from picked waveform through detection by match filter of continuous data to the generation of lag times to be used for relative locations.

The match-filter routine described here was used a previous Matlab code for the Chamberlain et al. 2014 G-cubed publication. The basis for the lag-time generation section is outlined in Hardebeck & Shelly 2011, GRL.

Code generated by Calum John Chamberlain of Victoria University of Wellington, 2015.

## Note

### Pre-requisites:

- gcc - for the installation of the openCV correlation routine
- python-cv2 - Python bindings for the openCV routines
- python-joblib - used for parallel processing
- **python-obspy - used for lots of common seismological processing**
  - **requires:**
    - \* numpy
    - \* scipy
    - \* matplotlib
- NonLinLoc - used outside of all codes for travel-time generation

Copyright 2015 Calum Chamberlain

This file is part of EQcorrscan.

EQcorrscan is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

EQcorrscan is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with EQcorrscan. If not, see <http://www.gnu.org/licenses/>.

**class** `match_filter.DETECTION` (*template\_name, detect\_time, no\_chans, detect\_val, threshold, type\_ofdet, chans=None*)

Information required for a full detection based on cross-channel correlation sums.

#### Attributes:

**type** `template_name` str

**param** `template_name` The name of the template for which this

detection was made. :type `detect_time`: :class: 'obspy.UTCDateTime' :param `detect_time`: Time of detection as an obspy UTCDateTime object :type `no_chans`: int :param `no_chans`: The number of channels for which the cross-channel correlation sum was calculated over. :type `chans`: list of str :param `chans`: List of stations for the detection :type `ccsum_val`: float :param `ccsum_val`: The raw value of the cross-channel correlation sum for this detection. :type `threshold`: float :param `threshold`: The value of the threshold used for this detection, will be the raw threshold value related to the ccsum. :type `typeofdet`: str :param `typeofdet`: Type of detection, STA, corr, bright

`match_filter._channel_loop` (*templates, stream, cores=1, debug=0*)

Loop to generate cross channel correlation sums for a series of templates hands off the actual correlations to a sister function which can be run in parallel.

**Parameters** `templates` – A list of templates, where each one should be an

obspy.Stream object containing multiple traces of seismic data and the relevant header information. :param `stream`: A single obspy.Stream object containing daylong seismic data to be correlated through using the templates. This is in effect the image. :type `core`: int :param `core`: Number of cores to loop over :type `debug`: int :param `debug`: Debug level.

**Returns** New list of :class: 'numpy.array' objects. These will contain



the correlation sums for each template for this day of data. :return: list of ints as number of channels used for each cross-correlation

`match_filter._template_loop(template, chan, station, channel, debug=0, i=0)`

Sister loop to handle the correlation of a single template (of multiple channels) with a single channel of data.

**Parameters** `i (int)` – Optional argument, used to keep track of which process is being

run.

**Returns** tuple of (i,ccc) with ccc as an ndarray

#### Note

..This function currently assumes only one template-channel per data-channel, while this is normal for a standard matched-filter routine, if we wanted to implement a subspace detector, this would be the function to change, I think. E.g. where I currently take only the first matching channel, we could loop through all the matching channels and then sum the correlation sums - however I don't really understand how you detect based on that. More reading of the Harris document required.

`match_filter.match_filter(template_names, template_list, st, threshold, threshold_type, trig_int, plotvar, plotdir='.', cores=1, tempdir=False, debug=0, plot_format='jpg')`

Over-arching code to run the correlations of given templates with a day of seismic data and output the detections based on a given threshold.

#### Parameters

- **template\_names (list)** – List of template names in the same order as `template_list`
- **template\_list (list :class: 'obspy.Stream')** – A list of templates of which each template is a `Stream` of `obspy` traces containing seismic data and header information.
- **st** – An `obspy.Stream` object containing all the data available and required for the correlations with templates given. For efficiency this should contain no excess traces which are not in one or more of the templates. This will now remove excess traces internally, but will copy the stream and work on the copy, leaving your input stream untouched.
- **threshold (float)** – A threshold value set based on the `threshold_type`
- **threshold\_type (str)** – The type of threshold to be used, can be `MAD`, `absolute` or `av_chan_corr`. `MAD` threshold is calculated as `threshold*(median(abs(cccsum)))` where `cccsum` is the cross-correlation sum for a given template. `absolute` threshold is a true absolute threshold based on the `cccsum` value `av_chan_corr` is based on the mean-values of single-channel cross-correlations assuming all data are present as required for the template, e.g. `av_chan_corr_thresh=threshold*(cccsum/len(template))` where `template` is a single template from the input and the length is the number of channels within this template.
- **trig\_int (float)** – Minimum gap between detections in seconds.
- **plotvar (bool)** – Turn plotting on or off
- **plotdir (str)** – Path to plotting folder, plots will be output here, defaults to run location.
- **tempdir (String or False)** – Directory to put temporary files, or `False`
- **cores (int)** – Number of cores to use
- **debug (int)** – Debug output level, the bigger the number, the more the output.

#### Returns

class 'DETECTIONS' detections for each channel formatted as

Class 'obspy.UTCDatetime' objects.



**Note** Plotting within the match-filter routine uses the Agg backend with interactive plotting turned off. This is because the function is designed to work in bulk. If you wish to turn interactive plotting on you must import matplotlib in your script first, when you then import match\_filter you will get the warning that this call to matplotlib has no effect, which will mean that match\_filter has not changed the plotting behaviour.

`match_filter.normxcorr2(template, image)`

Base function to call the c++ correlation routine from the openCV image processing suite. Requires you to have installed the openCV python bindings, which can be downloaded on Linux machines using: **sudo apt-get install python-openCV**.

Here we use the `cv2.TM_CCOEFF_NORMED` method within openCV to give the normalized cross-correlation. Documentation on this function can be found here:

[http://docs.opencv.org/modules/imgproc/doc/object\\_detection.html?highlight=matchtemplate#cv2.matchTemplate](http://docs.opencv.org/modules/imgproc/doc/object_detection.html?highlight=matchtemplate#cv2.matchTemplate)

#### Parameters

- **template** – Template array
- **image** – image to scan

the template through. The order of these matters, if you put the template after the image you will get a reversed correlation matrix

**Returns** New :class: 'numpy.array' object of the correlation values for the correlation of the image with the template.

### 3.3.4 lag\_calc

Functions to generate lag-times for events detected by correlation.

