EQcorrscan

*Correlation for earthquake detection in Python*

# EQcorrscan Documentation

***Release 0.1.1-alpha***

**Calum John Chamberlain**

October 13, 2015

# CONTENTS

**EQcorrscan**

*Correlation for earthquake detection in Python*

CHAPTER

# ONE

# 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 at 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.

CHAPTER

TWO

# 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.

CHAPTER

THREE

# CONTENTS:

## 3.1 Introduction to the EQcorrscan package

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

### 3.1.1 Installation

Most codes should work without any effort on your part. However you must install the packages this package relies on yourself, this includes the follwing packages:

- matplotlib
- numpy
- scipy
- obspy
- joblib
- openCV (2)

This install has only been tested on Linux machines and even then has some issues when installing on 32-Bit versus 64-Bit machines. In this instance you should be prepared for small differences in the results of your correlations relating to foating-point truncation differences between 32 and 64-Bit machines.

If you plan to run the bright_lights.py routines you will need to have NonLinLoc installed on your machine. 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.

### 3.1.2 Functions

This package is divided into sub-directories of *core*, *par* and *utils*. The *utils* directory contains simple functions for integration with seisan, these is 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.

Within *par* you will find parameter files which you will need to edit for each of the *core* scripts. *core* scripts often call on multiple *par* files so be sure to set them all up. The *template_gen_par.py* script is used by all *core* modules and must be set-up. Within this you will define all your template parameters. Currently the templates must all be of the same length, but this may change in a future release.

Within *core* you will find the core routines to generate templates, *(template_gen.py)* search for likely templates *(bright_lights.py)* and compute cross-channel correlations from these templates *(match_filter.py)*.

## 3.2 EQcorrscan tutorial

Welcome to EQcorrscan - this package is designed to compute earthquake detections using a paralleled match-filter network cross-correlation routine. The inner loop of this package is the cross-correaltion of templates of seismic data with daylong seismic data. This inner function is the openCV.match_template function - this appears to be a well optimized cross-correlation function 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 this won't be done for you. The list of these is in the Introduction section of this documentation.

As you will see, this package is divided into three main sub-modules, the Core, Utils and Par 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 *Seian* s-file;

**match_filter** The main match-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 reloactions.

The Par sub-module contains parameter files which are provided to allow for simple bulk processing of large datasets. These *MUST* be edited by the user for their dataset.

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 reloaction software), calculating magnitudes, clustering detections, stacking detections, making pretty plots, and processing seismic data in the same way repeatedly using *Obspy*'s functionality.

