HiPerSeis Documentation

Release Alpha

F. Zhang, A. Medlin, R. Hassan, A. Gorbatov, B. Hejrani

Jun 25, 2020

CONTENTS

CHAPTER

ONE

SUBJECT AREA API REFERENCE

1.1 seismic.ASDFdatabase package

1.1.1 Submodules

1.1.2 seismic.ASDFdatabase.ClientUtils module

class seismic.ASDFdatabase.ClientUtils.**Client2ASDF**(*client='IRIS'*, *network='AU'*)

Bases: object

queryByBBoxInterval(*outputFileName*, *bbox*, *timeinterval*, *chan='*Z'*, *bbpadding=2*, *event_id=None*, *verbose=False*)

Time interval is a tuple (starttime,endtime)

1.1.3 seismic.ASDFdatabase.FederatedASDFDataSet module

Description: Wrapper Class for providing fast access to data contained within a set of ASDF files

References:

CreationDate: 12/12/18 Developer: rakib.hassan@ga.gov.au

Revision History: LastUpdate: 12/12/18 RH LastUpdate: 2020-04-10 Fei Zhang clean up + added example run for the script

class seismic.ASDFdatabase.FederatedASDFDataSet.**FederatedASDFDataSet**(*asdf_source*, *logger=None*, *single_item_read_limit_in_mb=10*

Bases: object

get_closest_stations(*lon*, *lat*, *nn=1*)

Parameters

- **lon** – longitude (degree)
- **lat** – latitude (degrees)
- **nn** – number of closest stations to fetch

Returns A tuple containing a list of closest 'network.station' names and a list of distances (in ascending order) in kms

get_global_time_range(*network*, *station*, *location=None*, *channel=None*)

Parameters

- **network** – network code
- **station** – station code
- **location** – location code (optional)
- **channel** – channel code (optional)

Returns tuple containing min and max times as UTCDateTime objects. If no matching records are found min is set to 2100-01-01T00:00:00.000000Z and max is set to 1900-01-01T00:00:00.000000Z

get_stations (*starttime*, *endtime*, *network=None*, *station=None*, *location=None*, *channel=None*)

Parameters

- **starttime** – start time string in UTCDateTime format; can also be an instance of Obspy UTCDateTime
- **endtime** – end time string in UTCDateTime format; can also be an instance of Obspy UTCDateTime
- **network** – network code (optional)
- **station** – station code (optional)
- **location** – location code (optional)
- **channel** – channel code (optional)

Returns a list containing [net, sta, loc, cha, lon, lat] in each row

get_waveform_count (*network*, *station*, *location*, *channel*, *starttime*, *endtime*)

Count the number of traces within the given parameters of network, station, etc.. and date range. This is a fast method of determing whether any trace data exists in a given time period, if you don't actually need the waveform data itself.

Parameters

- **network** – network code
- **station** – station code
- **location** – location code
- **channel** – channel code
- **starttime** – start time string in UTCDateTime format; can also be an instance of Obspy UTCDateTime
- **endtime** – end time string in UTCDateTime format; can also be an instance of Obspy UTCDateTime

Returns The number of streams containing waveform data over the time-range provided

get_waveforms (*network*, *station*, *location*, *channel*, *starttime*, *endtime*, *trace_count_threshold=200*)

Parameters

- **network** – network code
- **station** – station code
- **location** – location code
- **channel** – channel code

- **starttime** – start time string in UTCDateTime format; can also be an instance of Obspy UTCDateTime
- **endtime** – end time string in UTCDateTime format; can also be an instance of Obspy UTCDateTime
- **trace_count_threshold** – returns an empty Stream if the number of traces within the time-range provided exceeds the threshold (default 200). This is particularly useful for filtering out data from bad stations, e.g. those from the AU.Schools network

Returns an Obspy Stream containing waveform data over the time-rage provided

local_net_sta_list()

This function provides an iterator over the entire data volume contained in all the ASDF files listed in the text file during instantiation. When FederatedASDFDataSet is instantiated in an MPI-parallel environment, meta-data for the entire data volume are equally partitioned over all processors – in such instances, this function provides an iterator over the data allocated to a given processor. This functionality underpins parallel operations, e.g. picking arrivals.

Returns tuples containing [net, sta, start_time, end_time]; start- and end-times are instances of Obspy UTCDateTime

property unique_coordinates

Returns dictionary containing [lon, lat] coordinates indexed by 'net.sta'

1.1.4 seismic.ASDFdatabase.asdf2mseed module

Description: Small utility for exporting mseed files from an asdf file in parallel.

References:

CreationDate: 06/08/18 Developer: rakib.hassan@ga.gov.au

Revision History: LastUpdate: 04/12/17 RH LastUpdate: dd/mm/yyyy Who Optional description

seismic.ASDFdatabase.asdf2mseed.**dump_traces**(*ds*, *sn_list*, *start_date*, *end_date*, *length*, *output_folder*)

Dump mseed traces from an ASDF file in parallel

Parameters

- **ds** – ASDF Dataset
- **sn_list** – station list to process
- **start_date** – start date
- **end_date** – end date
- **length** – length of each mseed file
- **output_folder** – output folder

seismic.ASDFdatabase.asdf2mseed.**split_list**(*lst*, *npartitions*)

1.1.5 seismic.ASDFdatabase.asdf_preprocess module

Description: Reads waveforms from an ASDF file, optionally applies instrument response correction, resamples and outputs them to another ASDF file. This preprocessing is crucial for large-scale studies involving > 10000 Green's Functions, e.g. in ambient noise tomography. This approach significantly reduces IO bottlenecks and computational costs associated with having to apply instrument response corrections on data from a given station in alternative workflows.

References:

CreationDate: 18/07/19

Developer: rakib.hassan@ga.gov.au

Revision History: LastUpdate: 18/07/19 RH LastUpdate: dd/mm/yyyy Who Optional description

seismic.ASDFdatabase.asdf_preprocess.**create_station_asdf**(*input_asdf*, *output_folder*, *resample_rate*, *instrument_response_inventory*, *instrument_response_output*, *water_level*)

seismic.ASDFdatabase.asdf_preprocess.**getStationInventory**(*master_inventory*, *inventory_cache*, *netsta*)

seismic.ASDFdatabase.asdf_preprocess.**split_list**(*lst*, *npartitions*)

1.1.6 seismic.ASDFdatabase.create_small_chunks module

seismic.ASDFdatabase.create_small_chunks.**create_chunk**(*out_dir*, *trace*, *st_time*, *end_time*, *sta*)

seismic.ASDFdatabase.create_small_chunks.**main**()

1.1.7 seismic.ASDFdatabase.make_field_notesDB module

1.1.8 seismic.ASDFdatabase.plot_data_quality module

Description: Reads waveform data from a FederatedASDFDataSet and generates data-quality plots into a multi-page PDF file

For each station, the program read-in all the waveform data over the period specified (usually over a year), and then perform statistics analysis over the waveform data for subsequent plotting.

The first page of the PDF shows the stations on a basemap with marker colored according to the number of usable days ("good" data available in the day, not gap, not all zeros) And the color is determined by c=mapper.to_rgba(days): Relatively speaking, black/blue/cold-ish colors indicate higher percentage of good data in the station. And red/yellow/warm-ish colors indicate higher percentage of problematic data.

The following pages of the PDF will be plotting the daily-averaged seismic wave data with gaps and all-zeros days are shaded.

CreationDate: 19/09/19 Developer: rakib.hassan@ga.gov.au

Revision History: LastUpdate: 19/09/2019 RH LastUpdate: 27/05/2020 FZ Refactoring and docs run examples etc.