Part of the EQcorrscan module to integrate seisan nordic files into a full cross-channel correlation for detection routine. EQcorrscan is a python module designed to run match filter routines for seismology, within it are routines for integration to seisan and obspy. With obspy integration (which is necessary) all main waveform formats can be read in and output.

This main section contains a script, `LFE_search.py` which demonstrates the usage of the built in functions from template generation from picked waveforms through detection by match filter of continuous data to the generation of lag times to be used for relative locations.

The match-filter routine described here was used a previous Matlab code for the Chamberlain et al. 2014 G-cubed publication. The basis for the lag-time generation section is outlined in Hardebeck & Shelly 2011, GRL.

Code generated by Calum John Chamberlain of Victoria University of Wellington, 2015.

#### Note

#### Pre-requisites:

- gcc - for the installation of the openCV correlation routine
- python-cv2 - Python bindings for the openCV routines
- python-joblib - used for parallel processing
- **python-obsy - used for lots of common seismological processing**
  - requires:
    - \* numpy
    - \* scipy
    - \* matplotlib

- NonLinLoc - used outside of all codes for travel-time generation

Copyright 2015 Calum Chamberlain

This file is part of EQcorrscan.

EQcorrscan is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

EQcorrscan is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with EQcorrscan. If not, see <http://www.gnu.org/licenses/>.

`lag_calc._channel_loop` (*detection, template, i=0*)

Utility function to take a stream of data for the detected event and parse the correct data to lag\_gen

#### Parameters

- **detection** (*obspy.Stream*) – Stream of data for the slave event detected using template
- **template** (*obspy.Stream*) – Stream of data as the template for the detection.
- **i** (*int, optional*) – Used to track which process has occurred when running in parallel

**Returns** picks, a tuple of (lag in s, cross-correlation value, station, chan)

`lag_calc.day_loop` (*detection\_streams, template*)

Function to loop through multiple detections for one template - ostensibly designed to run for the same day of data for I/O simplicity, but as you are passing stream objects it could run for all the detections ever, as long as you have the RAM!

#### Parameters

- **detection\_streams** (*List of obspy.Stream*) – List of all the detections for this template that you want to compute the optimum pick for.
- **template** (*obspy.Stream*) – The original template used to detect the detections passed

**Returns** lags - List of List of tuple: lags[i] corresponds to detection[i], lags[i][j] corresponds to a channel of detection[i], within this tuple is the lag (in seconds), normalised correlation, station and channel.

`lag_calc.lag_calc` (*detections, detect\_data, templates, shift\_len=0.2, min\_cc=0.4*)

Overseer function to take a list of detection objects, cut the data for them to lengths of the same length of the template + shift\_len on either side. This will then write out SEISAN s-file for the detections with pick times based on the lag-times found at the maximum correlation, providing that correlation is above the min\_cc.

#### Parameters

- **detections** (*List of DETECTION*) – List of DETECTION objects
- **detect\_data** (*obspy.Stream*) – All the data needed to cut from - can be a gappy Stream
- **templates** (*List of tuple of String, obspy.Stream*) – List of the templates used as tuples of template name, template
- **shift\_len** (*float*) – Shift length allowed for the pick in seconds, will be plus/minus this amount - default=0.2
- **min\_cc** (*float*) – Minimum cross-correlation value to be considered a pick, default=0.4

## 3.4 Utils

Utility functions for integration with other software (currently only seisan), and for the analysis of waveforms detected by cross-correlation.

### 3.4.1 Sfile\_util

Part of the EQcorrscan module to read nordic format s-files and write them EQcorrscan is a python module designed to run match filter routines for seismology, within it are routines for integration to seisan and obspy. With obspy integration (which is necessary) all main waveform formats can be read in and output.

Code generated by Calum John Chamberlain of Victoria University of Wellington, 2015.

Copyright 2015 Calum Chamberlain

This file is part of EQcorrscan.

EQcorrscan is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

EQcorrscan is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with EQcorrscan. If not, see <http://www.gnu.org/licenses/>.

```
class Sfile_util.EVENTINFO (time=<mock.Mock object at 0x321da50>, loc_mod_ind=' ',
                             dist_ind=' ', ev_id=' ', latitude=nan, longitude=nan, depth=nan,
                             depth_ind=' ', loc_ind=' ', agency=' ', nsta=0, t_RMS=nan,
                             Mag_1=nan, Mag_1_type=' ', Mag_1_agency=' ', Mag_2=nan,
                             Mag_2_type=' ', Mag_2_agency=' ', Mag_3=nan, Mag_3_type=' ',
                             Mag_3_agency=' ')
```

Header information for seisan events, again all fields can be left blank for a default empty header. The print function for header will print important information, but not as seen in an S-file.

For more information on parameters see the seisan manual.

#### Attributes:

**type time** obspy.UTCDatetime

**param time** Event origin time

**type loc\_mod\_ind** str

**param loc\_mod\_ind**

**type dist\_ind** str

**param dist\_ind** Distance flag, usually 'L' for local, 'R' for regional and 'D' for distant

**type ev\_id** str

**param ev\_id** Often blank, 'E' denotes explosion and fixes depth to 0km

**type latitude** float

**param latitude** Hypocentre latitude in decimal degrees

**type longitude** float

**param lognitude** Hypocentre longitude in decimal degrees

**type depth** float

**param depth** hypocentre depth in km

**type** **depth\_ind** str  
**param** **depth\_ind**  
**type** **loc\_ind** str  
**param** **loc\_ind**  
**type** **agency** str  
**param** **agency** Reporting agency, three letters  
**type** **nsta** int  
**param** **nsta** Number of stations recording  
**type** **t\_RMS** float  
**param** **t\_RMS** Root-mean-squared time residual  
**type** **Mag\_1** float  
**param** **Mag\_1** first magnitude  
**type** **Mag\_1\_type** str  
**param** **Mag\_1\_type** Type of magnitude for Mag\_1 ('L', 'C', 'W')  
**type** **Mag\_1\_agency** str  
**param** **Mag\_1\_agency** Reporting agency for Mag\_1  
**type** **Mag\_2** float  
**param** **Mag\_2** second magnitude  
**type** **Mag\_2\_type** str  
**param** **Mag\_2\_type** Type of magnitude for Mag\_2 ('L', 'C', 'W')  
**type** **Mag\_2\_agency** str  
**param** **Mag\_2\_agency** Reporting agency for Mag\_2  
**type** **Mag\_3** float  
**param** **Mag\_3** third magnitude  
**type** **Mag\_3\_type** str  
**param** **Mag\_3\_type** Type of magnitude for Mag\_3 ('L', 'C', 'W')  
**type** **Mag\_3\_agency** str  
**param** **Mag\_3\_agency** Reporting agency for Mag\_3

**class** **Sfile\_util.PICK**(*station=' '*, *channel=' '*, *impulsivity=' '*, *phase=' '*, *weight=999*, *polarity=' '*, *time=<mock.Mock object at 0x321da50>*, *coda=999*, *amplitude=nan*, *peri=nan*, *azimuth=nan*, *velocity=nan*, *AIN=999*, *SNR=nan*, *azimuthres=999*, *timeres=nan*, *finalweight=999*, *distance=nan*, *CAZ=999*)

Pick information for seisan implimentation, note all fields can be left blank to obtain a default pick: picks have a print function which will print them as they would be seen in an S-file.

