



EQcorrscan
*Correlation for earthquake
detection in Python*

EQcorrscan Documentation

Release 0.1.1-alpha

Calum John Chamberlain

July 16, 2015

CONTENTS

1	Introduction to the EQcorrscan package	3
1.1	Installation	3
1.2	Functions	3
2	EQcorrscan tutorial	5
3	Core	11
3.1	bright_lights	11
3.2	template_gen	13
3.3	match_filter	15
3.4	lag_calc	17
4	Utils	19
4.1	Sfile_util	19
4.2	findpeaks	20
4.3	clustering	20
4.4	pre_processing	21
4.5	EQcorrscan_plotting	22
4.6	mag_calc	23
4.7	stacking	24
4.8	catalogue2DD	25
5	Par	27
	Python Module Index	29
	Index	31

Contents:

INTRODUCTION TO THE EQCORRSCAN PACKAGE

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

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 following 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 floating-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.

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*).

EQCORRSCAN TUTORIAL

THIS TUTORIAL IS NOT REALLY WRITTEN YET!

You must first set-up your parameter files in the *par* directory. You can leave these as the default settings for now, but study the parameters so that you understand what each one is doing.

Then run the *LFE_search.py* routine in this top directory, running this will run all the *core* routines to search for templates, generate templates and compute the cross-channel cross-correlation values for the templates. Finally it will output detections from these templates.

