Introduction to Detex

By Derrick Chambers

Detex is a python package for waveform clustering, waveform correlation detection, and subspace detection. This document will attempt to walk you through some of the basic functionality of the package, keep in mind as of 3/23/15 detex is still very much in the alpha phase (there are still plenty of bugs and functionality not yet implemented). If you have any issues, find any bugs, have suggestions for future implementation, or want to help contribute to detex email me at

[derrick.ja.chambers@gmail.com](mailto:derrick.ja.chambers@gmail.com)

Basic functionality of detex has been tested on Windows, Linux and OS X.

Special thanks to Tex Kubacki whose work and name provided the inspiration for de”tex” and to Jared Stein who has been the most active alpha tester.

# Python installation

(Skip this section if you are already comfortable with python and have it set up on your computer)

Google “download python anaconda”

Go to the first hit (<http://continuum.io/downloads>) and select the appropriate distribution based on your OS. Follow the simple installation procedure. Choose python 2.7, I am not sure if the code works with 3.

Go to the terminal and type the following commands (note: “<” just means I am typing code, you don’t need to type “<”)

< easy\_install obspy

< easy\_install basemap

<easy\_install joblib

<conda update pandas

<conda update spyder

Great, you are good to go. I recommend you spend 30 minutes or so playing around with python, especially the IDE spyder (you can open it by typing “spyder” into the terminal or command line). Here as some quick and snappy tutorials to get you going if you have never used python before:

<http://www.stavros.io/tutorials/python/>

<http://www.tutorialspoint.com/python/python_quick_guide.htm>

Above all, I strongly recommend you familiarize yourself with the geophysical package obspy

<http://docs.obspy.org/tutorial/>

In my opinion it is the best thing to happen to seismology since Charles Richter.

Run the script setup.py from the command line (terminal) by changing your current directory to wherever the file you just downloaded is and typing

< python setup.py install

Now open spyder and type into the ipython command line (bottom left)

< import detex

If it doesn’t throw any errors you are probably good to go.

If you want to check the current version of detex that you have you can type

< detex.version.version

As of version 0.0.1 Detex does not need to compile any code so if the setup.py fails you can simply place the detex directory (inside GEO6920\_SE) into your site packages under the anaconda distribution. For me the path to the site packages is C:\Anaconda\Lib\site-packages, but this may be different depending on how you installed anaconda and your OS.

# Required Files

Let us refer to a particular area/set of events you are investigating as a job. I would strongly recommend you create a directory for each job in which you can store files and perform operations. For this tutorial let’s create a new directory called “blasts” and place it somewhere convenient to work with. Mining related seismicity is so often disregarded by geophysicists that, in an attempt to be more accepting, we will study surface mining blasts in south west Wyoming for this tutorial.

Copy the files TemplateKey.csv and StationKey.csv into the newly created directory.

Let’s examine each of this files as they are important.

StationKey.csv is a file with all the information detex needs about the stations you are using. The table looks like this:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| NETWORK | STATION | STARTTIME | ENDTIME | LAT | LON | ELEVATION | CHANNELS |
| TA | M18A | 2007-11-13T00:00:00 | 2009-05-04T23:59:59 | 41.4272 | -110.067 | 2103 | BHE-BHN-BHZ |
| TA | M17A | 2007-11-13T00:00:00 | 2009-05-04T23:59:59 | 41.4729 | -110.666 | 2101 | BHE-BHN-BHZ |

Most if the columns should be easily understood. STARTTIME and ENDTIME can be in any format that the obspy.core.UTCDateTime object can understand (such as a timestamp), and therefore do not have to be the format shown above. Channels must be all the channels you wish to use from the station with a “-“ used as a separating character, i.e. BHE-BHN-BHZ.

TemplateKey.csv contains information about all the events you wish to use. The first few lines should look like this:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CONTRIBUTOR | NAME | TIME | LAT | LON | DEPTH | MTYPE | MAG | STATIONKEY |
| ANF | 2009-04-01T17-36-58 | 2009-04-01T17-36-58 | 41.6824 | -110.631 | 2.0984 | ML | 2.52 | StationKey.csv |
| ANF | 2009-04-03T15-39-27 | 2009-04-03T15-39-27 | 41.814 | -110.644 | 1.8643 | ML | 2.34 | StationKey.csv |
| ANF | 2009-04-06T18-53-12 | 2009-04-06T18-53-12 | 41.8021 | -110.632 | 1.9012 | ML | 2.4 | StationKey.csv |
| ANF | 2009-04-10T19-27-22 | 2009-04-10T19-27-22 | 41.7018 | -110.625 | 8.7868 | ML | 2.27 | StationKey.csv |

The NAME field doesn’t have to be a time in this format, but in fact can be anything you want as long as it is unique (two events in the template key should not share a name). In this case the symbol “-“ is used to separate the hour minute seconds rather than the “:” character because the NAME field is used to save the file most OS won’t allow a “:” in the file name.

Like before, the TIME field can be in any obspy.core.UTCDateTime readable format. The STATIONKEY field points to which station key should be used in processing any given event. In theory you could use different station keys for different events but I have not yet tested this feature.