**Attributes:**

**type** **station** str  
**param** **station** Station name, less than five charectars required as standard  
**type** **channel** str  
**param** **channel** Two or three charactar channel name, stored as two charactars in S-file  
**type** **impulsivity** str  
**param** **impulsivity** either 'C' or 'D' for compressive and dilatational

**type phase** str  
**param phase** Any allowable phase name in two characters  
**type weight** int  
**param weight** 0-4 with 0=100%, 4=0%, use weight=9 for unknown timing  
**type polarity** str  
**type time** obspy.UTCDateTime()  
**param time** Pick time as an obspy.UTCDateTime object  
**type coda** int  
**param coda** Length of coda in seconds  
**type amplitude** float  
**param amplitude** Amplitude (zero-peak), type is given in phase  
**type peri** float  
**param peri** Period of amplitude  
**type azimuth** float  
**param azimuth** Direction of approach in degrees  
**type velocity** float  
**param velocity** Phase velocity (km/s)  
**type AIN** int  
**param AIN** Angle of incidence.  
**type SNR** float  
**param SNR** Signal to noise ratio  
**type azimuthres** int  
**param azimuthres** Residual azimuth  
**type timeres** float  
**param timeres** Time residual in seconds  
**type finalweight** int  
**param finalweight** Final weight used in location  
**type distance** float  
**param distance** Source-reciever distance in km  
**type CAZ** int  
**param CAZ** Azimuth at source.

**write** (*filename*)

Public function to write the pick to a file

**Parameters** **filename** (*str*) – Path to file to write to - will append to file

`Sfile_util._float_conv` (*string*)

Convenience tool to convert from string to float, if empty string return NaN rather than an error

`Sfile_util._int_conv` (*string*)

Convenience tool to convert from string to integer, if empty string return a 999 rather than an error

`Sfile_util._str_conv` (*number, rounded=False*)

Convenience tool to convert a number, either float or into into a string, if the int is 999, or the float is NaN, returns empty string.

`Sfile_util.blanksfile` (*wavefile, evtype, userID, outdir, overwrite=False, evtime=False*)

Module to generate an empty s-file with a populated header for a given waveform.

#### Parameters

- **wavefile** (*String*) – Wavefile to associate with this S-file, the timing of the S-file will be taken from this file if evtime is not set
- **evtype** (*String*) – L,R,D
- **userID** (*String*) – 4-character SEISAN USER ID
- **outdir** (*String*) – Location to write S-file
- **overwrite** (*Bool*) – Overwrite an existing S-file, default=False
- **evtime** (*UTCDateTime*) – If given this will set the timing of the S-file

**Returns** String, S-file name

`Sfile_util.populateSfile` (*sfilename, picks*)

Module to populate a blank nordic format S-file with pick information, arguments required are the filename of the blank s-file and the picks where picks is a dictionary of picks including station, channel, impulsivity, phase, weight, polarity, time, coda, amplitude, peri, azimuth, velocity, SNR, azimuth residual, Time-residual, final weight, epicentral distance & azimuth from event to station.

This is a full pick line information from the seisan manual, P. 341

#### Parameters

- **sfilename** (*str*) – Path to S-file to populate, must have a header already
- **picks** (*List of :class: PICK*) – List of the picks to be written out

`Sfile_util.readheader` (*sfilename*)

Function to read the header information from a seisan nordic format S-file.

**Parameters** **sfilename** (*str*) – Path to the s-file

#### Returns

`class EVENTINFO`

`Sfile_util.readpicks` (*sfilename*)

Function to read pick information from the s-file

**Parameters** **sfilename** (*String*) – Path to sfile

**Returns** List of :class: PICK

`Sfile_util.readwavename` (*sfilename*)

Convenience function to extract the waveform filename from the s-file, returns a list of waveform names found in the s-file as multiples can be present.

**Parameters** **sfilename** (*str*) – Path to the sfile

**Returns** List of str

`Sfile_util.test_rw()`

Function to test the functions herein.

### 3.4.2 findpeaks

Function to find peaks in data above a certain threshold as part of the EQcorr package written by Calum Chamberlain of Victoria University of Wellington in early 2015.

Copyright 2015 Calum Chamberlain

This file is part of EQcorrscan.

EQcorrscan is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

EQcorrscan is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with EQcorrscan. If not, see <http://www.gnu.org/licenses/>.

`findpeaks.find_peaks2(arr, thresh, trig_int, debug=0, maxwidth=10, starttime=<mock.Mock object at 0x4091810>, samp_rate=1.0)`

Function to determine peaks in an array of data using scipy `find_peaks_cwt`, works fast in certain cases, but for `match_filter` cccsum peak finding, `find_peaks2_short` works better. Test it out and see which works best for your application.

#### Parameters

- **arr** (*ndarray*) – 1-D numpy array is required
- **thresh** (*float*) – The threshold below which will be considered noise and peaks will not be found in.
- **trig\_int** (*int*) – The minimum difference in samples between triggers, if multiple peaks within this window this code will find the highest.
- **debug** (*int*) – Optional, debug level 0-5
- **maxwidth** (*int*) – Maximum peak width to look for in samples

**Returns** peaks, locs: Lists of peak values and locations.

`findpeaks.find_peaks2_short(arr, thresh, trig_int, debug=0, starttime=<mock.Mock object at 0x4091810>, samp_rate=1.0)`

Function to determine peaks in an array of data above a certain threshold. Uses a mask to remove data below threshold and finds peaks in what is left.