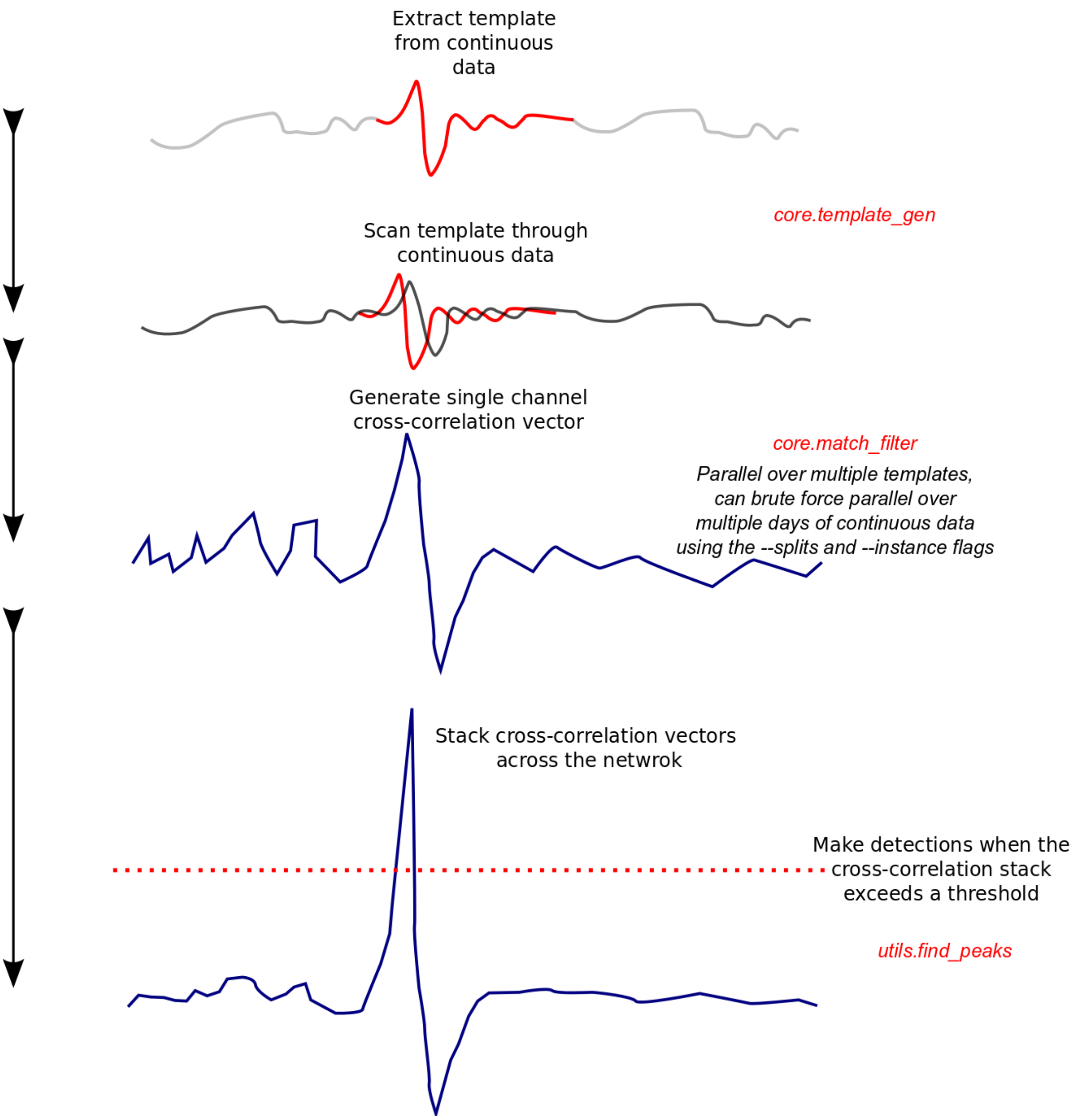
### 3.2.1 Match-filter detection

In this section we will discuss generating a template from a *Seisan* s-file, and using this template to scan for similar earthquakes within a day of data. This single template and single day of data does not truely 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 (*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-file provided is one of these events.

The main processing flow is outlined in the figure below, note the main speedups in this process are achioeved by running multiple templates at once, however this increaces memory usage. If memory is 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/python
"""
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 match-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.  The use of this brightness function is not
    included in this tutorial script yet because it is still in beta.

This package is dstributed 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:
from obspy import read, Stream
from core import template_gen, match_filter
from par import match_filter_par as matchdef
from utils import pre_processing, Sfile_util
import glob

# Now we find the s-file we want to use to generate a template from
sfiles=glob.glob('test_data/tutorial_data/*L.S*')

# Generate the template from these sfiles:
templates=[] # Open a list to be filled - only applies for multiple templates
template_names=[] # List of template names for later ID
i=0 # Template name iterator
for sfile in 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:
        if not 'wavefiles' in locals():
            wavefiles=glob.glob('test_data/tutorial_data/'+\
                                    pick.station+'.*')
        else:
            wavefiles+=glob.glob('test_data/tutorial_data/'+\
                                  pick.station+'.*')
    wavefiles=list(set(wavefiles))
    for wavefile in wavefiles:
        print 'Reading data from '+wavefile
        if not 'st' in locals():
            st=read(wavefile)
        else:
            st+=read(wavefile)
```

```
    st=st.merge(fill_value='interpolate')
    day=st[0].stats.starttime.date
    for tr in st:
        tr=pre_processing.dayproc(tr, 1.0, 20.0, 3, 100.0,\
                                  matchdef.debug, day)
    # Apply a small amoutn of delay before the pick
    for pick in picks:
        pick.time=pick.time-0.1
    template=template_gen._template_gen(picks, st, 1.0, 'all')
    # This will generate an obspy.Stream object
    # Append this Stream to the list of templates
    templates+=[template]
    template_names.append('tutorial_'+str(i))
    # Plot the template just to check that all is well!
    template.plot(size=(800,600), equal_scale=False)
    # Save template for later
    template.write('test_data/tutorial_data/'+template_names[i]+'_template.ms',\
                   format='MSEED')
    i+=1
    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:
    print 'Reading data from: test_data/tutorial_data/'+stachan[0]+'.*..*'+stachan[1][-1]+'.*'
    # Generate a new stream object and add to it
    if not 'st' in locals():
        st=read('test_data/tutorial_data/'+stachan[0]+'.*..*'+stachan[1][-1]+'.*')
    else:
        st+=read('test_data/tutorial_data/'+stachan[0]+'.*..*'+stachan[1][-1]+'.*')