Todo: The script currently have a low pylint score 4.3/10. Need to refactor to comply with Python standard pep-8: add required docstrings. use smaller function to modularize better. Consider to generate an additional page to executive summarise info for user.

seismic.ASDFdatabase.plot_data_quality.**plot_results**(*stations*, *results*, *output_basename*)

seismic.ASDFdatabase.plot_data_quality.**process_data**(*rank*, *fds*, *stations*, *start_time*, *end_time*)

Parameters

- **rank** – processor rank in parallel runs
- **fds** – FederatedASDFDataSer instance
- **stations** – list containing tuples (net, sta, loc, cha, lon, lat)
- **start_time** – start-time in UTCDateTime format
- **end_time** – end-time in UTCDateTime format

Returns a list containing UTCDateTimes marking the start of each day and their corresponding means in each row

seismic.ASDFdatabase.plot_data_quality.**setup_logger**(*name*, *log_file*, *level=20*)
Function to setup a logger; adapted from stackoverflow

seismic.ASDFdatabase.plot_data_quality.**split_list**(*lst*, *npartitions*)

1.1.9 seismic.ASDFdatabase.query_input_yes_no module

From http://code.activestate.com/recipes/577058/

seismic.ASDFdatabase.query_input_yes_no.**query_yes_no**(*question*, *default='yes'*)
Ask a yes/no question via input() and return their answer.

"question" is a string that is presented to the user. "default" is the presumed answer if the user just hits <Enter>. It must be "yes" (the default), "no" or None (meaning an answer is required of the user).

The "answer" return value is one of "yes" or "no".

1.1.10 seismic.ASDFdatabase.sc3toasdf module

Description: Reads waveforms (within a given time-range) from a Seiscomp3 server and dumps out ASDF files, along with a json file containing associated metadata

References:

CreationDate: 13/09/18 Developer: rakib.hassan@ga.gov.au

Revision History: LastUpdate: 22/08/18 RH LastUpdate: dd/mm/yyyy Who Optional description

class seismic.ASDFdatabase.sc3toasdf.**DictToStr**
Bases: dict

seismic.ASDFdatabase.sc3toasdf.**hasOverlap**(*stime1*, *etime1*, *stime2*, *etime2*)

seismic.ASDFdatabase.sc3toasdf.**make_ASDF_tag**(*tr*, *tag*)

1.1.11 seismic.ASDFdatabase.seisds module

class seismic.ASDFdatabase.seisds.**SeisDB**(*json_file=False*, *generate_numpy_index=True*)
Bases: object

generateIndex()

get_data_percentage(*net*, *sta*, *chan*, *tags*)

get_gaps_intervals(*net*, *sta*, *chan*, *tags*)

get_recording_intervals(*net*, *sta*, *chan*, *tags*)

get_unique_information()
Method to retreive the unique channels and tags within an ASDF file from the JSON Database :return: (unique channels, unique tags)

is_chan_related(*chan*, *net*, *sta*, *loc*)
Method to test if a channel code is related to a given net/stn/tag/loc :param chan: Channel Code, String :param net: Network Code, String :param sta: Station Code, String :param sta: Location Code, String :return: True/False, Bool

queryByTime(*net*, *sta*, *chan*, *tags*, *query_starttime*, *query_endtime*)

retrieve_full_db_entry(*json_key*)
Method to take an output from queryByTime and get the full information from the original JSON db :return: full DB

1.1.12 seismic.ASDFdatabase.utils module

seismic.ASDFdatabase.utils.**rtp2xyz**(*r*, *theta*, *phi*)

1.1.13 Module contents

1.2 seismic.amb_noise package

1.2.1 Module contents

1.3 seismic.gps_corrections package

1.3.1 Submodules

1.3.2 seismic.gps_corrections.gps_clock_correction_gui module

GUI interface to generating clock corrections from x-corr results.

class seismic.gps_corrections.gps_clock_correction_gui.**GpsClockCorrectionApp**(*master=None*)
Bases: tkinter.Frame

busy()

not_busy()

class seismic.gps_corrections.gps_clock_correction_gui.**SplineDegreeWidget**(*index*, *initial_value*, *master=None*, *command=None*)

Bases: tkinter.LabelFrame

property is_enabled

property spline_degree

1.3.3 seismic.gps_corrections.picks_reader_utils module

Functions to read picks from custom csv file into Pandas DataFrame format and related data filtering utilities.

seismic.gps_corrections.picks_reader_utils.**compute_matching_network_mask**(*df*, *net_dict*)

Compute the mask for df of network codes and station codes that match the sequence specified in net_dict.

This function differs from *pandas.DataFrame.isin()* function in that it only matches net.sta pairs from the same index in the 'net' and 'sta' lists from net_dict, whereas *isin()* matches net.sta pairs from arbitrary different positions in the lists.

Parameters

- **df** (*pandas.DataFrame*) – Pandas dataframe loaded from a pick event ensemble
- **net_dict** (*dict of corresponding network and station codes under keys 'net' and 'sta' respectively.*) – Lists of corresponding network and station codes to match pariwise against columns 'net and 'sta'.

Returns Row mask for df haev value True in rows whose ['net', 'sta'] columns match one of the ('net', 'sta') pairs generated from net_dict.

Return type numpy.array(bool)

seismic.gps_corrections.picks_reader_utils.**generate_large_events_catalog**(*df_picks*, *min_magnitude=8.0*, *min_record_count=400*, *label_historical_events=Tru*

Use input picks dataset to identify dates of large seismic events

Parameters

- **df_picks** (*pandas.DataFrame*) – Dataframe of picks data following PICKS_TABLE_SCHEMA
- **min_magnitude** (*float, optional*) – Minimum seismic magnitude to be considered 'large' event, defaults to 8.0
- **min_record_count** (*int, optional*) – Minimum number of records per day required with magnitude >= min_magnitude to be included in the event catalog, defaults to 400
- **label_historical_events** (*bool, optional*) – Whether to populate fixed list of known historical events, defaults to True

Returns Dataframe of large seismic events indexed by date.

Return type pandas.DataFrame

seismic.gps_corrections.picks_reader_utils.**get_network_date_range**(*df*, *netcode*)

Get the date range of pick events in df for a given network code

Parameters

- **df** (*pandas.DataFrame*) – Pandas dataframe loaded from a pick event ensemble
- **netcode** (*str*) – Network code whose pick event dates min and max will be returned

Returns Min and max dates of picks for given network

Return type tuple(obspy.UTCDateTime, obspy.UTCDateTime)

seismic.gps_corrections.picks_reader_utils.**get_network_location_mean**(*df*, *netcode*)

Get the mean station latitude and longitude coordinates for all stations in a given network.

Parameters

- **df** (*pandas.DataFrame*) – Pandas dataframe loaded from a pick event ensemble
- **netcode** (*str*) – Network code for which mean coordinates will be returned

Returns Mean (latitude, longitude) coordinates of stations in the network

Return type tuple(float, float)

seismic.gps_corrections.picks_reader_utils.**get_network_stations**(*df*, *netcode*)

Get the unique station codes belonging to a given network from a Pandas DataFrame

Parameters

- **df** (*pandas.DataFrame*) – Pandas dataframe loaded from a pick event ensemble
- **netcode** (*str*) – Network code for which station codes will be returned

Returns Sorted list of station code strings extracted from df

Return type list(str)

seismic.gps_corrections.picks_reader_utils.**get_overlapping_date_range**(*df*, *network_1*, *network_2*)

Get the range of dates for which pick events in df from network_1 set of stations overlap with any picks from network_2 set of stations.

Parameters

- **df** (*pandas.DataFrame*) – Pandas dataframe loaded from a pick event ensemble
- **network_1** (*dict of corresponding network and station codes under keys 'net' and 'sta'*) – Network and station codes for first network
- **network_2** (*dict of corresponding network and station codes under keys 'net' and 'sta'*) – Network and station codes for second network

Returns Start and end dates of datetime range during which pick events overlap for network_1 and network_2

Return type tuple(obspy.UTCDateTime, obspy.UTCDateTime)

seismic.gps_corrections.picks_reader_utils.**get_station_date_range**(*df*, *netcode*, *statcode*)

Get the date range of pick events in df for a given network and station code

Parameters

- **df** (*pandas.DataFrame*) – Pandas dataframe loaded from a pick event ensemble
- **netcode** (*str*) – Network code
- **statcode** (*str*) – Station code

Returns Min and max dates of picks for given network and station

Return type tuple(obspy.UTCDateTime, obspy.UTCDateTime)

seismic.gps_corrections.picks_reader_utils.**read_picks_ensemble**(*csv_file*)

Read picks from CSV file using PICKS_TABLE_SCHEMA and return as a Pandas DataFrame.