#### Parameters

- **arr** (*ndarray*) – 1-D numpy array is required
- **thresh** (*float*) – The threshold below which will be considered noise and peaks will not be found in.
- **trig\_int** (*int*) – The minimum difference in samples between triggers, if multiple peaks within this window this code will find the highest.
- **debug** (*int*) – Optional, debug level 0-5

**Returns** peaks, locs: Lists of peak values and locations.

`findpeaks.find_peaks_dep(arr, thresh, trig_int, debug=0, starttime=<mock.Mock object at 0x4091810>, samp_rate=1.0)`

Function to determine peaks in an array of data above a certain threshold.

Deprecated peak-finding routine, very slow, but accurate. If all else fails this one should work.

#### Parameters

- **arr** (*ndarray*) – 1-D numpy array is required
- **thresh** (*float*) – The threshold below which will be considered noise and peaks will not be found in.
- **trig\_int** (*int*) – The minimum difference in samples between triggers, if multiple peaks within this window this code will find the highest.

**Returns** peaks, locs: Lists of peak values and locations.

`findpeaks.is_prime(number)`

**Function to test primality of a number. Function lifted from online resource:**

<http://www.codeproject.com/Articles/691200/Primality-test-algorithms-Prime-test-The-fastest-w>

**This function is distributed under a separate licence:** This article, along with any associated source code and files, is licensed under The Code Project Open License (CPOL)

**Parameters** `number` (*int*) – Integer to test for primality

**Returns** `bool`

### 3.4.3 clustering

Functions to cluster seismograms by a range of constraints.

Copyright 2015 Calum Chamberlain

This file is part of EQcorrscan.

EQcorrscan is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

EQcorrscan is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with EQcorrscan. If not, see <<http://www.gnu.org/licenses/>>.

`clustering.SVD` (*stream\_list*)

Function to compute the SVD of a number of templates and return the singular vectors and singular values of the templates.

**Parameters** `stream_list` (*List of Obspy.Stream*) – List of the templates to be analysed

**Returns** `SVector`(list of ndarray), `SValues`(list) for each channel, `Uvalues`(list of ndarray) for each channel, `stachans`, List of String (station.channel)

#### Note

It is recommended that you align the data before computing the SVD, e.g., the P-arrival on all templates for the same channel should appear at the same time in the trace.

`clustering.SVD_2_stream` (*SVectors, stachans, k, sampling\_rate*)

Function to convert the singular vectors output by SVD to streams, one for each singular vector level, for all channels.

#### Parameters

- **SVectors** (*List of np.ndarray*) – Singular vectors
- **stachans** (*List of Strings*) – List of station.channel Strings
- **k** (*int*) – Number of streams to return = number of SV's to include
- **sampling\_rate** (*float*) – Sampling rate in Hz

**Returns** `SVstreams`, List of `Obspy.Stream`, with `SVStreams[0]` being composed of the highest rank singular vectors.

`clustering.cluster` (*stream\_list, show=True, corr\_thresh=0.3, save\_corrmat=False, cores='all', debug=1*)

Function to take a set of templates and cluster them, will return groups as lists of streams. Clustering is done by computing the cross-channel correlation sum of each stream in `stream_list` with every other stream in the list. `Scipy.cluster.hierarchy` functions are then used to compute the complete distance matrix, where



distance is 1 minus the normalised cross-correlation sum such that larger distances are less similar events. Groups are then created by clustering the distance matrix at distances less than 1 - corr\_thresh.

Will compute the distance matrix in parallel, using all available cores

#### Parameters

- **stream\_list** (*List of Obspy.Stream*) – List of templates to compute clustering for
- **show** (*bool*) – plot linkage on screen if True, defaults to True
- **corr\_thresh** (*float*) – Cross-channel correlation threshold for grouping
- **save\_corrmat** (*bool*) – If True will save the distance matrix to dist\_mat.npy
- **cores** (*int*) – numebr of cores to use when computing the distance matrix, defaults to 'all' which will work out how many cpus are available and hog them.
- **debug** (*int*) – Level of debugging from 1-5, higher is more output, currently only level 1 implimented.

**Returns** List of groups with each group a list of streams making up that group.

`clustering.corr_cluster` (*trace\_list, thresh=0.9*)

Group traces based on correlations above threshold with the stack - will run twice, once with a lower threshold, then again with your threshold to remove large outliers

#### Parameters

- **trace\_list** (*List of :class:obsipy.Trace*) – Traces to compute similarity between
- **thrsh** – Correlation threshold between -1-1

**Returns** np.ndarray of bool

`clustering.cross_chan_coherence` (*st1, st2, i=0*)

Function to determine the cross-channel coherancy between two streams of multichannel seismic data.

#### Parameters

- **st1** (*obsipy Stream*) – Stream one
- **st2** (*obsipy Stream*) – Stream two
- **i** (*int*) – index used for parallel async processing, returned unaltered

**Returns** cross channel coherence, float - normalized by number of channels, if i, returns tuple of (cccoh, i) where i is int, as input.

`clustering.distance_matrix` (*stream\_list, cores=1*)

Function to compute the distance matrix for all templates - will give distance as 1-abs(cccoh), e.g. a well correlated pair of templates will have small distances, and an equally well correlated reverse image will have the same distance as apositively correlated image - this is an issue

#### Parameters

- **tream\_list** – List of the streams to compute the distance matrix for
- **cores** – Number of cores to parallel process using, defaults to 1.

**Returns** ndarray - distance matrix

`clustering.empirical_SVD` (*stream\_list, linear=True*)

Empirical subspace detector generation function. Takes a list of templates and computes the stack as the first order subspace detector, and the differential of this as the second order subspace detector following the empirical subspace method of Barrett & Beroza, 2014 - SRL.

#### Parameters

- **stream\_list** (*list of stream*) – list of template streams to compute the subspace detectors from

- **linear** (*Bool*) – Set to true by default to compute the linear stack as the first subspace vector, False will use the phase-weighted stack as the first subspace vector.

**Returns** list of two streams

`clustering.extract_detections(detections, templates, contbase_list, extract_len=90.0, outdir=None, extract_Z=True, additional_stations=[])`

Function to extract the waveforms associated with each detection in a list of detections for the template, template. Waveforms will be returned as a list of `obspy.Streams` containing segments of `extract_len`. They will also be saved if `outdir` is set. The default is unset. The default `extract_len` is 90 seconds per channel.