# 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,\
                              matchdef.debug, day)

# Compute detections
detections=match_filter.match_filter(template_names, templates, st,\
                                     matchdef.threshold, matchdef.threshtype,\
                                     matchdef.trig_int, True,\
                                     'temp_0')

# 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:
    f.write(detection.template_name+', '+str(detection.detect_time)+\
            ', '+str(detection.detect_val)+', '+str(detection.threshold)+\
```

```
            ', '+str(detection.no_chans)+'\n')
f.close()
```

## 3.3 Core

Core programs for the EQcorrscan project.

### 3.3.1 bright_lights

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

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

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

This main section contains a script, LFE_search.py which demonstrates the usageof the built in functions from template generation from picked waveformsthrough detection by match filter of continuous data to the generation of lagtimes to be used for relative locations.

The match-filter routine described here was used a previous Matlab code for theChamberlain et al. 2014 G-cubed publication. The basis for the lag-timegeneration 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

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

**Returns** detections as :class: DETECTION

`bright_lights._node_loop` (*stations*, *lags*, *stream*, *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
- **i** (*int*) – Index of loop for parallelisation
- **mem_issue** (*bool*) – If True will write to disk rather than storing data in RAM
- **instance** (*int*) – instance for bulk parallelisation, only used if mem_issue=true
- **plot** (*bool*) – 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 coversion
- **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][:]
- **nodes** (*list, tuple*) – List of node points where nodes[i] referes to stations[i] and nodes[:][:][0] is latitude in degrees, nodes[:][:][1] is longitude in degrees, nodes[:][:][2] is depth in km.
- **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
- **mindepth** – Upper limit of volume
- **maxdepth** (*float*) – Lower limit of volume
- **corners** (*matplotlib.Path*) – 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][:]
- **nodes** (*list, tuple*) – List of node points where nodes[i] referes to stations[i] and nodes[:][:][0] is latitude in degrees, nodes[:][:][1] is longitude in degrees, nodes[:][:][2] is depth in km.
- **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
- **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.**brightness**(*stations*, *nodes*, *lags*, *stream*, *threshold*, *thresh_type*, *coherence_thresh*, *instance=0*, *matchdef=False*, *defaults=False*, *pre_pick=0.2*)

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][:]
- **nodes** (*list, tuple*) – List of node points where nodes[i] referes to stations[i] and nodes[:][:][0] is latitude in degrees, nodes[:][:][1] is longitude in degrees, nodes[:][:][2] is depth in km.
- **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.
- **data** – Data through which to look for detections.
- **threshold** (*float*) – Threshold value for detection of template within the brightness function
- **thresh_type** (*str*) – Either MAD or abs where MAD is the Median Absolute Deviation and abs is an absoulte brightness.
- **coherence_thresh** (*tuple of floats*) – Threshold for removing incoherant 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-kchan/b where kchan is the number of channels used to compute the coherence
- **pre_pick** (*float*) – Seconds before the detection time to include in template

**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 incoherant 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 theapplication of cross-correlation of seismic data for the detection of repeatingevents.

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

This main section contains a script, LFE_search.py which demonstrates the usageof the built in functions from template generation from picked waveformsthrough detection by match filter of continuous data to the generation of lagtimes to be used for relative locations.

The match-filter routine described here was used a previous Matlab code for theChamberlain et al. 2014 G-cubed publication. The basis for the lag-timegeneration 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



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

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

**Parameters**

- **picks** – Picks to extract data around
- **st** – Stream to etract 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 time default is 0.05 seconds
- **plot** (*bool*) – To plot the template or not, default is True

`template_gen.`**`from_contbase`**(*sfile*, *lowcut=None*, *highcut=None*, *samp_rate=None*, *filt_order=None*, *length=None*, *prepick=None*)

Function to read in picks from sfile then generate the template from the picks within this and the wavefiles from the continous 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).
- **lowcut** (*float*) – Low cut (Hz), if set to None will look in template defaults file
- **lowcut** – High cut (Hz), if set to None will look in template defaults 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 template defaults file.
- **prepick** (*float*) – Pre-pick time in seconds

template_gen.**from_sfile**(*sfile*, *lowcut=None*, *highcut=None*, *samp_rate=None*, *filt_order=None*, *length=None*)

Function to read in picks from sfile then generate the template from the picks within this and the wavefile 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.
- **lowcut** (*float*) – Low cut (Hz), if set to None will look in template defaults file