Parameters **csv_file** (*str or path*) – Input file containing picks ensemble

1.3.4 seismic.gps_corrections.relative_tt_residuals_plotter module

Bulk analysis script for analysing relative traveltime residuals from a pick ensemble for the purpose of identifying time periods of GPS clock error in specific stations.

Example usage, which plots 7X.MA11 and 7X.MA12 residuals relative to all common events on AU network:

```
relative_tt_residuals_plotter.py --network1=AU --networks2=7X
--stations2="MA11,MA12" /c/data_cache/Picks/20190320/ensemble.p.txt
```

class seismic.gps_corrections.relative_tt_residuals_plotter.**BatchOptions**

Bases: object

Simple container type for run time options.

class seismic.gps_corrections.relative_tt_residuals_plotter.**DisplayOptions**

Bases: object

Simple container type for display options.

class seismic.gps_corrections.relative_tt_residuals_plotter.**FilterOptions**

Bases: object

Simple container type for filtering options.

seismic.gps_corrections.relative_tt_residuals_plotter.**analyze_target_relative_to_ref**(*df_picks*, *ref_stn*, *target_stns*, *filter_opti*

Analyze a single (reference) station's residuals relative to all the other stations in a (target) network.

Parameters

- **df_picks** (*pandas.DataFrame*) – Pandas dataframe loaded from a pick event ensemble
- **ref_stn** (*dict of corresponding network and station codes under keys 'net' and 'sta' (expected to be just one entry)*) – Network and station codes for reference network (expected to be just one entry)

- **target_stns** (*dict of corresponding network and station codes under keys 'net' and 'sta'*) – Network and station codes for target network
- **filter_options** (*class FilterOptions*) – Filter options.

Returns Dataframe with filtering applied and reference residual populated ready for plotting.

Return type pandas.DataFrame

seismic.gps_corrections.relative_tt_residuals_plotter.**apply_event_quality_filtering**(*df*, *ref_stn*, *filter_optio*

Apply event quality requirements to pick events. The event magnitude threshold is only used to filter rows where the quality metrics ['snr', 'qualityMeasureCWT', 'qualityMeasureSlope', 'nSigma'] are all zero.

Parameters

- **df** (*pandas.DataFrame*) – Pandas dataframe loaded from a pick event ensemble
- **ref_stn** (*dict of corresponding network and station codes under keys 'net' and 'sta' (expected to be just one entry)*) – Network and station codes for reference network (expected to be just one entry)
- **filter_options** (*class FilterOptions*) – Filter options.

Returns Filtered dataframe of pick events

Return type pandas.DataFrame

seismic.gps_corrections.relative_tt_residuals_plotter.**broadcast_ref_residual_per_event**(*df_pl ref_n ref_s filter_o*

For each event in the dataframe, figure out the best reference residual and add it to new column 'ttResidualRef'

Parameters

- **df** (*pandas.DataFrame*) – Pandas dataframe loaded from a pick event ensemble
- **ref_netcode** (*str*) – The reference network code
- **ref_stacode** (*str*) – The reference station code
- **filter_options** (*class FilterOptions*) – Filter options.

Returns Dataframe with a consistent reference residual populated for each event for given reference station.

Return type pandas.DataFrame

seismic.gps_corrections.relative_tt_residuals_plotter.**determine_alternate_matching_codes**(*df ir or i- na*

Find stations from other networks in df with the same station codes, but different network codes, whose positions match the stations of the same code in the original network.

Parameters

- **df** (*pandas.DataFrame*) – Pandas dataframe loaded from a pick event ensemble

- **iris_file** (*str or path*) – IRIS catalog query result file containing network and station information in stationtxt format
- **original_network** (*str*) – Network code whose station codes will be extracted from IRIS file

Returns Matching sequences of network and station codes

Return type tuple(str), tuple(str)

seismic.gps_corrections.relative_tt_residuals_plotter.**filter_duplicated_network_codes**(*df_pic*

Filter picks dataframe to remove records for selected known duplicate network codes based on date of the records we want to keep.

Parameters **df_picks** (*pandas.DataFrame*) – Picks dataframe to filter

Returns Filtered picks dataframe with unwanted records removed.

Return type pandas.DataFrame

seismic.gps_corrections.relative_tt_residuals_plotter.**filter_limit_channels**(*df_picks*, *channel_pref*)

Filter picks dataframe to a limited range of preferred channel types.

Parameters **df_picks** (*pandas.DataFrame*) – Picks dataframe to filter

Returns Filtered picks dataframe with only preferred channels

Return type pandas.DataFrame

seismic.gps_corrections.relative_tt_residuals_plotter.**filter_to_teleseismic**(*df_picks*, *min_dist_deg*, *max_dist_deg*)

Filter picks dataframe to limited range of teleseismic distances of the original events.

Parameters

- **df_picks** (*pandas.DataFrame*) – Picks dataframe to filter
- **min_dist_deg** (*float*) – Minimum teleseismic distance (degrees)
- **max_dist_deg** (*float*) – Maximum teleseismic distance (degrees)

Returns Filtered picks dataframe containing only events with limited range of teleseismic distance.

Return type pandas.DataFrame

seismic.gps_corrections.relative_tt_residuals_plotter.**get_iris_station_codes**(*src_file*, *original_network*)

Extract the station codes for a given network code from a IRIS query result file.

Parameters

- **src_file** (*str or path*) – IRIS catalog query result file containing network and station information, in stationtxt format
- **original_network** (*str*) – Network code whose station codes will be extracted from IRIS file

Returns Pandas dataframe with station codes as the index and station latitude, longitude as columns

Return type pandas.DataFrame

seismic.gps_corrections.relative_tt_residuals_plotter.**pandas_timestamp_to_plottable_datetim**
Convert float UTC timestamp to equivalent type that is plottable by matplotlib

Parameters **data** (*pandas.Series*) – Pandas series of float timestamps

Returns Array of Python datetimes

Return type numpy.array(datetime)

seismic.gps_corrections.relative_tt_residuals_plotter.**utc_time_string_to_plottable_datetime**
Convert a UTC timestamp string to datetime type that is plottable by matplotlib

Parameters **utc_timestamp_str** (*str*) – ISO-8601 UTC timestamp string

Returns Plottable datetime value

Return type datetime.datetime with tzinfo

1.3.5 Module contents

1.4 seismic.inventory package

1.4.1 Subpackages

seismic.inventory.dataio package

Submodules

seismic.inventory.dataio.catalogcsv module

Lightweight reader for CSV seismic event catalogs, indexing the found events by event ID and station.

This was adapted for the speical use case of distance-to-event QA checks performed for ticket PST-340.

class seismic.inventory.dataio.catalogcsv.**CatalogCSV**(*event_folder*, *sampling_factor=1.0*)

Bases: object

Lightweight parser for seismic event catalog.

Catalog is format as follows:

```
#EHB, 2005, 09, 16, 07, 28, 39.001,  126.93300,    4.18700,    2.90000,    28, 4.
↪50, -999.00, -999.00, -999.00,       1, 134.3000, 1
FITZ, BHZ,  ,  ,  ,  , P , 2005, 09, 16, 07, 33, 37.00,  22.180
WR0 , BHZ,  ,  ,  ,  , P , 2005, 09, 16, 07, 34, 06.00,  25.130
KUM , BHZ,  ,  ,  ,  , P , 2005, 09, 16, 07, 35, 02.00,  26.220
MEEK, BHZ,  ,  ,  ,  , P , 2005, 09, 16, 07, 35, 04.00,  31.680
FORT, BHZ,  ,  ,  ,  , P , 2005, 09, 16, 07, 35, 32.00,  34.780
STKA, BHZ,  ,  ,  ,  , P , 2005, 09, 16, 07, 36, 04.00,  38.480
KSM , BHZ,  ,  ,  ,  , Pn, 2005, 09, 16, 07, 32, 39.00,  16.820
KAKA, BHZ,  ,  ,  ,  , P , 2005, 09, 16, 07, 33, 04.00,  17.660
FITZ, BHZ,  ,  ,  ,  , P , 2005, 09, 16, 07, 33, 36.00,  22.180
...
```

The header row, which starts with '#', indicates the number of phases listed in the subsequent lines of arrival data (28 in this example).

get_events()

seismic.inventory.dataio.catalogcsv.**recursive_glob**(*treeroot*, *pattern*)
Generate a complete list of files matching pattern under the root of a directory hierarchy.