#### Parameters

- **detections** (*List tuple of of :class: datetime.datetime, string*) – List of datetime objects, and their associated template name
- **templates** (*List of tuple of string and :class: obspy.Stream*) – A list of the tuples of the template name and the template Stream used to detect detections.
- **contbase\_list** (*List of tuple of string*) – List of tuples of the form ['path', 'type', 'network'] Where path is the path to the continuous database, type is the directory structure, which can be either Yyyy/Rjjj.01, which is the standard IRIS Year, julian day structure, or, yyymmdd which is a single directory for every day.
- **extract\_len** (*float*) – Length to extract around the detection (will be equally cut around the detection time) in seconds. Default is 90.0.
- **outdir** (*Bool or String*) – Default is None, with None set, no files will be saved, if set each detection will be saved into this directory with files named according to the detection time, NOT than the waveform start time. Detections will be saved into template subdirectories.
- **extract\_Z** (*Bool*) – Set to True to also extract Z channels for detections delays will be the same as horizontal channels, only applies if only horizontal channels were used in the template.
- **additional\_stations** (*List of tuple*) – List of stations, channels and networks to also extract data for using an average delay.

**Returns** List of :class: obspy.Stream

`clustering.group_delays(stream_list)`

Function to group template waveforms according to their delays

**Parameters** **stream\_list** (*List of obspy.Stream*) – List of the waveforms you want to group

**Returns** List of List of obspy.Streams where each initial list is a group with the same delays

`clustering.re_thresh_csv(path, old_thresh, new_thresh, chan_thresh)`

Function to remove detections by changing the threshold, can only be done to remove detection by increasing threshold, threshold lowering will have no affect.

#### Parameters

- **path** (*Str*) – Path to the .csv detection file
- **old\_thresh** (*float*) – Old threshold MAD multiplier
- **new\_thresh** (*float*) – New threshold MAD multiplier
- **chan\_thresh** (*int*) – Minimum number of channels for a detection

returns: List of detections

`clustering.space_time_cluster(detections, t_thresh, d_thresh)`

Function to cluster detections in space and time, use to separate repeaters from other events

#### Parameters

- **detections** (*List*) – List of tuple of tuple of location (lat, lon, depth (km)), and time as a datetime object
- **t\_thresh** (*float*) – Maximum inter-event time threshold in seconds
- **d\_thresh** (*float*) – Maximum inter-event distance in km

**Returns** List of tuple (detections, clustered) and list of indices of clustered detections

### 3.4.4 pre\_processing

Utilities module for the EQcorrscan package written by Calum Chamberlain of Victoria University Wleington. These functions are designed to do the basic processing of the data using obspy modules (which also rely on scipy and numpy).

Copyright 2015 Calum Chamberlain

This file is part of EQcorrscan.

EQcorrscan is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

EQcorrscan is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with EQcorrscan. If not, see <<http://www.gnu.org/licenses/>>.

`pre_processing._check_daylong` (*tr*)

Function to check the data quality of the daylong file - check to see that the day isn't just zeros, with large steps, if it is then the resampling will hate it.

**Parameters** *tr* (*obspy.Trace*) – Trace to check if the data are daylong.

**Return** *qual* bool

`pre_processing.dayproc` (*tr, lowcut, highcut, filt\_order, samp\_rate, debug, starttime*)

Basic function to bandpass, downsample and check headers and length of trace to ensure files start at the start of a day and are daylong. Works in place on data. This is employed to ensure all parts of the data are processed in the same way.

**Parameters**

- **tr** (*obspy.Trace*) – Trace to process
- **highcut** (*float*) – High cut in Hz for bandpass
- **filt\_order** (*int*) – Corners for bandpass
- **samp\_rate** (*float*) – Desired sampling rate in Hz
- **debug** (*int*) – Debug output level from 0-5, higher numbers = more output
- **starttime** (*obspy.UTCDateTime*) – Desired start of trace

**Returns** *obspy.Stream*

**..rubric:: Note** Will convert channel names to two characters long

`pre_processing.shortproc` (*st, lowcut, highcut, filt\_order, samp\_rate, debug=0*)

Basic function to bandpass, downsample. Works in place on data. This is employed to ensure all parts of the data are processed in the same way.

**Parameters**

- **st** (*obspy.Stream*) – Stream to process

- **highcut** (*float*) – High cut for bandpass in Hz
- **lowcut** (*float*) – Low cut for bandpass in Hz
- **filt\_order** (*int*) – Number of corners for bandpass filter
- **samp\_rate** (*float*) – Sampling rate desired in Hz
- **debug** (*int*) – Debug flag from 0-5, higher numbers = more output

**Returns** obspy.Stream

**..rubric:: Note** Will convert channel names to two characters long

### 3.4.5 EQcorrscan\_plotting

Utility code for most of the plots used as part of the EQcorrscan package.

Copyright 2015 Calum Chamberlain

This file is part of EQcorrscan.

EQcorrscan is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

EQcorrscan is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with EQcorrscan. If not, see <<http://www.gnu.org/licenses/>>.

`EQcorrscan_plotting.NR_plot` (*stream*, *NR\_stream*, *detections*, *false\_detections=False*, *size=(18.5, 10)*, *save=False*, *title=False*)

Function to plot the Network response alongside the streams used - highlights detection times in the network response

#### Parameters

- **stream** – Stream to plot
- **NR\_stream** – Stream for the network response
- **detections** (*List of datetime objects*) – List of the detections
- **false\_detections** (*List of datetime*) – Either False (default) or list of false detection times
- **size** (*tuple*) – Size of figure, default is (18.5,10)
- **save** (*bool*) – Save figure or plot to screen, if not False, must be string of save path
- **title** (*str*) – String for the title of the plot, set to False

`EQcorrscan_plotting.Noise_plotting` (*station*, *channel*, *PAZ*, *datasource*)

Function to make use of obspy's PPSD functionality to read in data from a single station and the poles-and-zeros for that station before plotting the PPSD for this station. See McNamara(2004) for more details.

#### Parameters

- **station** (*String*) – Station name as it is in the filenames in the database
- **channel** (*String*) – Channel name as it is in the filenames in the database
- **PAZ** (*Dict*) – Must contain, Poles, Zeros, Sensitivity, Gain :type Poles: List of Complex :type Zeros: List of Complex :type Sensitivity: Float :type Gain: Float
- **datasource** (*String*) – The directory in which data can be found, can contain wild-cards.