The CONTRIBUTOR, and MTYPE fields are optional but they can be useful in keeping track of the events. You can add any other fields you want to the file, and the fields can occur in any order, but the required fields must be present.

With StationKey.csv and TemplateKey.csv in our job folder (blasts) we are ready to use IRIS to get the event waveforms and continuous data.

# Getting the Data

The detex module getdata is used to retrieve continuous data from the IRIS web service. It acts in real time and doesn’t fill up your email inbox with garbage as can other overly prescribed methods.

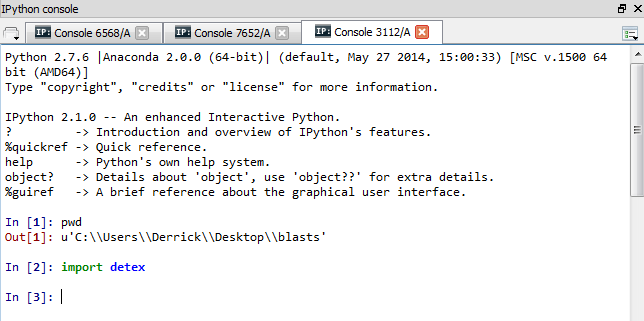
Let’s open Spyder and use its directory navigation bar to find our job directory. The navigation bar can be found in the upper right corner.



Click the folder button, find “blasts” and click open, then click the bottom next to the open folder button to set it as our current directory. Then type pwd in the ipython terminal and make sure it worked. Next type

< import detex

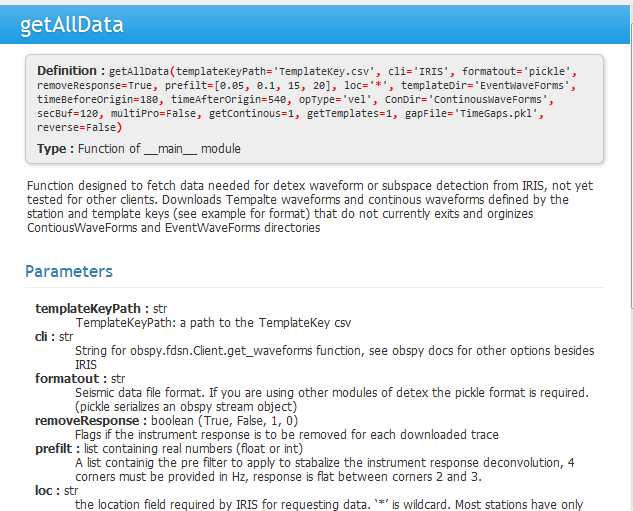
Again, if no errors are thrown we are good to go.



Let’s look at the documentation for the getalldata function in the getdata module. This can be done by typing

<detex.getdata.getalldata

Into the ipython console. With the curser at the end of the statement (right after the “a” of “data”) hit control and the “I” key for inspect. It should load the documentation in the object inspector window. It looks like this:



I am not going to describe all the parameters and what they do here, but will simply refer you to the doc string. One thing to note, however, is that in python a function has required parameters and optional parameters that already have some default value. In the definition the required parameters are not already assigned a default value, IE they don’t have an “=” next to them. You must supply the function an argument or it will throw an error. getAllData does not have any parameters you are required to set, but you can change the default value by reassigning it upon calling the function. For example, if I did not want to remove the instrument responses before saving the downloaded data I could do something like this:

<detex.getdata.getAllData(removeResponse=False)

Which would override the default value for removeResponse (True) and set it to False. Note: there is also a useful way to pass an arbitrary number of parameters to a function using the \*args or \*\*Kwargs methods but I don’t use this much in detex.

For our example the defaults here should be fine. Type:

< %time detex.getdata.getAllData()

The %time before any command will time how long it takes to execute (only works if you are using an Ipython console). For every 25 events download detex will print to screen the progress it has made. There are several things I need to do to speed up the download process but have not yet completed so it may be a bit slow. Mine took about 7 minutes to complete.

Now look in the job directory (“blasts”). You can see detex created two new directories, ContinousWaveForms and EventWaveForms. You can click through them to see how they are set up.

# Cross Correlation

Let’s learn how to do cross correlations before moving on to more complicated subspace methods. We will use two detex modules to do this, first detex.xcorr to run the correlations, and second detex.results to interpret the results of the correlations and associate detections across stations to form coherent events.

Currently in our template key we have 80ish events. Make a copy of the TemplateKey.csv called TemplateKey\_All.csv. Now open the original and delete all but the first 4 events (running all 81 events over 3 days of continuous data is do-able but let’s just stick to 4 for this tutorial).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CONTRIBUTOR | NAME | TIME | LAT | LON | DEPTH | MTYPE | MAG | STATIONKEY |
| ANF | 2009-04-01T17-36-58 | 2009-04-01T17-36-58 | 41.6824 | -110.631 | 2.0984 | ML | 2.52 | StationKey.csv |
| ANF | 2009-04-03T15-39-27 | 2009-04-03T15-39-27 | 41.814 | -110.644 | 1.8643 | ML | 2.34 | StationKey.csv |
| ANF | 2009-04-06T18-53-12 | 2009-04-06T18-53-12 | 41.8021 | -110.632 | 1.9012 | ML | 2.4 | StationKey.csv |
| ANF | 2009-04-10T19-27-22 | 2009-04-10T19-27-22 | 41.7018 | -110.625 | 8.7868 | ML | 2.27 | StationKey.csv |

## Trim Templates

Next, we have to define the template start times. To do this we will use a light PyQT4 GUI modified slightly from the one here (<https://github.com/megies/obspyck/wiki>, much thanks to the authors).