Parameters

- **treeroot** (*str or pathlib.Path*) – Path to the root of the directory tree.
- **pattern** (*str*) – File name pattern to match, e.g. "*.csv"

Returns List of paths to the files matching the pattern, qualified relative to treeroot

Return type list(str)

seismic.inventory.dataio.event_attrs module

class seismic.inventory.dataio.event_attrs.**Arrival**(*net*, *sta*, *loc*, *cha*, *lon*, *lat*, *elev*, *phase*, *utctime*, *distance*)
Bases: object

Arrival of seismic event signal at other location

class seismic.inventory.dataio.event_attrs.**Event**
Bases: object

Container for a seismic event with high level attributes.

class seismic.inventory.dataio.event_attrs.**Magnitude**(*mag*, *mag_type*)
Bases: object

Seismic event magnitude

class seismic.inventory.dataio.event_attrs.**Origin**(*utctime*, *lat*, *lon*, *depthkm*)
Bases: object

Container for seismic event origin (location) within the earth

epicenter()
Get the epicenter attribute as (lat, long). Lat and long are in degrees.

Returns Location of the seismic event epicenter

Return type tuple(double, double)

location()
Get the location attribute as (lat, long, depth). Lat and long are in degrees, depth is in km.

Returns Location of the seismic event origin

Return type tuple(float, float, float)

seismic.inventory.dataio.event_attrs.**indentprint**(*x*)

Module contents

1.4.2 Submodules

1.4.3 seismic.inventory.add_time_corrections module

Add GPS clock time correction csv_data into inventory file to get a modified station xml file

CreationDate: 24/02/2020

Developer: fei.zhang@ga.gov.au

seismic.inventory.add_time_corrections.**add_gpscorrection_into_stationxml**(*csv_file*, *input_xml*, *out_xml=None*)

Read in the correction CSV data from a file, get the station metadata node from input_xml file, then add the CSV data into the station xml node to write into out_xml

Parameters

- **csv_file** – input csv file with correction data
- **input_xml** – input original stationXML file which contains the metadata for the network and station of csv_file
- **out_xml** – Directory of the output xml file

Returns full path of the output xml file

seismic.inventory.add_time_corrections.**extract_csvdata**(*path2xml*)

Read the station xml file and extract the csv data to be parsed by pandas

Parameters **path2xml** – path_to_stationxml

Returns csv_str

seismic.inventory.add_time_corrections.**get_csv_correction_data**(*path_csvfile*)

Read in the csv data from an input file, get the network_code, station_code, csv_data. Format:: $ head 7D.DE43_clock_correction.csv net,sta,date,clock_correction 7D,DE43,2012-11-27,1.0398489013215846 7D,DE43,2012-11-28,0.9408504322549281 7D,DE43,2012-11-29,0.8418519631882714 7D,DE43,2012-11-30,0.7428534941216148 7D,DE43,2012-12-01,0.6438550250549583

Parameters **path_csvfile** – input csv file in /g/data/ha3/Passive/SHARED_DATA/GPS_Clock/corrections/

Returns (network_code, station_code, csv_data)

1.4.4 seismic.inventory.engd2stxml module

Engdahl and ISC STN data conversion to FDSN station XML

Creates database of stations from .STN files, curates the data using heuristic rules, and exports new stations to FDSN station XML format network with non-empty, nominal instrument response data.

Cleanup steps applied: * Removes “blacklisted” networks that add little value and cause problems due to station code conflicts. * Add default station dates where missing. * Make “future” station end dates consistent to max Pandas timestamp. * Remove records with illegal station codes. * Remove duplicate station records. * Merge overlapping channel dates for given NET.STAT.CHAN to a single epoch.

seismic.inventory.engd2stxml.**cleanup_database**(*df*)
Main cleanup function encompassing the sequential data cleanup steps.

Parameters **df** (*pandas.DataFrame conforming to table_format.TABLE_SCHEMA*) – Dataframe of station records to clean up

Returns Cleaned up dataframe of station records

Return type pandas.DataFrame conforming to table_format.TABLE_SCHEMA

seismic.inventory.engd2stxml.**compute_neighboring_station_matrix**(*df*)
Compute sparse matrix representing index of neighboring stations.

Ordering of matrix corresponds to ordering of Dataframe df, which is expected to be sequential integer indexed. For a given station index i, then the non-zero off-diagonal entries in row i of the returned matrix indicate the indices of adjacent, nearby stations.

Parameters **df** (*pandas.DataFrame conforming to table_format.TABLE_SCHEMA*) – Dataframe containing station records.

Returns Sparse binary matrix having non-zero values at indices of neighboring stations.

Return type scipy.sparse.csr_matrix

seismic.inventory.engd2stxml.**latlong_to_cosinedistance**(*latlong_deg_set1*, *latlong_deg_set2*)
Compute the approximate cosine distance between each station of 2 sets.

Each set is specified as a numpy column vector of [latitude, longitude] positions in degrees.

This function performs an outer product and will produce matrix of size N0 x N1, where N0 is the number of rows in latlong_deg_set1 and N1 is the number of rows in latlong_deg_set2.

Returns np.ndarray containing cosines of angles between each pair of stations from the input arguments. If input is 1D, convert to 2D for consistency of matrix orientations.

Parameters

- **latlong_deg_set1** (*np.ndarray*) – First set of numpy column vector of [latitude, longitude] positions in degrees
- **latlong_deg_set2** (*np.ndarray*) – Second set of numpy column vector of [latitude, longitude] positions in degrees

Returns Array containing cosines of angles between each pair of stations from the input arguments.

Return type np.ndarray

seismic.inventory.engd2stxml.**main**(*iris_xml_file*, *stations_folder*, *output_directory*, *test_mode=False*)
Main entry point.

Parameters

- **iris_xml_file** (*str or pathlib.Path*) – Name of IRIS xml file to load (generated by script update_iris_inventory.py)
- **stations_folder** (*str or pathlib.Path*) – Path to folder containing STN files to process

seismic.inventory.engd2stxml.**merge_overlapping_channel_epochs**(*df*)
Removed overlapping time intervals for a given network.station.channel, as this needs to be unique.

This function expects the input DataFrame to have a sequential integer index.

Parameters **df** (*pandas.DataFrame conforming to table_format.TABLE_SCHEMA*) – Dataframe of station records in which to merge overlapping channel dates

seismic.inventory.engd2stxml.**populate_default_station_dates**(*df*)

Replace all missing station start and end dates with default values. Replace all missing channel start and end dates with their corresponding station/end dates.

Parameters **df** (*pandas.DataFrame conforming to table_format.TABLE_SCHEMA*) – Dataframe in which to fill in missing station start and end dates.

seismic.inventory.engd2stxml.**read_eng**(*fname*)

Read Engdahl STN file having the following format of fixed width formatted columns:

:: AAI Ambon BMG, Indonesia, IA-Ne -3.6870 128.1945 0.0 2005001 2286324 I :: AAII -3.6871 128.1940 0.0 2005001 2286324 I :: AAK Ala Archa Kyrgyzstan 42.6390 74.4940 0.0 2005001 2286324 I :: ABJI -7.7957 114.2342 0.0 2005001 2286324 I :: APSI -0.9108 121.6487 0.0 2005001 2286324 I :: AS01 Alice Springs Arra -23.6647 133.9508 0.0 2005001 2286324 I :: 0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890 :: 10 20 30 40 50 60 70 80

Each Station Code (first column) might NOT be unique, and network codes are missing here, so all records are placed under 'GE' network.

Parameters **fname** (*str*) – STN file name to load

Returns Pandas Dataframe containing the loaded data in column order of TABLE_COLUMNS.