**Returns** PPSD object

`EQcorrscan_plotting.SVD_plot (SVStreams, SValues, stachans, title=False)`

Function to plot the singular vectors from the clustering routines, one plot for each stachan

#### Parameters

- **SVStreams** (*List of :class:Obspy.Stream*) – See clustering.SVD\_2\_Stream - will assume these are ordered by power, e.g. first singular vector in the first stream
- **SValues** (*List of float*) – List of the singular values corresponding to the SVStreams
- **stachans** (*List*) – List of station.channel

`EQcorrscan_plotting.chunk_data (tr, samp_rate, state='mean')`

Function to downsample data for plotting by computing the maximum of data within chunks, useful for plotting waveforms or cccsums, large datasets that could otherwise exceed the complexity allowed, and overflow.

#### Parameters

- **tr** – Trace to be chunked
- **samp\_rate** (*float*) – Desired sampling rate in Hz
- **state** (*str*) – Either 'Min', 'Max', 'Mean' or 'Maxabs' to return one of these for the chunks. Maxabs will return the largest (positive or negative) for that chunk.

#### Returns

`class obspy.Trace`

`EQcorrscan_plotting.cumulative_detections (dates, template_names, save=False, save_file='')`

Simple plotting function to take a list of datetime objects and plot a cumulative detections list. Can take dates as a list of lists and will plot each list separately, e.g. if you have dates from more than one template it will overlay them in different colours.

#### Parameters

- **dates** (*list of lists of datetime.datetime*) – Must be a list of lists of datetime.datetime objects
- **template\_names** (*list of strings*) – List of the template names in order of the dates
- **save** (*Boolean, optional*) – Save figure or show to screen
- **savefile** (*String, optional*) – String to save to.

`EQcorrscan_plotting.detection_multiplot (stream, template, times, streamcolour='k', templatecolour='r')`

Function to plot the stream of data that has been detected in, with the template on top of it timed according to a list of given times, just a pretty way to show a detection!

#### Parameters

- **stream** (*obsipy.Stream*) – Stream of data to be plotted as the base (black)
- **template** (*obsipy.Stream*) – Template to be plotted on top of the base stream (red)
- **times** (*List of datetime.datetime*) – list of times of detections in the order of the channels in template.
- **streamcolour** (*str*) – String of matplotlib colour types for the stream
- **templatecolour** (*str*) – Colour to plot the template in.

`EQcorrscan_plotting.freq_mag (magnitudes, completeness, max_mag, binsize=0.2)`

Function to make a frequency-magnitude histogram and cumulative density plot. This can compute a b-value, but not a completeness at the moment.

#### Parameters

- **magnitude** – list of float of magnitudes
- **completeness** (*float*) – Level to compute the b-value above
- **max\_mag** (*float*) – Maximum magnitude to try and fit a b-value to
- **binsize** (*float*) – Width of histogram bins, defaults to 0.2

EQcorrscan\_plotting.**interev\_mag** (*times, mags*)

Function to plot interevent times against magnitude for given times and magnitudes.

#### Parameters

- **times** (*list of datetime*) – list of the detection times, must be sorted the same as mags
- **mags** (*list of float*) – list of magnitudes

EQcorrscan\_plotting.**interev\_mag\_sfiles** (*sfiles*)

Function to plot interevent-time versus magnitude for series of events. Wrapper for interev\_mag.

**Parameters** **sfiles** (*List*) – List of sfiles to read from

EQcorrscan\_plotting.**multi\_event\_singlechan** (*streams, picks, clip=10.0, pre\_pick=2.0, freqmin=False, freqmax=False, realign=False, cut=(-3.0, 5.0), PWS=False, title=False*)

Function to plot data from a single channel at a single station for multiple events - data will be aligned by their pick-time given in the picks

#### Parameters

- **streams** (*List of :class:obspy.stream*) – List of the streams to use, can contain more traces than you plan on plotting
- **picks** (*List of :class:PICK*) – List of picks, one for each stream
- **clip** (*float*) – Length in seconds to plot, defaults to 10.0
- **pre\_pick** (*Float*) – Length in seconds to extract and plot before the pick, defaults to 2.0
- **freqmin** (*float*) – Low cut for bandpass in Hz
- **freqmax** (*float*) – High cut for bandpass in Hz
- **realign** (*Bool*) – To compute best alignment based on correlation or not.
- **cut** (*tuple:*) – tuple of start and end times for cut in seconds from the pick
- **PWS** (*bool*) – compute Phase Weighted Stack, if False, will compute linear stack
- **title** (*str*) – Plot title.

**Returns** Aligned and cut traces, and new picks

EQcorrscan\_plotting.**peaks\_plot** (*data, starttime, samp\_rate, save=False, peaks=[(0, 0)], savefile=''*)

Simple utility code to plot the correlation peaks to check that the peak finding routine is running correctly, used in debugging for the EQcorrscan module.

#### Parameters

- **data** (*numpy.array*) – Numpy array of the data within which peaks have been found
- **starttime** (*obspy.UTCDateTime*) – Start time for the data
- **samp\_rate** (*float*) – Sampling rate of data in Hz
- **save** (*Boolean, optional*) – Save figure or plot to screen (False)
- **peaks** (*List of Tuple, optional*) – List of peak locations and amplitudes (loc, amp)
- **savefile** (*String, optional*) – Path to save to, only used if save=True

`EQcorrscan_plotting.plot_synth_real` (*real\_template*, *synthetic*, *channels=False*)

Plot multiple channels of data for real data and synthetic

#### Parameters

- **real\_template** (*obspy.Stream*) – Stream of the real template
- **synthetic** (*obspy.Stream*) – Stream of synthetic template
- **channels** (*List of str*) – List of tuples of (station, channel) to plot, default is False, which plots all.

`EQcorrscan_plotting.pretty_template_plot` (*template*, *size=(18.5, 10.5)*, *save=False*, *title=False*, *background=False*)

Function to make a pretty plot of a single template, designed to work better than the default obspy plotting routine for short data lengths.

#### Parameters

- **template** – Template stream to plot
- **size** (*tuple*) – tuple of plot size
- **save** (*Boolean*) – if False will plot to screen, if True will save
- **title** (*Boolean*) – String if set will be the plot title
- **background** – Stream to plot the template within.

`EQcorrscan_plotting.threeD_gridplot` (*nodes*, *save=False*, *savefile=''*)

Function to plot in 3D a series of grid points.