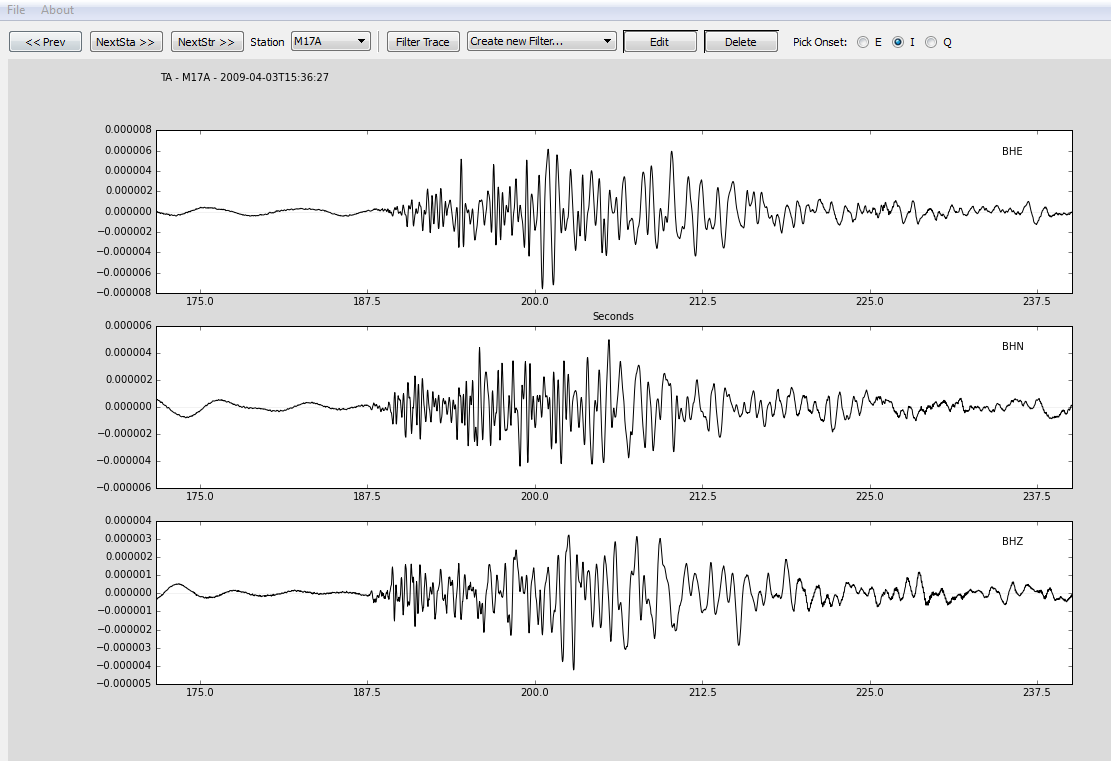
While in the blasts directory type

<detex.util.trimTemplates

And again hit control + I to look at the docs. Once you have read what the parameters are call the function by closing the brackets

< detex.util.trimTemplates()

It should launch a GUI to help you pick phases. If this does not work you are in trouble because I have not yet implemented non-manual picking methods. It should look like this:



Click on the about menu at the top to see what all the keys do. You can zoom in and out using the mouse wheel, hold the mouse wheel and drag for horizontal panning.

Make a pick by holding down the Q key and clicking on one of the waveforms. This should create a vertical line labeled with the P phase. Now hold down r and click, this should remove the phase pick. Zoom in and pick the P phase, which in detex is used for the start time of the waveform. You only need to do this on one of the channels. Now click on the “NextStr” button to pick the next event start time or hit the V key. If you click the X to close the window before picking all the events your progress up to that point will be saved as EventPicks.pkl by default.

## Estimate False Alarm Statistic

Next, we will try to estimate the distribution of the null space (The distribution of each template with the seismic data that has no signals in it).

In order to do this we will us the detex.xcorr.getFAS function which is a wrapper for the FAS class.

<detex.xcorr.getFAS(5)

See the docs for more on this. Now the file ‘FAS.pkl’ should exist in the current directory.

Currently I do not have any methods to allow you to easily look at the distributions and the fitted PDFs (both a normal and a beta are calculated). Until I create such methods I will leave it to you as an exercise to lean pandas, numpy, and scipy better. The results are stored in a pandas data frame as dictionaries with the templates being the indices and the stations being the columns.

## Run Correlations

Once we have created our FAS.pkl file we are ready to run the correlations. We do not actually need to use the FAS.pkl, as we can use a set correlation coefficient value, or an STA/LTA of the correlation coefficient (as recommended in Gibbons and Ringdal 2006) to declare detections. Since we only have a few days of data the false detection statistics as calculated by the getFAS method may or may not be very meaningful so let’s not use them. Rather, let’s use the STA/LTA method.