Return type pandas.DataFrame conforming to table_format.TABLE_SCHEMA

seismic.inventory.engd2stxml.**read_isc**(*fname*, *use_pickle=False*)

Read ISC station inventory having such format and convert to Pandas DataFrame:

:: 109C 32.8892 -117.1100 0.0 2006-06-01 04:11:18 2008-01-04 01:26:30 :: 109C 32.8882 -117.1050 150.0 2008-01-04 01:26:30 :: FDSN 109C TA – BHZ 2004-05-04 23:00:00 2005-03-03 23:59:59 :: FDSN 109C TA – LHZ 2004-05-04 23:00:00 2005-03-03 23:59:59 :: FDSN 109C TA – BHZ 2005-04-11 00:00:00 2006-01-25 22:31:10 :: FDSN 109C TA – LHZ 2005-04-11 00:00:00 2006-01-25 22:31:10 :: 0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890 :: 10 20 30 40 50 60 70 80

The lines starting with a station code are HEADER rows, and provide the station coordinates. The idented lines starting with FDSN provide distinct station, network and channel data for the given station location.

Parameters **fname** (*str*) – STN file name to load

Returns Pandas Dataframe containing the loaded data in column order of TABLE_COLUMNS.

Return type pandas.DataFrame conforming to table_format.TABLE_SCHEMA

seismic.inventory.engd2stxml.**remove_blacklisted**(*df*)

Remove network codes that are explicitly blacklisted due to QA issues or undesirable overlap with trusted FDSN station codes.

Parameters **df** (*pandas.DataFrame conforming to table_format.TABLE_SCHEMA*) – Dataframe of initially loaded data from STN files

seismic.inventory.engd2stxml.**remove_duplicate_stations**(*df*, *neighbor_matrix*)

Remove stations which are identified as duplicates: * Removes duplicated stations in df based on station code and locality of lat/long coordinates. * Removes duplicated stations based on codes and channel data matching, IRRESPECTIVE of locality

Parameters

- **df** (*pandas.DataFrame conforming to table_format.TABLE_SCHEMA*) – Dataframe containing station records. Is modified during processing.
- **neighbor_matrix** (*scipy.sparse.csr_matrix*) – Sparse binary matrix having non-zero values at indices of neighboring stations.

Returns Dataframe containing station records with identified duplicates removed.

Return type pandas.DataFrame conforming to table_format.TABLE_SCHEMA

seismic.inventory.engd2stxml.**remove_illegal_stationNames**(*df*)

Remove records for station names that do not conform to expected naming convention. Such names can cause problems in downstream station, in particular names with asterisk.

Parameters **df** (*pandas.DataFrame conforming to table_format.TABLE_SCHEMA*) – Dataframe containing station records from which illegal station codes should be removed (modified in-place)

seismic.inventory.engd2stxml.**reportUnpickleFail**(*filename*)

Standard failure report message when trying to unpickle file.

Parameters **filename** (*str*) – The name of the file that failed to unpickle.

seismic.inventory.engd2stxml.**write_portable_inventory**(*df*, *fname*)

Write the final database to re-usable file formats.

Parameters

- **df** (*pandas.DataFrame conforming to table_format.TABLE_SCHEMA*) – Dataframe containing complete inventory of station records.
- **fname** (*str*) – Base filename to export to. Output filename will be appended with file format extensions.

1.4.5 seismic.inventory.fdsnxml_convert module

Helper functions to convert FDSN station XML files to Seiscomp3 SC3ML format.

Can be used as a standalone tool as well: *fdsnxml_convert.py src_path dst_path*

seismic.inventory.fdsnxml_convert.**sc3_conversion_available**()

Check whether conversion to seiscomp3 format is available os this system. Only works for Python 3, for Python 2 it always returns True (i.e. give it a try).

Returns True if conversion to sc3ml can be performed on this system, False otherwise.

Return type bool

seismic.inventory.fdsnxml_convert.**toSc3ml**(*src_path*, *dst_path*, *response_fdsnxml=None*)

Convert file(s) in src_path from FDSN station XML to SC3ML and emit result(s) to dst_path.

If src_path is a file, dst_path will be treated as a file. If dst_path already exists as a folder, an exception is raised.

If src_path is a folder, dst_path will be treated as a folder. If dst_path already exists as a file, an exception is raised. The src_path directory hierarchy will be walked to find all .xml files, each of which will be converted to a mirrored relative path under dst_path.

Parameters

- **src_path** (*str or pathlib.Path*) – The source path from which to input XML file(s).

- **dst_path** (*str or pathlib.Path*) – The destination path into which converted sc3ml file(s) are output.
- **response_fdsnxml** (*str or Path*) – Path to existing for containing dummy response to insert into records where instrument response is missing.

Raises OSError, FileNotFoundError, RuntimeError, FileExistsError

1.4.6 seismic.inventory.generate_network_plots module

Load inventory file from hdf5 format and export station location plots to graphics files.

Usage:: python generate_network_plots.py inventory_20190206.h5

where inventory_20190206.h5 here is an example file name. The output folder for the graphics files is inferred from the inventory file name, which would be "inventory_20190206" in this example.

1.4.7 seismic.inventory.inventory_merge module

Merge a master inventory (e.g. from IRIS) with custom inventories.

seismic.inventory.inventory_merge.**get_matching_net**(*search_space*, *item_to_find*)

seismic.inventory.inventory_merge.**inventory_merge**(*iris_inv*, *custom_inv*, *output_file*, *test_mode=False*)

Merge an IRIS inventory with a custom inventory, filtering out any custom inventory records that duplicate IRIS records.

Parameters

- **iris_inv** (*str or Path to file*) – Station XML file from which to load IRIS inventory
- **custom_inv** (*str or Path to file*) – Station XML file with custom records to merge with IRIS inventory
- **output_file** (*str or Path to file*) – File name of output file which will contain merged IRIS and custom inventory.

seismic.inventory.inventory_merge.**mean_lat_long**(*network*)

Compute mean of station lat/long coordinates within a network

Parameters **network** (*obspy.core.inventory.network.Network*) – Network on which to compute mean lat/long

seismic.inventory.inventory_merge.**prune_iris_duplicates**(*db_other*, *db_iris*)

Prune the records out of db_other which duplicate records that exist in db_iris.

Filtering concepts: Station codes and channel codes are reliable, but network codes and station/channel dates are not. Station lat/lon locations are approximately reliable. Therefore the merge method here matches records with matching station code, channel code and approximate location. For each match, we examine station dates, and discard any records of db_other that overlap with db_iris.

db_iris is not changed by this function.

Parameters

- **db_other** (*pandas.DataFrame*) – DataFrame containing metadata records of custom inventory

- **db_iris** (*pandas.DataFrame*) – DataFrame containing metadata records of IRIS inventory

Returns DataFrame containing records from db_other which are unique and do not appear in IRIS.

Return type pandas.DataFrame

1.4.8 seismic.inventory.inventory_split module

Split a station inventory file into a separate file per network.

seismic.inventory.inventory_split.**inventory_split**(*inv_file*, *output_folder*, *sc3ml=False*)

Split an inventory file (station XML) into separate inventory file per network, stored in folder output_folder.

Parameters

- **inv_file** (*str or path*) – Station inventory to be split. This file is not changed by the script.
- **output_folder** (*str or path to folder*) – Folder where per-network station xml files will be generated.
- **sc3ml** (*bool, optional*) – If True, try to convert output files to sc3ml format if possible using seiscomp3. Defaults to False.

seismic.inventory.inventory_split.**split_inventory_by_network**(*obspy_inv*, *output_folder*, *validate=False*)

Export a station XML file per network for each network in given obspy Inventory.

Parameters

- **obspy_inv** (*obspy.core.inventory.inventory.Inventory*) – Obspy Inventory containing the networks to export to file.
- **output_folder** (*str or Path*) – Folder in which to output the per-network XML files. Will be created if doesn't yet exist.
- **validate** (*bool, optional*) – Whether to validate the station data on write, defaults to False

1.4.9 seismic.inventory.inventory_util module

Utility functions and constants shared amongst inventory management modules.

class seismic.inventory.inventory_util.**Instrument**(*sensor*, *response*)

Bases: tuple

property response

Alias for field number 1

property sensor

Alias for field number 0

seismic.inventory.inventory_util.**extract_unique_sensors_responses**(*inv*, *req*, *show_progress=True*, *black-listed_networks=None*, *test_mode=False*)

For the channel codes in the given inventory, determine a nominal instrument response suitable for that code.