- **lowcut** – High cut (Hz), if set to None will look in template defaults 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 template defaults file.

### 3.3.3 match_filter

Function to cross-correlate templates generated by template_gen function withdata and output the detecitons. The main component of this script is thenormxcorr2 function from the openCV image processing package. This is a highlyoptimized and accurate normalized cross-correlation routine. The details ofthis code can be found here: - http://www.cs.ubc.ca/research/deaton/remarks_ncc.htmlThe cpp code was first tested using the Matlab mex wrapper, and has since beenported to a python callable dynamic library.

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

This main section contains a script, LFE_search.py which demonstrates the usageof the built in functions from template generation from picked waveformsthrough detection by match filter of continuous data to the generation of lagtimes to be used for relative locations.

The match-filter routine described here was used a previous Matlab code for theChamberlain et al. 2014 G-cubed publication. The basis for the lag-timegeneration 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



**class** `match_filter.`**`DETECTION`**(*template_name*, *detect_time*, *no_chans*, *detect_val*, *threshold*, *typeofdet*, *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 cccsum_val** float

**param cccsum_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 cccsum.

**type typeofdet** str

**param typeofdet** Type of detection, STA, corr, bright

`match_filter.`**`_channel_loop`**(*templates*, *stream*)

Loop to generate cross channel correaltion 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.
- **`stream`** – A single obspy.Stream object containing daylong seismic data to be correlated through using the templates. This is in effect the image

**Returns** New list of :class: 'numpy.array' objects. These will contain the correlation sums for each template for this day of data.

**Returns** list of ints as number of channels used for each cross-correlation `match_filter._template_loop` (*template*, *chan*, *station*, *channel*, *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

`match_filter.match_filter` (*template_names*, *templates*, *stream*, *threshold*, *threshold_type*, *trig_int*, *plotvar*, *tempdir=False*)

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**

- **templates** (*list :class: 'obspy.Stream'*) – A list of templates of which each template is a Stream of obspy traces containing seismic data and header information.
- **stream** – 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.
- **threshold** (*float*) – A threshold value set based on the threshold_type :type threshold_type: str
- **threshold_type** – The type of threshold to be used, can be MAD, absolute or av_chan_corr. MAD threshold is calculated as the threshold*(median(abs(cccsum))) where cccsum is the cross-correlation sum for a given template. absolute threhsold 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.
- **tempdir** (*String or False*) – Directory to put temporary files, or False

**Returns**

**class** 'DETECTIONS' detections for each channel formatted as :class: 'obspy.UTCDateTime' objects.

`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-correaltion. Documentation on this function can be found here:

•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 correaltion 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-obspy - used for lots of common seismological processing**
  - **requires:**
    - numpy
    - scipy
    - matplotlib
- NonLinLoc - used outside of all codes for travel-time generation



`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 occured 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

Codes to run basic utility functions for integration with seisan and to find peaks in noisy data.

### 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.



**class** `Sfile_util.`**`EVENTINFO`**(*time*, *loc_mod_ind*, *dist_ind*, *ev_id*, *latitude*, *longitude*, *depth*, *depth_ind*, *loc_ind*, *agency*, *nsta*, *t_RMS*, *Mag_1*, *Mag_1_type*, *Mag_1_agency*, *Mag_2*, *Mag_2_type*, *Mag_2_agency*, *Mag_3*, *Mag_3_type*, *Mag_3_agency*)

Header information for seisan events

**class** `Sfile_util.PICK` (*station*, *channel*, *impulsivity*, *phase*, *weight*, *polarity*, *time*, *coda*, *amplitude*, *peri*, *azimuth*, *velocity*, *AIN*, *SNR*, *azimuthres*, *timeres*, *finalweight*, *distance*, *CAZ*)

Pick information for seisan implimentation

`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-charectar 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*)

Fucntion 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.

`findpeaks.`**`find_peaks2`**(*arr*, *thresh*, *trig_int*, *debug=0*, *maxwidth=10*, *starttime=UTCDateTime(1970, 1, 1, 0, 0)*, *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=UTCDateTime(1970, 1, 1, 0, 0)*, *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=UTCDateTime(1970, 1, 1, 0, 0)*, *samp_rate=1.0*)

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

Depreciated 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 seperate 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

Code to compute the linkage between seismograms and cluster them accordingly

Written by Calum Chamberlain, in alpha stages of development as of 24/06/2015

Implimented to streamline templates after template detection in beamforming methods, employed by implimentation of Frank et al. code.

As such this code is designed to work only for templates with the same channels

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**(*templates*)

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

**Parameters** **templates** (*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`**(*templates*, *show=True*)

Function to take a set of templates and cluster them, will return distance matrix of grouped templates