The following is verbatim the *LFEsearch.py* routine which outlines usage of this package:

```
#!/usr/bin/python

#-----
#   Purpose:      Script to call all elements of EQcorrscan module to search
#                 continuous data for likely LFE repeats
#   Author:       Calum John Chamberlain
#-----

"""
LFEsearch - Script to generate templates from previously picked LFE's and then
search for repeats of them in continuous data.
"""

import sys, os, glob
bob=os.path.realpath(__file__)
bob=bob.split('/')
path="/"
for i in xrange(len(bob)):
    path+=bob[i]+'/'
print path
sys.path.insert(0,path)

from par import template_gen_par as templatedef
from par import match_filter_par as matchdef
#from par import lagcalc as lagdef
from obspy import UTCDateTime, read as obsread
# First generate the templates
from core import template_gen

Split=False
if len(sys.argv) == 2:
    flag=str(sys.argv[1])
    if flag == '--debug':
        Test=True
        Prep=False
    elif flag == '--debug-prep':
        Test=False
        Prep=True
```

```
    else:
        raise ValueError("I don't recognise the argument, I only know --debug and --debug-prep")
elif len(sys.argv) == 5:
    # Arguments to allow the code to be run in multiple instances
    Split=True
    Test=False
    Prep=False
    args=sys.argv[1:len(sys.argv)]
    for i in xrange(len(args)):
        if args[i] == '--instance':
            instance=int(args[i+1])
            print 'I will run this for instance '+str(instance)
        elif args[i] == '--splits':
            splits=int(args[i+1])
            print 'I will divide the days into '+str(splits)+' chunks'

elif not len(sys.argv) == 1:
    raise ValueError("I only take one argument, no arguments, or two flags with arguments")
else:
    Test=False
    Prep=False
    Split=False

templates=[]
delays=[]
stations=[]
print 'Template generation parameters are:'
print 'sfilebase: '+templatedef.sfilebase
print 'samp_rate: '+str(templatedef.samp_rate)+' Hz'
print 'lowcut: '+str(templatedef.lowcut)+' Hz'
print 'highcut: '+str(templatedef.highcut)+' Hz'
print 'length: '+str(templatedef.length)+' s'
print 'swin: '+templatedef.swin+'\n'
for sfile in templatedef.sfiles:
    print 'Working on: '+sfile+'\r'
    if not os.path.isfile(templatedef.saveloc+'/'+sfile+'_template.ms'):
        template=template_gen.from_contbase(templatedef.sfilebase+'/'+sfile)

        print 'saving template as: '+templatedef.saveloc+'/'+sfile+str(template[0].stats.starttime)+'.ms'
        template.write(templatedef.saveloc+'/'+sfile+'_template.ms',format="MSEED")
    else:
        template=obsread(templatedef.saveloc+'/'+sfile+'_template.ms')
    templates+=[template]
    # Will read in seisan s-file and generate a template from this,
    # returned name will be the template name, used for parsing to the later
    # functions

for tfile in templatedef.tfiles:
    # Loop through pre-existing template files
    sys.stdout.write("\rReading in pre-existing template: "+tfile+"\r")
    sys.stdout.flush()
    templates.append(obsread(tfile))

for template in templates:
    # Calculate the delays for each template, do this only once so that we
    # don't have to do it heaps!
    # Check that all templates are the correct length
    for tr in template:
        if not templatedef.samp_rate*templatedef.length == tr.stats.npts:
            raise ValueError('Template for '+tr.stats.station+'.'+tr.stats.channel+' is not the correct length, recut.'+tr.stats.channel)
```

```

        ' It is: '+str(tr.stats.npts)+' and should be '+
        str(templatedef.samp_rate*templatedef.length))
# Generate list of stations in templates
for tr in template:
    # Correct FOZ channels
    if tr.stats.station=='FOZ' and len(tr.stats.channel)==3:
        tr.stats.channel='HH'+tr.stats.channel[2]
    if len(tr.stats.channel)==3:
        stations.append(tr.stats.station+'.'+tr.stats.channel[0]+\
            '*' +tr.stats.channel[2]+'.'+tr.stats.network)
        tr.stats.channel=tr.stats.channel[0]+tr.stats.channel[2]
    elif len(tr.stats.channel)==2:
        stations.append(tr.stats.station+'.'+tr.stats.channel[0]+\
            '*' +tr.stats.channel[1]+'.'+tr.stats.network)
    else:
        raise ValueError('Channels are not named with either three or two charectars')

# Template generation and processing is over, now to the match-filtering

# Sort stations into a unique list - this list will ensure we only read in data
# we need, which is VERY important as I/O is very costly and will eat memory
stations=list(set(stations))

# Now run the match filter routine
from core import match_filter
from obspy import read as obsread
# from obspy.signal.filter import bandpass
# from obspy import Stream, Trace
# import numpy as np
from utils import pre_processing
from joblib import Parallel, delayed

# Loop over days
ndays=len(matchdef.dates)
newsfiles=[]
f=open('detections/run_start_'+str(UTCDateTime().year)+'\
    str(UTCDateTime().month).zfill(2)+'\
    str(UTCDateTime().day).zfill(2)+'T'+\
    str(UTCDateTime().hour).zfill(2)+str(UTCDateTime().minute).zfill(2), 'w')
f.write('template, detect-time, cccsum, threshold, number of channels\n')
print 'Will loop through '+str(ndays)+' days'
if Split:
    if instance==splits:
        ndays=ndays-(ndays/splits)*(splits-1)
        dates=matchdef.dates[-ndays:]
    else:
        ndays=ndays/splits
        dates=matchdef.dates[ndays*instance:(ndays*instance)+ndays]
    print 'This instance will run for '+str(ndays)+' days'
    print 'This instance will run from '+str(startdate)
else:
    dates=matchdef.dates

for day in dates:
    if 'st' in locals():
        del st
    # Read in data using obspy's reading routines, data format will be worked
    # out by the obspy module
    # Note you might have to change this bit to match your naming structure
    actual_stations=[] # List of the actual stations used
    for stachan in stations:

```

```

# station is of the form STA.CHAN, to allow these to be in an odd
# arrangements we can separate them
station=stachan.split('.')[0]
channel=stachan.split('.')[1]
netcode=stachan.split('.')[2]
if not Test:
    # Set up the base directory format
    for base in matchdef.contbase:
        if base[2]==netcode:
            contbase=base
    if not 'contbase' in locals():
        raise NameError('contbase is not defined for '+netcode)
    baseformat=contbase[1]
    if baseformat=='yyyy/mm/dd':
        daydir=str(day.year)+'/'+str(day.month).zfill(2)+'/'+\
            str(day.day).zfill(2)
    elif baseformat=='Yyyyy/Rjjj.01':
        daydir='Y'+str(day.year)+'/R'+str(day.julday).zfill(3)+'.01'
    elif baseformat=='yyyymmdd':
        daydir=str(day.year)+str(day.month).zfill(2)+str(day.day).zfill(2)

    # Try and find the appropriate files
    if glob.glob(contbase[0]+'/'+daydir+'/*'+station+'*'+channel+'*'):
        if not 'st' in locals():
            st=obsread(contbase[0]+'/'+daydir+'/*'+station+'*'+channel+'*')
        else:
            st+=obsread(contbase[0]+'/'+daydir+'/*'+station+'*'+channel+'*')
        actual_stations.append(station) # Add to this list only if we have the data
    else:
        print 'No data for '+stachan+' for day '+daydir+' in '\
            +contbase[0]

else:
    fname='test_data/'+station+'-'+channel+'-'+str(day.year)+\
        '-' +str(day.month).zfill(2)+\
        '-' +str(day.day).zfill(2)+'-processed.ms'
    if glob.glob(fname):
        if not 'st' in locals():
            st=obsread(fname)
        else:
            st+=obsread(fname)
        actual_stations.append(station)
actual_stations=list(set(actual_stations))

if not 'st' in locals():
    print 'No data found for day: '+str(day)
elif len(actual_stations) < matchdef.minsta:
    print 'Data from fewer than '+str(matchdef.minsta)+' stations found, will not detect'
else:
    if not Test:
        # Process data
        print 'Processing the data for day '+daydir
        if matchdef.debug >= 4:
            for tr in st:
                tr=pre_processing.dayproc(tr, templatedef.lowcut, templatedef.highcut,\
                    templatedef.filter_order, templatedef.samp_rate,\
                    matchdef.debug, day)
        else:
            st=Parallel(n_jobs=len(st))(delayed(pre_processing.dayproc)(tr, templatedef.lowcut,\
                templatedef.highcut,\
                templatedef.filter_order,\
                templatedef.samp_rate,\
                matchdef.debug, day)\
                for tr in st)

```

```

if not Prep:
    # Call the match_filter module - returns detections, a list of detections
    # contained within the detection class with elements, time, template,
    # number of channels used and cross-channel correlation sum.
    print 'Running the detection routine'
    template_names=[tfile.split('/')[1] for tfile in templatedef.tfiles]
    template_names.append(templatedef.sfiles)
    detections=match_filter.match_filter(template_names, templates, st,
                                         matchdef.threshold, matchdef.thresholdtype,
                                         matchdef.trig_int, matchdef.plot)

    for detection in detections:
        # output detections to file
        f.write(detection.template_name+', '+str(detection.detect_time)+'\n'
               ', '+str(detection.detect_val)+'', '+str(detection.threshold)+'\n'
               ', '+str(detection.no_chans)+'\n')
        print 'template: '+detection.template_name+' detection at: '\n
              +str(detection.detect_time)+' with a cccsum of: '+str(detection.detect_val)
    if detections:
        f.write('\n')
    else:
        for tr in st:
            tr.write('test_data/'+tr.stats.station+'-'+tr.stats.channel+'\n'
                   '-'+str(tr.stats.starttime.year)+'\n'
                   '-'+str(tr.stats.starttime.month).zfill(2)+'\n'
                   '-'+str(tr.stats.starttime.day).zfill(2)+'\n'
                   '-processed.ms', format='MSEED')
f.close()

```


Core programs for the EQcorrscan project.

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

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 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 in 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.

All rights reserved.

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._find_detections (cum_net_resp, nodes, threshold, thresh_type, samp_rate, re-
alstations)`

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

Parameters

- **cum_net_resp** (*np.array*) – Array of cumulative network response for nodes
- **nodes** (*list of tuples*) – Nodes associated with the source of energy in the cum_net_resp

Returns detections as class DETECTION

`bright_lights._node_loop (stations, lags, stream, i=0)`

Internal function to allow for parallelisation of brightness

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

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

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

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] refers to stations[i] and nodes[i][:][0] is latitude in degrees, nodes[i][:][1] is longitude in degrees, nodes[i][:][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] refers to stations[i] and nodes[i][:][0] is latitude in degrees, nodes[i][:][1] is longitude in degrees, nodes[i][:][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)

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] refers to stations[i] and nodes[i][:][0] is latitude in degrees, nodes[i][:][1] is longitude in degrees, nodes[i][:][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 Mean Absolute Deviation and abs is an absolute brightness.
- **coherence_thresh** (*float*) – Threshold for removing incoherent peaks in the network response, those below this will not be used as templates.