Note that no attempt is made here to determine an ACTUAL correct response for a given network and station. The only requirement here is to populate a plausible, non-empty response for a given channel code, to placate Seiscomp3 which requires that an instrument response always be present.

Parameters

- **inv** (*obspy.Inventory*) – Seismic station inventory
- **req** (*Object conforming to interface of 'requests' library*) – Request object to use for URI query

Returns Python dict of (obspy.core.inventory.util.Equipment, obspy.core.inventory.response.Response) indexed by str representing channel code

Return type {str: Instrument(obspy.core.inventory.util.Equipment, obspy.core.inventory.response.Response) } where Instrument is a namedtuple("Instrument", ['sensor', 'response'])

seismic.inventory.inventory_util.**load_station_xml**(*inventory_file*)

Load a stationxml file

Parameters **inventory_file** (*str or path*) – [description]

seismic.inventory.inventory_util.**obtain_nominal_instrument_response**(*netcode*, *statcode*, *chcode*, *req*)

For given network, station and channel code, find suitable response(s) in IRIS database and return as dict of obspy instrument responses.

Parameters

- **netcode** (*str*) – Network code (may include wildcards)
- **statcode** (*str*) – Station code (may include wildcards)
- **chcode** (*str*) – Channel code (may include wildcards)
- **req** (*Object conforming to interface of 'requests' library*) – Request object to use for URI query

Returns Dictionary of instrument responses from IRIS for given network(s), station(s) and channel(s).

Return type dict of {str, *Instrument*(obspy.core.inventory.util.Equipment, obspy.core.inventory.response.Response)}

1.4.10 seismic.inventory.iris_query module

Helper functions for making and managing web queries to IRIS web service.

seismic.inventory.iris_query.**form_channel_request_url**(*netmask='*'*, *statmask='*'*, *chanmask='*'*)

Form request URL to download station inventory in stationxml format, down to channel level, with the given filters applied to network codes, station codes and channel codes.

Parameters

- **netmask** – Pattern of network codes to match, comma separated with wildcards, defaults to "*"
- **netmask** – str, optional

- **statmask** – Pattern of station codes to match, comma separated with wildcards, defaults to "*"
- **statmask** – str, optional
- **chanmask** – Pattern of channel codes to match, comma separated with wildcards, defaults to "*"
- **chanmask** – str, optional

Returns Fully formed URL to perform IRIS query and get back FDSN station XML result.

Return type str

seismic.inventory.iris_query.**form_response_request_url**(*netmask*, *statmask*, *chanmask*)

Form request URL to download station inventory in stationxml format, down to response level, for the given network, station and channel codes.

Parameters

- **netmask** – Pattern of network codes to match, comma separated with wildcards
- **netmask** – str, optional
- **statmask** – Pattern of station codes to match, comma separated with wildcards
- **statmask** – str, optional
- **chanmask** – Pattern of channel codes to match, comma separated with wildcards
- **chanmask** – str, optional

Returns Fully formed URL to perform IRIS query and get back FDSN station XML result.

Return type str

seismic.inventory.iris_query.**set_text_encoding**(*resp*, *quiet=False*)

For the given response object, set its encoding from the contents of the text returned from server.

Parameters **resp** (*requests.Response*) – Query response object returned by response.get()

1.4.11 seismic.inventory.pdconvert module

Helper functions for converting between Pandas dataframe and FDSN Inventory, Network, Station and Channel objects.

seismic.inventory.pdconvert.**dataframe_to_fdsn_station_xml**(*inventory_df*, *nominal_instruments*, *filename*, *show_progress=True*)

Export dataframe of station metadata to FDSN station xml file

Parameters

- **inventory_df** (*pandas.DataFrame conforming to table_format.TABLE_SCHEMA*) – Dataframe containing all the station records to export.
- **nominal_instruments** (*{str: Instrument(obspy.core.inventory.util.Equipment, obspy.core.inventory.response.Response) }*) – Dictionary mapping from channel code to nominal instrument characterization
- **filename** (*str or path*) – Output filename

seismic.inventory.pdconvert.**dataframe_to_network**(*netcode*, *network_df*, *instrument_register*, *progressor=None*)

Convert Pandas dataframe with unique network code to obspy Network object.

Parameters

- **netcode** (*str*) – Network code
- **network_df** (*pandas.DataFrame conforming to table_format.TABLE_SCHEMA*) – Dataframe containing records for a single network code.
- **instrument_register** – Dictionary of nominal instrument responses indexed by channel code, defaults to None
- **instrument_register** – dict of {str, Instrument(obspy.core.inventory.util.Equipment, obspy.core.inventory.response.Response)}, optional
- **progressor** – Progress bar functor to receive progress updates, defaults to None
- **progressor** – Callable object receiving incremental update on progress, optional

Returns Network object containing the network information from the dataframe

Return type obspy.core.inventory.network.Network

seismic.inventory.pdconvert.**inventory_to_dataframe**(*inv_object*, *show_progress=True*)

Convert a obspy Inventory object to a Pandas Dataframe.

Parameters

- **inv_object** (*obspy.core.inventory.inventory.Inventory*) – Obspy inventory object to convert to dataframe
- **show_progress** – Whether to use a progress bar, defaults to True
- **show_progress** – bool, optional

Returns Pandas Dataframe with sequential integer index and sorted by [NetworkCode, StationCode]. Only populates entries for non-empty channels.

Return type pandas.DataFrame conforming to table_format.TABLE_SCHEMA

1.4.12 seismic.inventory.plotting module

Helper functions for plotting FDSN Network objects.

seismic.inventory.plotting.**save_network_local_plots**(*df*, *plot_folder*, *progressor=None*, *include_stations_list=True*)

Save visual map plot per network, saved to file netcode.png.

Parameters

- **df** (*pandas.DataFrame conforming to table_format.TABLE_SCHEMA*) – Dataframe of station records to save.
- **plot_folder** (*str*) – Name of output folder
- **progressor** – Progress bar functor to receive progress updates, defaults to None
- **progressor** – Callable object receiving incremental update on progress, optional
- **include_stations_list** – If True, also export stationtxt file alongside each png file, defaults to True

- **include_stations_list** – bool, optional

seismic.inventory.plotting.**save_station_local_plots**(*df*, *plot_folder*, *progressor=None*, *include_stations_list=True*)

Save visual map plot per station, saved to file netcode.stationcode.png.

Parameters

- **df** (*pandas.DataFrame conforming to table_format.TABLE_SCHEMA*) – Dataframe of station records to save.
- **plot_folder** (*str*) – Name of output folder
- **progressor** – Progress bar functor to receive progress updates, defaults to None
- **progressor** – Callable object receiving incremental update on progress, optional
- **include_stations_list** – If True, also export stationtxt file alongside each png file, defaults to True
- **include_stations_list** – bool, optional

1.4.13 seismic.inventory.response module

Description: Implements a Class for reading/storing responses from a number of sources. The ResponseFactory class is used for attaching 'bogus' responses to station inventories that lack them.