#### Parameters

- **nodes** (*List of tuples*) – List of tuples of the form (lat, long, depth)
- **save** (*bool*) – if True will save without plotting to screen, if False (default) will plot to screen but not save
- **savefile** (*str*) – required if save=True, path to save figure to.

`EQcorrscan_plotting.threeD_seismpoint` (*stations*, *nodes*)

Function to plot seismicity and stations in a 3D, movable, zoomable space using matplotlib's Axes3D package.

#### Parameters

- **stations** (*list of tuple*) – list of one tuple per station of (lat, long, elevation), with up positive
- **nodes** (*list of tuple*) – list of one tuple per event of (lat, long, depth) with down positive

`EQcorrscan_plotting.triple_plot` (*cccsun*, *cccsun\_hist*, *trace*, *threshold*, *save=False*, *savefile=''*)

Main function to make a triple plot with a day-long seismogram, day-long correlation sum trace and histogram of the correlation sum to show normality.

#### Parameters

- **cccsun** (*numpy.ndarray*) – Array of the cross-channel cross-correlation sum
- **cccsun\_hist** – ccsum for histogram plotting, can be the same as ccsum but included if ccsum is just an envelope.
- **trace** (*obspy.Trace*) – A sample trace from the same time as ccsum
- **threshold** (*float*) – Detection threshold within ccsum
- **save** (*Bool, optional*) – If True will save and not plot to screen, vice-versa if False
- **savefile** (*String, optional*) – Path to save figure to, only required if save=True



### 3.4.6 mag\_calc

Functions to simulate Wood Anderson traces, pick maximum peak-to-peak amplitudes write these amplitudes and periods to SEISAN s-files and to calculate magnitudes from this and the information within SEISAN s-files.

Written as part of the EQcorrscan package by Calum Chamberlain - first written to implement magnitudes for the 2015 Wanaka aftershock sequence, written up by Warren-Smith [2014/15].

Copyright 2015 Calum Chamberlain

This file is part of EQcorrscan.

EQcorrscan is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

EQcorrscan is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with EQcorrscan. If not, see <<http://www.gnu.org/licenses/>>.

`mag_calc.Amp_pick_sfile(sfile, datapath, respdir, chans=['Z'], var_wintype=True, winlen=0.9, pre_pick=0.2, pre_filt=True, lowcut=1.0, highcut=20.0, corners=4)`

Function to read information from a SEISAN s-file, load the data and the picks, cut the data for the channels given around the S-window, simulate a Wood Anderson seismometer, then pick the maximum peak-to-trough amplitude.

Output will be put into a mag\_calc.out file which will be in full S-file format and can be copied to a REA database.

#### Parameters

- **datapath** (*String*) – Path to the waveform files - usually the path to the WAV directory
- **respdir** (*String*) – Path to the response information directory
- **chans** (*List of strings*) – List of the channels to pick on, defaults to ['Z'] - should just be the orientations, e.g. Z,1,2,N,E
- **var\_wintype** (*Bool*) – If True, the winlen will be multiplied by the P-S time if both P and S picks are available, otherwise it will be multiplied by the hypocentral distance\*0.34 - derived using a p-s ratio of 1.68 and S-velocity of 1.5km/s to give a large window, defaults to True
- **winlen** (*Float*) – Length of window, see above parameter, if var\_wintype is False Then this will be in seconds, otherwise it is the multiplier to the p-s time, defaults to 0.5
- **pre\_pick** (*Float*) – Time before the s-pick to start the cut window, defaults to 0.2
- **pre\_filt** (*Bool*) – To apply a pre-filter or not, defaults to True
- **lowcut** (*Float*) – Lowcut in Hz for the pre-filter, defaults to 1.0
- **highcut** (*Float*) – Highcut in Hz for the pre-filter, defaults to 20.0
- **corners** (*Int*) – Number of corners to use in the pre-filter

`mag_calc.SVD_moments(U, s, V, stachans, event_list, n_SVs=4)`

Function to convert basis vectors calculated by singular value decomposition (see the SVD functions in clustering) into relative magnitudes.

#### Parameters

- **U** (*List of np.ndarray*) – List of the input basis vectors from the SVD, one array for each channel used.
- **s** (*List of nd.array*) – List of the singular values, one array for each channel



- **v** (*List of np.ndarray*) – List of output basis vectors from SVD, one array per channel.
- **stachans** (*List of string*) – List of station.channel input
- **event\_list** (*List of list*) – List of events for which you have data, such that event\_list[i] corresponds to stachans[i], U[i] etc. and event\_list[i][j] corresponds to event j in U[i]

type n\_SVs: int :param n\_SVs: Number of singular values to use, defaults to 4.

**Returns** M, np.array of relative moments

mag\_calc.\_GSE2\_PAZ\_read(*GSEfile*)

Function to read the instrument response information from a GSE Poles and Zeros file as generated by the SEISAN program RESP.

Format must be CAL2, not coded for any other format at the moment, contact the author to add others in.

**Parameters** **GSEfile** (*Str*) – Path to GSE file

**Returns** Dict of poles, zeros, gain and sensitivity

mag\_calc.\_find\_resp(*station, channel, network, time, delta, directory*)

Helper function to find the response information for a given station and channel at a given time and return a dictionary of poles and zeros, gain and sensitivity.

**Parameters**

- **station** (*String*) – Station name (as in the response files)
- **channel** (*String*) – Channel name (as in the response files)
- **network** (*String*) – Network to scan for, can be a wildcard
- **time** (*datetime.datetime*) – Date-time to look for response information
- **delta** (*float*) – Sample interval in seconds
- **directory** (*String*) – Directory to scan for response information

**Returns** Dictionary

mag\_calc.\_max\_p2t(*data, delta*)

Function to find the maximum peak to trough amplitude and period of this amplitude. Originally designed to be used to calculate magnitudes (by taking half of the peak-to-trough amplitude as the peak amplitude).

**Parameters**

- **data** (*ndarray*) – waveform trace to find the peak-to-trough in.
- **delta** (*float*) – Sampling interval in seconds

**Returns** tuple of (amplitude, period, time) with amplitude in the same scale as given in the input data, and period in seconds, and time in seconds from the start of the data window.

mag\_calc.\_pairwise(*iterable*)

Wrapper on itertools for SVD\_magnitude

mag\_calc.\_sim\_WA(*trace, PAZ, seedresp, water\_level*)

Function to remove the instrument response from a trace and return a de-measured, de-trended, Wood Anderson simulated trace in its place.