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

bright_lights.**coherence**(stream)

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.

Returns float - coherence

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 waveform through 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 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.

All rights reserved.

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)`

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 extract templates from
- **length** (*float*) – Length of template in seconds
- **swin** (*string*) – P, S or all

`template_gen.from_contbase(sfile)`

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).

`template_gen.from_sfile(sfile)`

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.

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 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 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.

All rights reserved.

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, type-of-det, 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 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.
- **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*)

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*(mean(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.

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-correlation. Documentation on this function can be found here:

- http://docs.opencv.org/modules/imgproc/doc/object_detection.html?highlight=matchtemplate#cv2.matchTemplate

Parameters **image** – Requires two numpy arrays, the template and the 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.

3.4 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.

All rights reserved.

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-obspace - 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 **i** (*int, optional*) – Used to track which process has occurred when running in parallel

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

`lag_calc.day_loop` (*detections, 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

- **detections** (*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

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

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.

All rights reserved.

```
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.blankinfile (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

```
Sfile_util.readpicks (sfilename)
```

Function to read pick informaiton from the s-file

Returns Sfile_tile.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.

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_peaks(arr, thresh, trig_int)`

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

EXPERIMENTAL CODE, DOES NOT WORK EFFICIENTLY, RECOMEND USING `find_peaks2`

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.find_peaks2(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. Improvement on `find_peaks`, speed-up of approx 250x.

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.

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

`clustering.SVD_2_stream_testing(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** (*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

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

Note

IN ALPHA, not working as expected

`clustering.SVD_tetsing` (*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(ndarray), SValues(ndarray) for each channel, stachans, List of String (station.channel)

Note

IN ALPHA, not working as expected

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.cluster` (*templates, show=True*)

Function to take a set of templates and cluster them, will return clustered templates

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

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

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

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

Returns ndarray - distance matrix

`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

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).

`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.

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.

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.

Returns `obspy.Stream`

4.5 EQcorrscan_plotting

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

`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.cumulative_detections` (*dates, template_names, save=False, save-file=''*)

Simple plotting function to take a list of UTCDateTime 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

`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.

`EQcorrscan_plotting.detection_timeseries` (*stream, detector, detections*)

Function to plot the data and detector with detections labelled in red, will downsample if too many data points.

Parameters **detections** (*np.array*) – array of positions of detections in samples

`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.

`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)

`EQcorrscan_plotting.threeD_seisplot` (*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, 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

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].