References:

CreationDate: 14/02/19

Developer: rakib.hassan@ga.gov.au

Revision History: LastUpdate: 14/02/19 RH LastUpdate: dd/mm/yyyy Who Optional description

class seismic.inventory.response.**ResponseFactory**

Bases: object

The ResponseFactory class encapsulates the generation of a collection of named Instrument Response Objects from a variety of sources. Currently it provides the facility to create Response objects from two sources, namely, Poles and Zeroes supplied by the user and from StationXML files generated from RESP files using the PDCC tool (link below). The conversion of RESP files into a corresponding StationXML file, at this stage, must take place externally, because ObsPy lacks that functionality. The intended usage of this class during the creation of an ASDF dataset is as follows:

```
1. User creates a number of uniquely named Response objects (see associated tests␣
↪as well) pertaining to different
   channels in a given survey.
2. User fetches these Response objects from an instance of ResponseFactory as␣
↪needed, while creating ObsPy Channel
   objects, during which Response objects can be passed in as an argument.
3. User builds a hierarchy of channel->station->network inventories, with the␣
↪appropriate instrument response
   information embedded
4. The master FDSN StaionXML file output after step 3 can then be converted into␣
↪an SC3ML file (which can be ingested
   by SeisComp3) using the fdsnxml2inv tool.
```

PDCC tool: https://ds.iris.edu/ds/nodes/dmc/software/downloads/pdcc/

CreateFromInventory(*name*, *obspy_inventory*)
Create response from an Inventory

Parameters

- **name** (*str*) – Name of the response for later retrieval
- **obspy_inventory** (*obspy.core.inventory.inventory.Inventory*) – Inventory from which to extract response

CreateFromPAZ(*name*, *pzTransferFunctionType*, *normFactor*, *normFreq*, *stageGain*, *stageGainFreq*, *poles*, *zeros*)

CreateFromStationXML(*name*, *respFileName*)
Create response from an XML file

Parameters

- **name** (*str*) – Name of the response for later retrieval
- **respFileName** (*str*) – XML file to load

class ResponseFromInventory(*source_inventory*)
Bases: object

Helper class to get Obspy Response object from an Inventory

Raises **RuntimeError** – Raises error if response not found

class ResponseFromPAZ(*pzTransferFunctionType='LAPLACE (RADIANS/SECOND)'*, *normFactor=80000.0*, *normFreq=0.01*, *stageGain=2000.0*, *stageGainFreq=0.01*, *poles=[0j]*, *zeros=[0j]*)
Bases: object

class ResponseFromStationXML(*respFileName*)
Bases: *seismic.inventory.response.ResponseFactory.ResponseFromInventory*

Helper class to get Obspy Response object from a station xml file

getResponse(*name*)
Retrieve response by name

Parameters **name** (*str*) – Name given to response at creation time

Raises **RuntimeError** – Raises error if name is not recognized

Returns The requested response

Return type obspy.core.inventory.response.Response

1.4.14 seismic.inventory.table_format module

Common format for Pandas Dataframe column ordering of Network, Station and Channel metadata.

1.4.15 seismic.inventory.update_iris_inventory module

Automatically update IRIS-ALL.xml file from IRIS web portal.

Outout file is saved as FDSN station xml. Script also generates human readable form as IRIS-ALL.txt.

Example usages:

python update_iris_inventory.py

python update_iris_inventory.py -o outfile.xml

python update_iris_inventory.py –netmask=U –statmask=K**

python update_iris_inventory.py –netmask=UW,LO –output outfile.xml

`seismic.inventory.update_iris_inventory.`**`cleanup`**(*tmp_filename*)
Helper function to clean up temporary file on disk.

Parameters **`tmp_filename`** (*str*) – File name to clean up

`seismic.inventory.update_iris_inventory.`**`regenerate_human_readable`**(*iris_data*, *outfile*)
Generate human readable, tabular version of the IRIS database.

Parameters

- **`iris_data`** (*str*) – String containing result string returned from IRIS query (without data errors).
- **`outfile`** (*str*) – Output text file name

`seismic.inventory.update_iris_inventory.`**`repair_iris_metadata`**(*iris*)
Perform text subtitutions to fix known errors in station xml returned from IRIS.

Parameters **`iris`** (*requests.models.Response*) – Response to IRIS query request containing response text

Returns The text from the response with known faulty data substituted with fixed data.

Return type str (Python 3) or unicode (Python 2)

`seismic.inventory.update_iris_inventory.`**`update_iris_station_xml`**(*req*, *output_file*, *options=None*)
Pull the latest IRIS complete station inventory (down to station level, not including instrument responses) from IRIS web service and save to file in FDSN station xml format.

Parameters

- **`req`** (*Object conforming to interface of 'requests' library*) – Request object to use for URI query
- **`output_file`** (*str*) – Destination file to generate
- **`options`** – Filtering options for network, station and channel codes, defaults to None
- **`options`** – Python dict of key-values pairs matching command line options, optional

1.4.16 Module contents

1.5 seismic.inversion package

1.5.1 Subpackages

seismic.inversion.mcmc package

Submodules

seismic.inversion.mcmc.bulk_inversion_report module

Produce PDF report of network stations showing RF inversion results

seismic.inversion.mcmc.plot_inversion module

Plot inversion results from Bodin code

seismic.inversion.mcmc.plot_inversion.**plot_bodin_inversion**(*pdf*, *data_dir*, *rf_waveform*, *station=''*)

Standardized plotting of the results of Bodin RF inversion code.

Module contents

seismic.inversion.wavefield_decomp package

Submodules

seismic.inversion.wavefield_decomp.call_count_decorator module

Decorator to count number of times a function is called.

seismic.inversion.wavefield_decomp.call_count_decorator.**call_counter**(*func*)

Decorator to count calls to a function. The number of calls can be queryied from func.counter.

Parameters **func** – Function whose calls to count

Returns func wrapper

seismic.inversion.wavefield_decomp.plot_nd_batch module

Batch plotting a MCMC solution for batch of stations to a single pdf file.

seismic.inversion.wavefield_decomp.plot_nd_batch.**convert_Vs_to_k**(*soln*, *config*)

Transform Vs variable into k variable in MCMC solution. Modifies soln in-place.

Parameters

- **soln** (*Customized scipy.optimize.OptimizeResult*) – Solution container
- **config** (*dict*) – Solution configuration

Returns None

seismic.inversion.wavefield_decomp.plot_nd_batch.**plot_aux_data**(*soln*, *config*, *log*, *scale*)

seismic.inversion.wavefield_decomp.runners module

Batch execution interfaces for wavefield continuation methods and solvers.

seismic.inversion.wavefield_decomp.runners.**curate_seismograms**(*data_all*, *curation_opts*, *logger*, *rotate_to_zrt=True*)

Curation function to remove bad data from streams. Note that this function will modify the input dataset during curation.

Parameters

- **data_all** – NetworkEventDataset containing seismograms to curate.
- **curation_opts** – Dict containing curation options.
- **logger** – Logger for emitting log messages
- **rotate_to_zrt** – Whether to automatically rotate to ZRT coords.

Returns None, curation operates directly on data_all

seismic.inversion.wavefield_decomp.runners.**load_mcmc_solution**(*h5_file*, *job_timestamp=None*, *logger=None*)

Load Monte Carlo Markov Chain solution from HDF5 file.

Parameters

- **h5_file** – File from which to load solution
- **job_timestamp** – Timestamp of job whose solution is to be loaded
- **logger** – Output logging instance

Returns (solution, job configuration), job timestamp

seismic.inversion.wavefield_decomp.runners.**mcmc_solver_wrapper**(*model*, *obj_fn*, *mantle*, *Vp*, *rho*, *flux_window*)

Wrapper callable for passing to MCMC solver in scipy style, which unpacks inputs into vector variables for solver.

Parameters

- **model** – Per-layer model values (the vector being solved for) as flat array of (H, Vs) value pairs ordered by layer.
- **obj_fn** – Callable to WfContinuationSuFluxComputer to compute SU flux.
- **mantle** – Mantle properties stored in class LayerProps instance.
- **Vp** – Array of Vp values ordered by layer.
- **rho** – Arroy of rho values ordered by layer.
- **flux_window** – Pair of floats indicating the time window over which to perform SU flux integration

Returns Integrated SU flux energy at top of mantle

seismic.inversion.wavefield_decomp.runners.**run_mcmc**(*waveform_data*, *config*, *logger*)

Top level runner function for MCMC solver on SU flux minimization for given settings.

Parameters

- **waveform_data** – Iterable container of obspy.Stream objects.
- **config** – Dict of job settings. See example files for fields and format of settings.
- **logger** – Log message receiver. Optional, pass None for no logging.

Returns Solution object based on scipy.optimize.OptimizeResult

seismic.inversion.wavefield_decomp.runners.**run_station**(*config_file*, *waveform_file*, *network*, *station*, *location*, *logger*)

Runner for analysis of single station. For multiple stations, set up config file to run batch job using mpi_job CLI.

The output file is in HDF5 format. The configuration details are added to the output file for traceability.