< %time detex.xcorr.correlate(trigCon=1,trigParameter=6)

Alternatively, if we were feeling lazy, we could simply enter

< %time detex.xcorr.correlate(1,6)

Because trigCon is the first input parameter, and trigParameter is the second python interprets this correctly.

trigCon is the parameter that describes the trigger condition. It is explained in detail in the doc string. trigCon = 1 means we are using the STA/LTA method with a required STA/LTA ratio of 5.5.

On my machine (i7 4 core 3.5 gHz) these correlations take about 1 minute, not too bad for running 3 days of 40Hz continuous data for 4 templates on 2 two stations. Detex tells us it found 163 detections for TA.M18A and 148 detections for TA.M17A. Since our templates are blasts that occur at most several times a day, nearly all of these must be false detections (or other transient signals). Let’s look at the results.

## Simple Viewing Method

The results are stored in a SQLite database by default named Corrs.db. We can load it into a data frame for viewing using the detex utility detex.util.loadSQLite.

< df = detex.util.loadSQLite(“Corrs.db”,”cor\_df”)

“cor\_df” is the table of the database we are loading that contains the detection information. There is also a sql parameter on the loadSQLite function that allows the user to pass sql code as a string to filter the data in the table. We won’t worry about this for now.

Now if we type

< df

The contents of the dataframe should be printed to screen. It looks like perhaps our STA/LTA requirement was too low because some triggers have negative correlation coefficient values. We can examine the relationship between the two by plotting the STA/LTA vs the correlation coefficient:

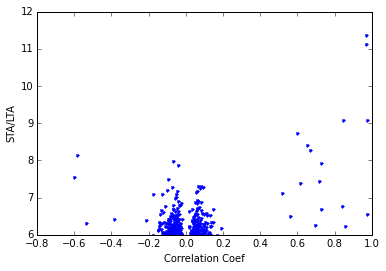
< import matplotlib.pyplot as plt

< df.sort(columns='Coef',inplace=True)

< plt.plot(df.Coef,df.STA\_LTACoef,’.’)

<plt.xlabel(‘Correlation Coef’)

<plt.ylabel(‘STA/LTA’)



Most of the points are centered on 0 between -.2 and .2. This indicates we may need to play around with the STA/LTA parameters to better tune for these type of events, or perhaps the STA/LTA subroutine needs some more work (very possible). In this case perhaps we would do better to use a hard correlation coefficient requirement above .4? Let’s take 1 minute and run it again.

<%time detex.xcorr.correlate(0 , 0.4)

And when it finishes type:

< df = detex.util.loadSQLite('Corrs.db','cor\_df')

Now you can see we have 22 detections all with correlation coefficients above 0.4. Let’s look at some more complicated (sophisticated) ways of analyzing our data.

# Cross Correlation Results

We will use the results module to associate detections together as coherent events across the network, and generate inputs to location programs like hypoDD or hypoInverse.

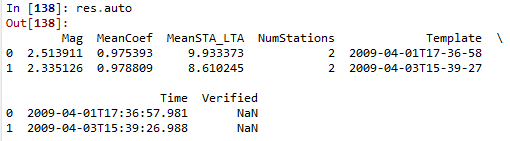
Once we are in our blasts directory with the template key, station key, and correlation database we can use:

< res = detex.results.corrResults(requiredNumStations=2)

Normally the required number of stations should be 4 or more (in order to use other location methods) but since we only used 2 stations it is the best we can do.

An instance of the CorResults class is now stored as the res variable. Here are a few useful attributes:

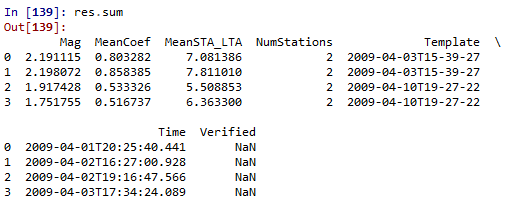
< res.auto



Displays all the grouped detections that represent any of the template events (Auto correlations). Note the Mag listed here are scaled magnitudes and are very close to the original magnitudes. Also, the MeanCoef field is the mean correlation coefficient which is slightly less than 1. ~~This loss of precision is probably due to some methods I used to speed up the process and I will see if it cannot be improved in the future.~~  (Fixed on 2/12/15)

We can see the detected events that are not template events by typing

< res.sum



Now if we look at the blasting log the mine provided us for April 1st to April 3rd we can see our templates detected all the blasts that occurred in our continuous data range with no false detections:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **shotnumber** | **date** | **time** | **UTCTime** | **maxweightdetonated** |
| 9107 | 4/1/2009 | 11:35:00 | 17:35:00 | 2943 |
| 9108 | 4/1/2009 | 14:22:00 | 20:22:00 | 1713 |
| 9109 | 4/2/2009 | 10:24:00 | 16:24:00 | 1542 |
| 9110 | 4/2/2009 | 13:20:00 | 19:20:00 | 812 |
| 9111 | 4/3/2009 | 9:36:00 | 15:36:00 | 2233 |
| 9112 | 4/3/2009 | 11:30:00 | 17:30:00 | 1022 |

Shots with red UTCTimes were templates and the shots with blue UTCTimes were detections.

