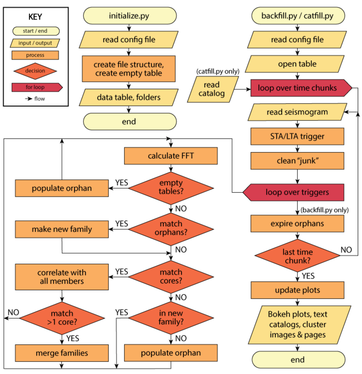
Documentation

Alicia Hotovec-Ellis edited this page on Apr 5, 2018 · 18 revisions

Edit | New page

Basic Logic



1. Data are downloaded and run through a sensitive STA/LTA triggering algorithm, then filtered for obviously bad triggers (moved to itable for testing if considered either as noise or a possible teleseism)
2. Each trigger is cross-correlated with any manually deleted events in dtable.
3. If it correlates, it's thrown out. If it doesn't, it is then cross-correlated with any events in otable.
4. If it correlates, it grabs any orphans it correlated with. If it doesn't, it is cross-correlated with 'core' events in rtable. The core is an automatically-selected representative waveform for each family. Family information is also stored in ftable.
5. If it correlates with any core event (at just below the final threshold), it is then further cross-correlated with the rest of the events in the family for each core matched/nearly matched. Any cross-correlations above the threshold are written to ctable, and the trigger and any orphans it adopted are written to rtable as long as it correlates above the threshold for at least one other event. If it did not correlate with any events in rtable but brought orphans with it, they are all appended to rtable as a new family. New families initiate OPTICS to find the core event for that family.
6. If it does not correlate with any cores or orphans, the trigger is appended to otable as an orphan and assigned an expiration date.
7. Repeat for all triggers.
8. Remove 'expired' orphans from otable.
9. Repeat for next data chunk.
10. Write outputs (including plots)

See Scripts and Helper Functions for descriptions of additional interactive functionality.

Saving to File: PyTables

PyTables is a framework for handling and storing large amounts of hierarchical data. Think of it like a database where all the information for REDPy is stored on the disk. Each `*.h5` file contains tables with different information. For example, each row in the otable or rtable corresponds to an individual event, and each column describes that event. One of the nice things about PyTables is the ability to store numpy arrays into cells, allowing us to easily store waveforms and Fourier transforms to save computation time. Be aware that the file size of a table can easily reach several GBs (for example, the table for all detected repeaters from 2009-2018 at Mount St. Helens is ~2GB, and Rainier is ~5GB).

Triggers Table

The ttable contains rows with a single column corresponding to the time of all triggers. This is used to make the histogram plot at the top of the overview pages, and to ensure events don't double trigger (i.e., running both catfill.py and backfill.py). For very long runs and periods of intense seismicity, this table can grow large.

Orphan Table

The otable contains rows corresponding to triggers that do not currently have any matches. Each orphan has a unique time where it will 'expire' and be removed from the system permanently, based on its signal to noise ratio. This allows REDPy to have a limited memory of the past, with a shorter memory for small events and longer memory for large events.

The columns contained in the orphan table are as follows (with type):

- `id` : unique ID number for the event (integer)
- `startTime` : UTC time of start of the waveform (string)
- `startTimeMPL` : matplotlib number associated with time (float)
- `waveform` : Waveform data (ndarray)
- `windowStart` : "trigger" time, in samples from start (integer)
- `windowCoeff` : amplitude scaling for cross-correlation (float)
- `windowFFT` : Fourier transform of window (complex ndarray)
- `windowAmp` : amplitude in first half of window (float)
- `expires` : UTC time of when orphan should no longer be considered (string)

Repeater Table

The rtable contains rows corresponding to triggers that correlate above the correlation threshold with at least one other row in the table. The columns are mostly similar to otable.

The columns contained in the repeater table are as follows (with type):

- `id` : unique ID number for the event (integer)
- `startTime` : UTC time of start of the waveform (string)
- `startTimeMPL` : matplotlib number associated with time (float)
- `waveform` : Waveform data (ndarray)
- `windowStart` : "trigger" time, in samples from start (integer)
- `windowCoeff` : amplitude scaling for cross-correlation (float)
- `windowFFT` : Fourier transform of window (complex ndarray)
- `windowAmp` : amplitude in first half of window (float)
- `alignedTo` : ID of event this one is aligned to (integer)

The repeater table also contains some header information with some of the important settings from the `*.cfg` file.

Correlation Table

The ctable contains all of the cross-correlation values between pairs of triggers. Pairs are only written to this table if the cross-correlation value is above the correlation threshold.

The columns are simply:

- `id1` : unique ID number for the first event (integer)
- `id2` : unique ID number for the second event (integer)
- `ccc` : cross-correlation coefficient between those two events (float)

Additionally, `id1` > `id2`.

Families Table

The ftable contains shortcut information for the members of each family and whether the family needs to be plotted.

The most important columns are:

- `members` : rows in rtable that contain members of the family as an ordered string (string)
- `core` : row corresponding to the 'core' event (int)
- `printme` : describes whether the family has been updated since last printing (int)

The members column is a string so that it can be of completely arbitrary length. Well, not completely arbitrary. The itemsize set here (1000000) is a guess at how big the string might get for really large families. One hopes to never encounter a family this big... (100000+ members?).

The header also contains a shortcut to the rows of all the cores (`ftable.attrs.cores`), as well as cores from the previous clustering (`ftable.attrs.prevcores`).

Deleted and Junk Tables

There are currently two other tables (dtable for manually deleted events, and itable for junk events). The dtable is identical to otable except without the expiration column. The itable is a holding table for bad triggers (i.e., noise and teleseisms) and is kept for testing.

Clustering and OPTICS

A cluster contains all events that correlate with at least one other event in the cluster above the correlation threshold, rather than requiring that all events correlate with each other above the threshold. This allows for more related events to be grouped together into one cluster. Additionally, clusters can combine if a new event correlates with an event in two or more clusters.

OPTICS is an acronym for '**O**rdering **P**oints **t**o **I**dentify the **C**lustering **S**tructure,' and is a density-based clustering algorithm. Points/events are ordered such that the closest events are nearest each other in the ordering, and distance is defined by cross-correlation coefficient. 'Reachability' is then used to determine the 'core' or representative event for the cluster. As the cluster adds more members the core event can change.

Note on Cross-Correlation

I'm *currently* not doing anything fancy with the cross-correlation step. It is done in a window around the trigger (10% before, 90% after) with a set window length (of some 2^N samples usually). Take the FFT of both triggers, multiply one by the conjugate of the other, take the IFFT of that, and the value is the resulting maximum after some scaling so that auto-correlation is 1. See the function `redpy.correlation.xcorr1x1()` for the exact code.

Add a custom footer

Add a custom sidebar

Clone this wiki locally

https://github.com/ahotovec/REI