**Parameters**

- **`templates`** (*List of Obspy.Stream*) – List of templates to compute clustering for
- **`show`** (*bool*) – plot linkage on screen if True, defaults to True

**Returns** List of cluster groups, array of length len(templates), with each number relating to a cluster

**Note: Not fully feautured yet, returns the Z matrix, but doesn't tell you what can be clustered.**

`clustering.`**`corr_cluster`**(*traces*, *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**

- **`traces`** (*List of :class:obspy.Trace*) – Traces to compute similarity between
- **`thrsh`** – Correlation threshold between -1-1

**Returns** np.ndarray of bool

`clustering.`**`cross_chan_coherence`**(*st1*, *st2*)

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

**Parameters**

- **`st1`** (*obspy Stream*) – Stream one
- **`st2`** (*obspy Stream*) – Stream two

**Returns** cross channel coherence, float - normalized by number of channels

`clustering.`**`distance_matrix`**(*templates*)

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** **`template`** – List of the streams to compute the distance matrix for

**Returns** ndarray - distance matrix

`clustering.`**`empirical_SVD`**(*templates*, *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 emprical subspace method of Barrett & Beroza, 2014 - SRL.

**Parameters**

- **`templates`** (*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*, *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.
- **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, chanels and networks to also extract data for using an average delay.

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

clustering.**group_delays**(*templates*)

Function to group template waveforms according to their delays

**Parameters** **templates** (*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

### 3.4.4 pre_processing

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



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

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

### 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 wildcards.

**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.**cumulative_detections**(*dates*, *template_names*, *save=False*, *savefile=''*)

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 seperately, 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** (*obspy.Stream*) – Stream of data to be plotted as the base (black)
- **template** (*obspy.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.**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.

**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 alligned 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 alignement 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** Alligned 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.**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_seismplot**(*stations*, *nodes*)

Function to plot seismicity and stations in a 3D, movable, zoomable space using matplotlibs 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**(*cccsum*, *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**

- **cccsum** (*numpy.array*) – Array of the cross-channel cross-correlation sum
- **trace** (*obspy.Trace*) – A sample trace from the same time as cccsum
- **threshold** (*float*) – Detection threshold within cccsum
- **save** (*Bool, optional*) – If True will svae 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 informaiton within SEISAN s-files.

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

`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 - dervided 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.`**`_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 repsonse 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.**_sim_WA**(*trace*, *PAZ*, *seedresp*, *water_level*)

Function to remove the insturment response from a trace and return a de-meaned, de-trended, Wood Anderson simulated trace in it's 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 determiniation 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 calcualte 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.

In alpha stages and only with linear stacking implimented thusfar

Calum Chamberlain 24/06/2015



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*`)`

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

**Parameters**

- **`trace_list`** (*List of Traces*) – List of traces to allign
- **`shift_len`** (*int*) – Length to allow shifting within in samples

**Returns** list of shifts for best allignment 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 oiutput 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.



`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._separation`(*loc1*, *loc2*)

Function to calculate the distance between two points in the earth

**Parameters**

- **loc1** (*tuple (float, float, float)*) – First point location as lat, long, depth in deg, deg, km
- **loc2** (*tuple (float, float, float)*) – First point location as lat, long, depth in deg, deg, km

**Returns** distance in km (float)

`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 formated 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 seperation 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 refienements 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 correclation window
- **shift_len** (*float*) – Time to allow pick to vary
- **lowcut** (*float*) – Lowcut in Hz - default=1.0
- **highcut** (*float*) – Highcut in Hz - deafult=10.0
- **max_sep** (*float*) – Maximum seperation 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

## 3.5 Par

User-editable codes to input parameters for *core* files. Scripts are coded in the files with a full description of the parameters  Outline definitions for match filter python code



Outline definitions for template generation

# PYTHON MODULE INDEX

# T



# W