<res.writePhaseDD

And

< res.writePhaseHyp

Are the function to write the phase files for hypoDD and hypoInverse. See the documentation of these functions for more details (they are rather barebones currently and could probably use some more testing and features).

# Subspace

Detex.subspace is the module that handles the subspace methods. The methodology follows Harris 2006 very closely with a few exceptions. In order to start lets rename the current template key (the one that only has four events in it) to TemplateKey\_old.csv and rename TemplateKey\_all.csv to TemplateKey.csv so it will be the file used by default.

## Clustering

The first step is to perform waveform clustering on all the events in the template key which now constitutes our waveform pool. It is important to know that a single link algorithm is used, which allows waveforms of significant variation to be grouped together provided there are intermediary events that bridge the extremes. This is advantages when accounting for event source/location migration. The clustering algorithm is essentially the same as the one described in Harris 2006 Appendix B.

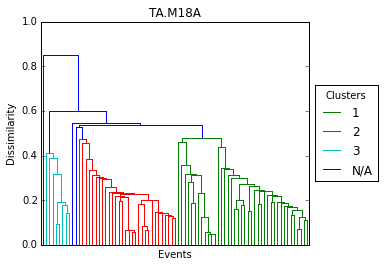
<import detex

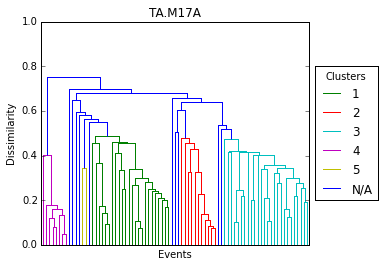
< %time cl = detex.subspace.createCluster()

This will create the cluster object, assign it to the variable cl, and serialize the cluster object in the current directory. Note that the parameter masterStation can be used to force the same events from the cluster of one station on all other stations when creating a subspace. This often works, but sometimes can cause problems so I recommend you allow each station to for its own clusters by leaving the masterStation parameter as None. Some of the important methods of the cluster objects:

< cl.dendro()

Creates dendrograms of the clusters by station





If we decided that the default correlation coefficient of 0.5 was too low we could reassign it for all the stations by using the cl.updateReqCC method.

<cl.updateReqCC(0.55)

This will create 4 nice clusters on station M18A, but there may be too many small clusters on M17A now. Let’s change the correlation coefficient requirement for station M17A so 4 nice clusters also form.

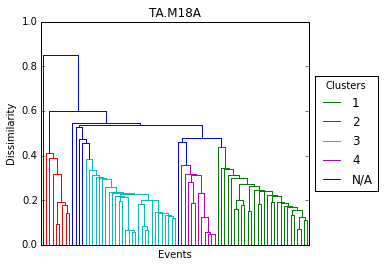
< cl['M17A'].updateReqCC(.39)

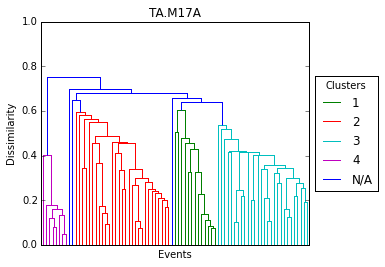
< cl['TA.M17A'].updateReqCC(.39)

Or

< cl[1].updateReqCC(.39)

Would all work. Now when we look at the dendrograms:





Each station now has 4 clusters. Differing the required correlation coefficient for clustering by station may or may not be desirable, depending on the data set and objective.

If you wanted to access the names of the events in each of these groups they can be reached by the cl.clusts attribute which is a list of names pertaining to the clusters in the dendrogram. For example, the names of all events in the green cluster on station M18A (labeled 1 in the key above) can be seen by displaying the list

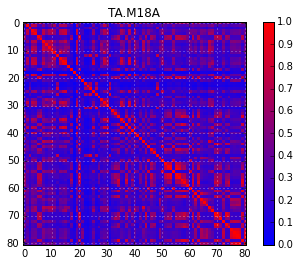
<cl[‘M18A’].clusts[0]

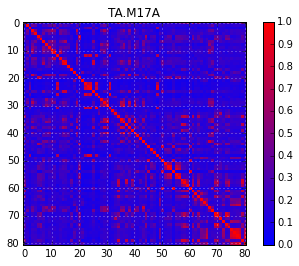
I choose to label cluster 0 as cluster 1 in the dendrogram key because a numbering system that starts with zero takes a real physiological toll on some people.

The events that did not cluster on station M18A (labeled N/A) can be accessed via cl[‘M18A’].singles.

We can also create a basic similarity matrices

<cl.simMatrix()





With the event numbers listed along the x and y axis. To link event numbers with event names look at cl[station].key which is a list of the event names (here station=’M17A’ or ‘M18A’). The corresponding index of the list is the event number. For example, typing

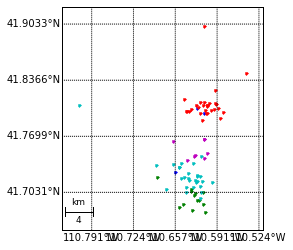
< cl[‘M17A’].key[0]

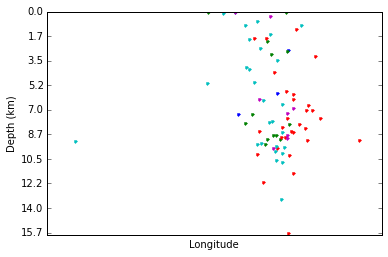
would display the name of the 0th event.