Works in-place on data and will destroy your original data, copy the trace before giving it to this function!

**Parameters**

- **trace** (*obspy.Trace*) – A standard obspy trace, generally should be given without pre-filtering, if given with pre-filtering for use with amplitude determination for magnitudes you will need to worry about how you cope with the response of this filter yourself.
- **PAZ** (*dict*) – Dictionary containing lists of poles and zeros, the gain and the sensitivity.
- **water\_level** (*int*) – Water level for the simulation.

**Returns** obspy.Trace

`mag_calc.dist_calc(loc1, loc2)`

Function to calculate the distance in km between two points, uses the flat Earth approximation

**Parameters**

- **loc1** (*Tuple*) – Tuple of lat, lon, depth (in decimal degrees and km)
- **loc2** (*Tuple*) – Tuple of lat, lon, depth (in decimal degrees and km)

### 3.4.7 stacking

Utility module of the EQcorrscan package to allow for different methods of stacking of seismic signal in one place.

Calum Chamberlain 24/06/2015

Copyright 2015 Calum Chamberlain

This file is part of EQcorrscan.

EQcorrscan is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

EQcorrscan is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with EQcorrscan. If not, see <<http://www.gnu.org/licenses/>>.

`stacking.PWS_stack(streams, weight=2)`

Function to compute the phase weighted stack of a series of streams. Recommend aligning the traces before stacking.

**Parameters**

- **streams** (*list of obspy.Stream*) – List of Stream to stack
- **weight** (*float*) – Exponent to the phase stack used for weighting.

**Returns** obspy.Stream

`stacking.align_traces(trace_list, shift_len, master=False)`

Function to align traces relative to each other based on their cross-correlation value

**Parameters**

- **trace\_list** (*List of Traces*) – List of traces to align
- **shift\_len** (*int*) – Length to allow shifting within in samples
- **master** (*obspy.Trace*) – Master trace to align to, if set to False will align to the largest amplitude trace (default)

**Returns** list of shifts for best alignment in seconds

`stacking.linstack(streams)`

Function to compute the linear stack of a series of seismic streams of multiplexed data

**Parameters** **stream** – List of streams to stack

**Returns** stack - Stream

### 3.4.8 catalogue2DD

Module written by Calum Chamberlain as part of the EQcorrscan package.

This module contains functions to convert a seisan catalogue to files ready for relocation in hypoDD - it will generate both a catalogue (dt.ct) file, event file (event.dat), station information file (station.dat), and a correlation output file correlated every event in the catalogue with every other event to optimize the picks (dt.cc).

The correlation routine relies on obspy's `xcorrPickCorrection` function from the `obspy.signal.cross_correlation` module. This function optimizes picks to better than sample accuracy by interpolating the correlation function and finding the maximum of this rather than the true maximum correlation value. The output from this function is stored in the dt.cc file.

Information for the station.dat file is read from SEISAN's STATION0.HYP file

Earthquake picks and locations are taken from the catalogued s-files - these must be pre-located before entering this routine as origin times and hypocentre locations are needed for event.dat files.

Copyright 2015 Calum Chamberlain

This file is part of EQcorrscan.

EQcorrscan is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

EQcorrscan is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with EQcorrscan. If not, see <<http://www.gnu.org/licenses/>>.

`catalogue2DD._av_weight (W1, W2)`

Function to convert from two seisan weights (0-4) to one hypoDD weight(0-1)

#### Parameters

- **W1** (*str*) – Seisan input weight (0-4)
- **W2** (*str*) – Seisan input weight (0-4)

#### Returns

*str*

`catalogue2DD._cc_round (num, dp)`

Convenience function to take a float and round it to dp padding with zeros to return a string

#### Parameters

- **num** (*float*) – Number to round
- **dp** (*int*) – Number of decimal places to round to.

#### Returns

*string*

`catalogue2DD.readSTATION0 (path, stations)`

Function to read the STATION0.HYP file on the path given. Outputs written in station.dat file.

#### Parameters

- **path** (*String*) – Path to the STATION0.HYP file
- **station** (*List*) – Stations to look for

#### Returns

List of tuples of station, lat, long, elevation

`catalogue2DD.write_catalogue (event_list, max_sep=1, min_link=8)`

Function to write the dt.ct file needed by hypoDD - takes input event list from `write_event` as a list of tuples of event id and sfile. It will read the pick information from the seisan formatted s-file using the `Sfile_util` utilities.

**Parameters**

- **event\_list** (*List of tuple*) – List of tuples of event\_id (int) and sfile (String)
- **max\_sep** (*float*) – Maximum separation between event pairs in km
- **min\_link** (*int*) – Minimum links for an event to be paired

**Returns** List stations

catalogue2DD.**write\_correlations**(*event\_list, wavbase, extract\_len, pre\_pick, shift\_len, lowcut=1.0, highcut=10.0, max\_sep=4, min\_link=8, coh\_thresh=0.0*)

Function to write a dt.cc file for hypoDD input - takes an input list of events and computes pick refinements by correlation.

Note that this is **NOT** fast.

**Parameters**

- **event\_list** (*List of tuple*) – List of tuples of event\_id (int) and sfile (String)
- **wavbase** (*string*) – Path to the seisan wave directory that the wavefiles in the S-files are stored
- **extract\_len** (*float*) – Length in seconds to extract around the pick
- **pre\_pick** (*float*) – Time before the pick to start the correlation window
- **shift\_len** (*float*) – Time to allow pick to vary
- **lowcut** (*float*) – Lowcut in Hz - default=1.0
- **highcut** (*float*) – Highcut in Hz - default=10.0
- **max\_sep** (*float*) – Maximum separation between event pairs in km
- **min\_link** (*int*) – Minimum links for an event to be paired

catalogue2DD.**write\_event**(*sfile\_list*)

Function to write out an event.dat file of the events

**Parameters** **sfile\_list** (*List*) – List of s-files to sort and put into the database

**Returns** List of tuples of event ID (int) and Sfile name

## **b**

bright\_lights, 12

## **c**

catalogue2DD, 38

clustering, 28

## **e**

EQcorrscan\_plotting, 32

## **f**

findpeaks, 26

## **l**

lag\_calc, 21

## **m**

mag\_calc, 35

match\_filter, 18

## **p**

pre\_processing, 31

## **s**

Sfile\_util, 23

stacking, 38

## **t**

template\_gen, 16