`mag_calc.Amp_pick_sfile` (*sfile, datapath, respdir, chans=['Z'], var_wintype=True, winlen=0.9, pre_pick=1.0, 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 1.0
- **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.

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** – 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 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*

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)`

Function to compute the phase weighted stack of a series of streams

Parameters `weight` (*float*) – Exponent to the phase stack used for weighting.

Returns `obspy.Stream`

`stacking.linstack (streams)`

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

Returns `stack - Stream`

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.

`catalogue2DD._av_weight (W1, W2)`

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

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

Returns `string`

`catalogue2DD.readSTATION0 (path)`

Function to read the STATION0.HYP file on the path given.

Parameters `path` (*String*) – Path to the STATION0.HYP file

Returns List of tuples of station, lat, long, elevation

`catalogue2DD.write_catalogue (event_list)`

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)

`catalogue2DD.write_correlations (event_list, wavbase, extract_len, pre_pick, shift_len, low-cut=1.0, highcut=10.0)`

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

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

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

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.

b

bright_lights, 11

c

catalogue2DD, 25

clustering, 20

e

EQcorrscan_plotting, 22

f

findpeaks, 20

l

lag_calc, 17

m

mag_calc, 23

match_filter, 15

match_filter_par, 27

p

pre_processing, 21

s

Sfile_util, 19

stacking, 24

t

template_gen, 13

template_gen_par, 27

Symbols

[_GSE2_PAZ_read\(\)](#) (in module [mag_calc](#)), 24
[_av_weight\(\)](#) (in module [catalogue2DD](#)), 25
[_cc_round\(\)](#) (in module [catalogue2DD](#)), 25
[_channel_loop\(\)](#) (in module [lag_calc](#)), 17
[_channel_loop\(\)](#) (in module [match_filter](#)), 16
[_check_daylong\(\)](#) (in module [pre_processing](#)), 21
[_find_detections\(\)](#) (in module [bright_lights](#)), 11
[_find_resp\(\)](#) (in module [mag_calc](#)), 24
[_max_p2t\(\)](#) (in module [mag_calc](#)), 24
[_node_loop\(\)](#) (in module [bright_lights](#)), 12
[_read_tt\(\)](#) (in module [bright_lights](#)), 12
[_resample_grid\(\)](#) (in module [bright_lights](#)), 12
[_rm_similarlags\(\)](#) (in module [bright_lights](#)), 12
[_sim_WA\(\)](#) (in module [mag_calc](#)), 24
[_template_gen\(\)](#) (in module [template_gen](#)), 14
[_template_loop\(\)](#) (in module [match_filter](#)), 16

A

[Amp_pick_sfile\(\)](#) (in module [mag_calc](#)), 23

B

[blanksfile\(\)](#) (in module [Sfile_util](#)), 19
[bright_lights](#) (module), 11
[brightness\(\)](#) (in module [bright_lights](#)), 13

C

[catalogue2DD](#) (module), 25
[cluster\(\)](#) (in module [clustering](#)), 21
[clustering](#) (module), 20
[coherence\(\)](#) (in module [bright_lights](#)), 13
[cross_chan_coherence\(\)](#) (in module [clustering](#)), 21
[cumulative_detections\(\)](#) (in module [EQ-corrscan_plotting](#)), 22

D

[day_loop\(\)](#) (in module [lag_calc](#)), 17
[dayproc\(\)](#) (in module [pre_processing](#)), 21
[DETECTION](#) (class in [match_filter](#)), 15
[detection_multiplot\(\)](#) (in module [EQcorrscan_plotting](#)), 22
[detection_timeseries\(\)](#) (in module [EQ-corrscan_plotting](#)), 22
[distance_matrix\(\)](#) (in module [clustering](#)), 21

E

[EQcorrscan_plotting](#) (module), 22

F

[find_peaks\(\)](#) (in module [findpeaks](#)), 20
[find_peaks2\(\)](#) (in module [findpeaks](#)), 20
[findpeaks](#) (module), 20
[from_contbase\(\)](#) (in module [template_gen](#)), 14
[from_sfile\(\)](#) (in module [template_gen](#)), 14

G

[group_delays\(\)](#) (in module [clustering](#)), 21

L

[lag_calc](#) (module), 17
[lag_calc\(\)](#) (in module [lag_calc](#)), 18
[linstack\(\)](#) (in module [stacking](#)), 25

M

[mag_calc](#) (module), 23
[match_filter](#) (module), 15
[match_filter\(\)](#) (in module [match_filter](#)), 16
[match_filter_par](#) (module), 27

N

[Noise_plotting\(\)](#) (in module [EQcorrscan_plotting](#)), 22
[normxcorr2\(\)](#) (in module [match_filter](#)), 16

P

[peaks_plot\(\)](#) (in module [EQcorrscan_plotting](#)), 22
[PICK](#) (class in [Sfile_util](#)), 19
[populateSfile\(\)](#) (in module [Sfile_util](#)), 19
[pre_processing](#) (module), 21
[PWS_stack\(\)](#) (in module [stacking](#)), 25

R

[readpicks\(\)](#) (in module [Sfile_util](#)), 19
[readSTATION0\(\)](#) (in module [catalogue2DD](#)), 25
[readwavename\(\)](#) (in module [Sfile_util](#)), 19

S

[Sfile_util](#) (module), 19
[shortproc\(\)](#) (in module [pre_processing](#)), 22
[stacking](#) (module), 24
[SVD_2_stream_testing\(\)](#) (in module [clustering](#)), 20

SVD_tetsing() (in module clustering), [21](#)

T

template_gen (module), [13](#)

template_gen_par (module), [27](#)

threeD_gridplot() (in module EQcorrscan_plotting), [23](#)

threeD_seismpoint() (in module EQcorrscan_plotting),
[23](#)

triple_plot() (in module EQcorrscan_plotting), [23](#)

W

write_catalogue() (in module catalogue2DD), [25](#)

write_correlations() (in module catalogue2DD), [25](#)

write_event() (in module catalogue2DD), [26](#)