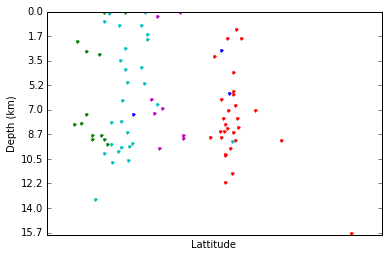
The last important method, which must be called at the station level is:

< cl[‘M17A’].plotEvents()

Which calls the dendrogram function but also creates a basic map using the locations in the template key to display spatial relations between clusters. This function still needs some work to make the plots more presentable but functions to convey spatial relationships between clusters.







Let’s see in which cluster the templates we used in waveform correlation ended up in. We could load the names of the event into a list and perform the following operation

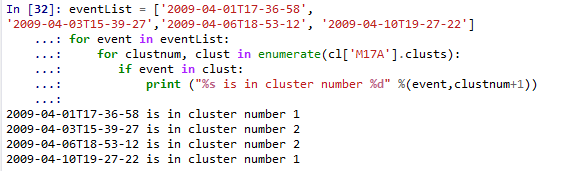
< eventList = [“2009-04-01T17-36-58”, “2009-04-03T15-39-27”, “2009-04-06T18-53-12”, “2009-04-10T19-27-22”]

< for event in eventList:

< for clustnum, clust in enumerate(cl[‘M17A’].clusts):

< if event in clust:

< print (“ %s is in cluster number %d” %(event,clustnum+1))



Next we will move on to subspace detection which uses the results of createClusters to form subspaces for detection purposes.

## Subspace Detection

Since the clustering object already has information regarding paths to station keys, template keys, and event directory location we now only need relatively few parameters to get started in creating the subspaces. In fact, we can leave the default parameters and create the subspace object in the same directory we used before because detex knows to look for the saved cluster object if the default parameters were used.

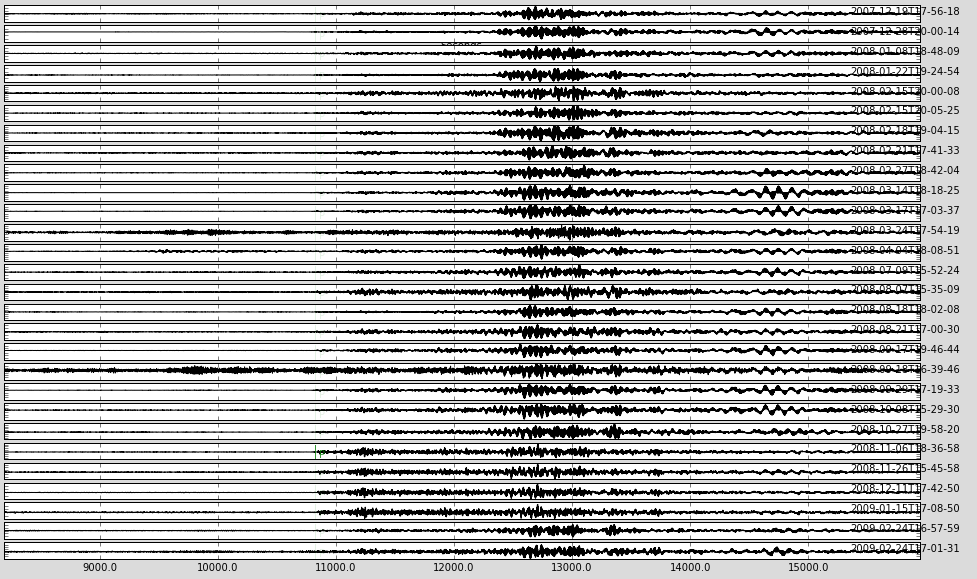
< %time ss = detex.subspace.createSubSpace()

See the docs for the available parameters for this function.

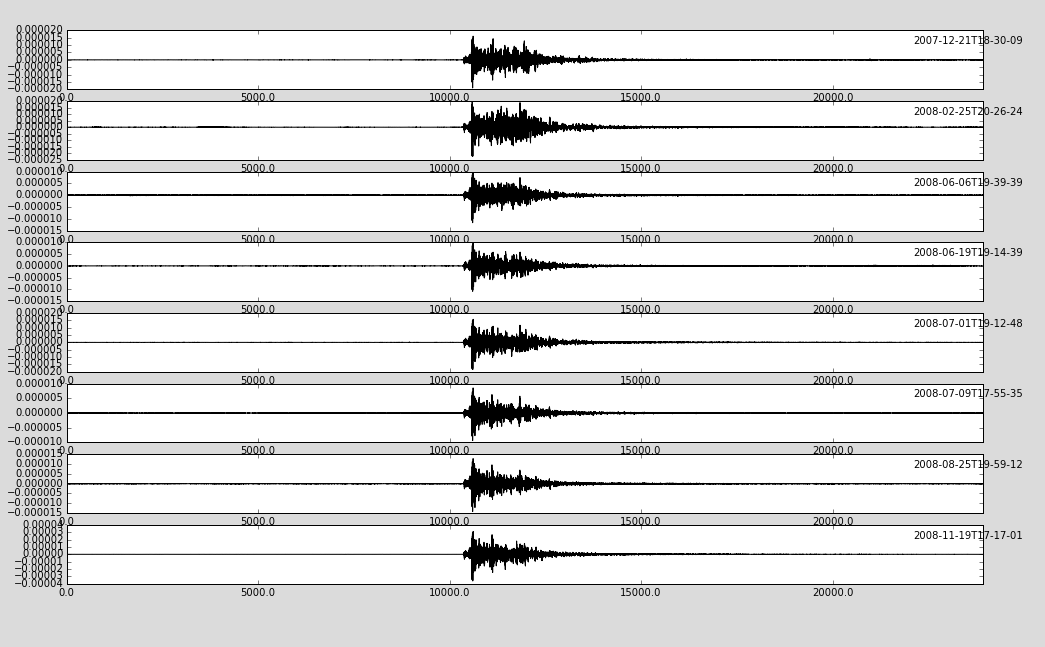
The creatSubSpace call has not yet done all the work for us, it has only read in all the events and aligned the waveforms. Before we perform the SVD to get the basis vectors we need to trim the waveforms to avoid using excessively long waveforms with a lot of noise before and after the events. With 80+ events, as are in our group, we wouldn’t want to face the tedium of picking each event by hand so detex lets us act on the entirety of each cluster (because the waveforms have been aligned). We will have to do this for each subspace/station pair.

< ss.pickSubSpaceTimes()

The default method is to define as single pick time and then use the duration value set on the pickSubSpace call to define the end time. You can also pass None to the duration parameter and pick a start and stop time for each event group but I have yet to test this.



We can see the same GUI used to trim templates is called. We are looking at the multiplexed waveforms so they may appear a little strange if we were to zoom in on them. Here we only have to pick one arrival on one of the waveforms to act as the start time for the whole group. With the larger groups sometimes significant variation in waveforms can be seen due to the single linking algorithm used to group the events together. The smaller groups generally look more similar:



After picking the start times of the waveforms it is a good idea to save the subspace object so we don’t have to re-pick if something happens.

< ss.write()

Now the file subspace.pkl should have been created. We can load it anytime by

< ss=detex.subspace.loadSubSpace()

The use of non-default filenames are detailed in the docs for both the write and load methods.

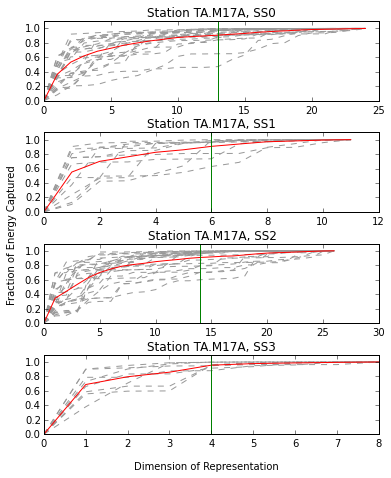
Once a pick time for each group has been made we can create the basis vectors and set our detection thresholds. The SVD function of the subspace object is used to carry out both of these tasks. There are several statistical methods for setting thresholds available, but generally they are not very meaningful because the tails of the distribution don’t fit well due to transient signals that may or may not be significant.

The docs outline various less statistically rigorous (and resource taxing) methods, one of which is method 3, which sets the thresholds based on some fraction of the minimum captured energy. In order to make a fair comparison with the waveform correlation detector (approximately) we should fix the detection threshold to 0.25

< ss.SVD(3,0.9, Threshold=0.25)

Now our number of basis vectors and detection statistic thresholds have been set for each station-subspace pair. The number of basis vectors kept from the SVD is determined such that the fraction of 0.9 of the total template event energy is captured by the truncated basis. We can see how well newly formed basis captures the energy for each of the events as a function of representation dimension by calling the plotFracEnergy function of the subspace object

< ss.plotFracEnergy()

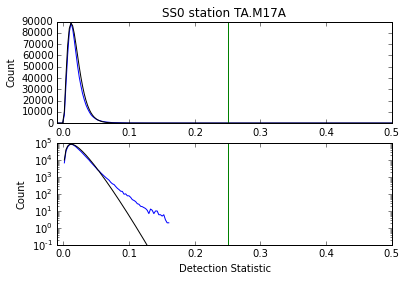


The plot shows the average fractional energy capture of all the subspaces/station pairs as a function of dimension of representation. The grey dotted lines are the fractional energy for each of the events in the waveform pool, the red line is the average and the green vertical line shows how many of the basis vectors the SVD function selected based on the input criteria (the determined dimension of rep.).

Now let’s try to get an idea of how well our set detection threshold of 0.25 will work.

< ss.plotThresholds(5)

The 5 is the number of continuous data files to use to generate a histogram of the detection statistic. In practice you will probably want to use a larger number. The following plots are produced for each subspace-station pair:



The blue line is a plotted histogram of the detection statistic, the green line is the detection statistic threshold, and the black line is a beta distribution fit to the detection statistic. The upper plot is a linear scale and the lower is a log scale. In the log scale a significant variation from the fitted beta distribution is apparent in the tail.

If we wanted to experiment with different methods and try using longer amounts of continuous data to characterize the distributions better we could do so, but let’s move on to the actual subspace detection.

Since we already took care of setting the detection statistic threshold we can simply call the detex function of the subspace stream and leave the default parameters.

< %time ss.detex()

On my machine this takes about 4 minutes. There are about 35 basis vectors used in the calculation of the detection statistic for each station compared to only 4 for the correlation detection performed earlier, which explains the increase in runtime. However, we are effectively scanning for about 80 events in the subspace operation whereas we were only able to scan for 4 with the waveform correlations so we sacrifice some runtime for increased detection capabilities. Let’s look at the results:

< res= detex.util.loadSQLite(‘SubSpace.db’,’ss\_df’)

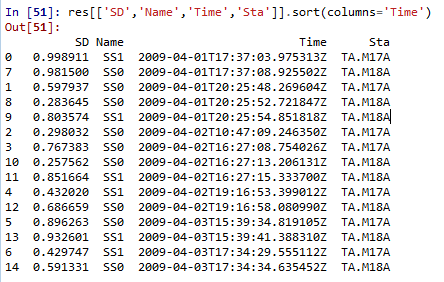
The time information is stored in the column STMP as a time stamp, which is not the most readable form. Let’s add a column in a more familiar format using a python list comprehension.

< import obspy

< res[‘Time’]=[obspy.core.UTCDateTime(float(x)) for x in res.STMP]

Now we will display only some of the columns and sort by time

< res[[‘SD’,’Name’,’Time’,’Sta’]].sort(columns=’Time’)



And looking back on the blast log:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **shotnumber** | **date** | **time** | **UTCTime** | **maxweightdetonated** |
| 9107 | 4/1/2009 | 11:35:00 | 17:35:00 | 2943 |
| 9108 | 4/1/2009 | 14:22:00 | 20:22:00 | 1713 |
| 9109 | 4/2/2009 | 10:24:00 | 16:24:00 | 1542 |
| 9110 | 4/2/2009 | 13:20:00 | 19:20:00 | 812 |
| 9111 | 4/3/2009 | 9:36:00 | 15:36:00 | 2233 |
| 9112 | 4/3/2009 | 11:30:00 | 17:30:00 | 1022 |

We can see that we detected all the blasts but also have one potentially false detection at 2009-04-02T10:47:08. The potential false detection, however, only occurs on one station on one subspace, so if we enforced some multi-station detection requirement we could effectively remove it.

# Subspace Results

< res=detex.results.ssResults(requiredNumStations=2) #

Here detex is associating events together based on a predicted travel time range from the original waveform trim time on each station and the reported origin time in the template key. requiredNumStations (the number of station a detection must occur on) is set to 2, because we only used 2 stations.

< res



Calls the representation method which tells us what the object is and how many autodections (detections also in the template key) and new detections where found.

< res.Autos

Can be used to access a dataframe of the autodetections and

< res.Dets

Can be used to access a dataframe of the new detections

If we have independent information which we want to use to verify the detections we can also do so by passing a path to the verification file to the ssResults class using the keyword verFile. In the distribution you should have a file called veriFile.csv. This is our verification info in the same form as the template key. It looks like so:



The required columns, which are also cap sensitive, are: NAME,TIME,MAG,LAT,LON, and DEPTH. I added the MWD column which is just the max weight detonated.

Now if we again call the ssResults method:

< res=detex.results.ssResults(requiredNumStations=2,veriFile='veriFile.csv')

< res



Now rather than N/A detex reports that 0 of our detections were verified. The problem is that the default time parameters for associating events in the verification file with the predicted origin time of the detections is based on predicted travel time ranges of the subspace, plus a default buffer of 1 second. The blasting times recorded were done on a wristwatch and required accuracy was only within 5 minutes or so, so the default conservative parameters will not work here. Let’s try allowing a buffer of 10 minutes, (5 before 5 after) instead.

< res=detex.results.ssResults(requiredNumStations=2,veriFile='veriFile.csv',veriBuffer=10\*60)

< res



Now we see that all 6 of the detections are verified. To access the verified data frame we can type

< res.Vers

References

------------------------

Gibbons, S. J. and F. Ringdal (2006), “The detection of low-magnitude seismic events using array-based waveform correlation”, Geophysical Journal International, 165, 149-166.

Slinkard, M., Schaff, D., Mikhailova, N., Heck, S., Young, C., & Richards, P. G. (2014). Multistation Validation of Waveform Correlation Techniques as Applied to Broad Regional Monitoring. *Bulletin of the Seismological Society of America*, *104*(6), 2768-2781.

Wiechecki-Vergara, S., H. L. Gray, W. A. Woodward (2001), Statistical Development in Support of CTBT Monitoring, DTRA report DTRA-TR-00-22, 8725 John J. Kingman Road, Mail Stop 6201, Fort Belvoir, VA 22060-6201.

Harris, D. B., & Paik, T. (2006). *Subspace detectors: efficient implementation* (p. 36). United States. Department of Energy.

Harris, D. B. (2006). *Subspace detectors: theory*. United States. Department of Energy.