Parameters

- **config_file** – Config filename specifying job settings
- **waveform_file** – Event waveform source file for seismograms, generated using extract_event_traces.py script
- **network** – Network code of station to analyse
- **station** – Station code to analyse
- **location** – Location code of station to analyse. Can be '' (empty string) if not set.
- **logger** – Output logging instance

Returns Pair containing (solution, configuration) containers. Configuration will have additional traceability information.

seismic.inversion.wavefield_decomp.runners.**save_mcmc_solution**(*soln_configs*, *input_file*, *output_file*, *job_timestamp*, *job_tracking*, *logger=None*)

Save solution to HDF5 file.

Parameters

- **soln_configs** – List of (solution, configuration) pairs to save (one per station).
- **input_file** – Name of input file. Saved to job node for traceability
- **output_file** – Name of output file to create
- **job_timestamp** – Job timestamp that will be used to generate the top level job group
- **job_tracking** – Dict containing job identification information for traceability
- **logger** – [OPTIONAL] Log message destination

Returns None

seismic.inversion.wavefield_decomp.solvers module

Objective function minimization solvers.

class seismic.inversion.wavefield_decomp.solvers.**AdaptiveStepsize**(*stepper*, *accept_rate=0.5*, *interval=50*, *factor=0.9*, *ar_tolerance=0.05*)

Bases: object

Class to implement adaptive stepsize. Pulled from scipy.optimize.basinhopping and adapted.

notify_accept()

take_step(*x*)

class seismic.inversion.wavefield_decomp.solvers.**BoundedRandNStepper**(*bounds*, *initial_step=None*)

Bases: object

Step one dimensional at a time using normal distribution of steps within defined parameter bounds.

property stepsize

class seismic.inversion.wavefield_decomp.solvers.**HistogramIncremental**(*bounds*, *nbins=20*)

Bases: object

Class to incrementally accumulate N-dimensional histogram stats at runtime.

property bins

property dims

property histograms

class seismic.inversion.wavefield_decomp.solvers.**SolverGlobalMhMcmc**

Bases: object

Drop-in custom solver for scipy.optimize.minimize, based on Metrolpolis-Hastings Monte Carlo Markov Chain random walk with burn-in and adaptive acceptance rate, followed by N-dimensional clustering.

Rather than returning one global solution, the solver returns the N best ranked local solutions. It also returns a probability distribution of each unknown based on Monte Carlo statistics.

seismic.inversion.wavefield_decomp.solvers.**optimize_minimize_mhmcmc_cluster**(*objective*, *bounds*, *args=()*, *x0=None*, *T=1*, *N=3*, *burnin=100000*, *maxiter=1000000*, *target_ar=0.4*, *ar_tolerance=0.05*, *cluster_eps=0.05*, *rnd_seed=None*, *collect_samples=None*, *logger=None*)

Minimize objective function and return up to N local minima solutions.

Parameters

- **objective** – Objective function to minimize
- **bounds** – Bounds of the parameter space.
- **args** – Any additional fixed parameters needed to completely specify the objective function.
- **x0** – Initial guess. If None, will be selected randomly and uniformly within the parameter bounds.
- **T** – The "temperature" parameter for the accept or reject criterion. To sample the domain well, should be in the order of the typical difference in local minima objective valuations.
- **N** – Maximum number of minima to return
- **burnin** – Number of random steps to discard before starting to accumulate statistics.
- **maxiter** – Maximum number of steps to take (including burnin).
- **target_ar** – Target acceptance rate of point samples generated by stepping.
- **ar_tolerance** – Tolerance on the acceptance rate before actively adapting the step size.
- **cluster_eps** – Point proximity tolerance for DBSCAN clustering, in normalized bounds coordinates.
- **rnd_seed** – Random seed to force deterministic behaviour
- **collect_samples** – If not None and integral type, collect collect_samples at regular intervals and return in solution.
- **logger** – Logger instance for outputting log messages.

Returns OptimizeResult containing solution(s) and solver data.

seismic.inversion.wavefield_decomp.wavefield_continuation_tao module

Class encapsulating algorithm for 1D inversion using wavefield continuation.

Based on reference: Kai Tao, Tianze Liu, Jieyuan Ning, Fenglin Niu, "Estimating sedimentary and crustal structure using wavefield continuation: theory, techniques and applications", *Geophysical Journal International*, Volume 197, Issue 1, April, 2014, Pages 443-457, https://doi.org/10.1093/gji/ggt515

class seismic.inversion.wavefield_decomp.wavefield_continuation_tao.**WfContinuationSuFluxCo**

Bases: object

Implements computation of the upwards mean S-wave energy flux at the top of the mantle for an ensemble of events for one station.

Class instance can be loaded with a dataset and then evaluated for arbitrary 1D earth models.

Process:

1. **Load data from a station event dataset. Copy of the data is buffered by this** class in efficient format for energy flux calculation.
2. Define 1D earth model and mantle half-space material properties (in external code).
3. Call instance with models and receive energy flux results.

grid_search(*mantle_props*, *layer_props*, *layer_index*, *H_vals*, *k_vals*, *flux_window=- 10, 20*, *ncpus=- 1*)

Compute SU energy flux over a grid of H and k values for a particular (single) layer. The layer to compute over is indicated by layer_index, which is a zero-based index into layer_props.

Parameters

- **mantle_props** (*LayerProps*) – Mantle bulk properties
- **layer_props** (*list of LayerProps*) – Layer bulk properties as a list
- **layer_index** (*int*) – Index of layer to vary.
- **H_vals** (*numpy.array*) – Set of thickness (H) values to apply to the selected layer.
- **k_vals** (*numpy.array*) – Set of k values to apply to the selected layer.
- **flux_window** (*pair of numeric values*) – Time window in which to compute energy flux
- **ncpus** (*int*) – Number of CPUs to use. Use -1 to use all.

Returns tuple(H grid, k grid, energy values)

Return type tuple(numpy.array, numpy.array, numpy.array)

propagate_to_base(*layer_props*)

Given a stack of layers in layer_props, propagate the surface seismograms down to the base of the bottom layer.

Parameters **layer_props** – List of LayerProps through which to propagate the surface seismograms.

Returns (Vr, Vz) time series per event at the bottom of the stack of layers.

times()

seismic.inversion.wavefield_decomp.wfd_plot module

Plotting helper functions for Wavefield Decomposition module.

seismic.inversion.wavefield_decomp.wfd_plot.**plot_Esu_space**(*H*, *k*, *Esu*, *title=None*, *savefile_name=None*, *show=True*, *c_range=None, None*, *decorator=None*)

Plot SU energy as function of H-k.

Parameters

- **H** – Grid of H coordinates corresponding to Esu point values.
- **k** – Grid of k coordinates corresponding to Esu point values.
- **Esu** – Energy values on H-k grid.
- **title** – Plot title [OPTIONAL]
- **savefile_name** – Output file in which to save plot [OPTIONAL]
- **show** – If True, display the image and block until it is closed.
- **c_range** – Custom range of Esu to contour (min, max values)
- **decorator** – Callback function to customize plot.

Returns None

seismic.inversion.wavefield_decomp.wfd_plot.**plot_Nd**(*soln*, *title=''*, *scale=1.0*, *vars=None*)

Plotting routine for N-dimensional solution in grid format. Diagonal contains histograms of solution samples for each variable, and off-diagonal contains pair-wise scatter plots of sample points plus solution clusters. The distinct solutions coordinates are colour coded and labelled on the histograms.

This function is intended to be a domain-agnostic means of plotting an N-dimensional solution generated by optimize_minimize_mhmcmc_cluster(). It should not have seismology-specific fields or terminology added into it.

Parameters

- **soln** (*scipy.optimize.OptimizeResult with additional custom fields*) – Solution container returned from optimize_minimize_mhmcmc_cluster()
- **title** (*str, optional*) – Overall plot title, defaults to '' (no title)
- **scale** (*float, optional*) – Scale size of overall plot, defaults to 1.0. Adjust this depending on dimensionality or desired size.
- **vars** (*list(str)*) – List of names of variables, should be same length as dimensionality of solution

Returns Tuple containing the PairGrid, list of axes for the secondary (histogram) axes in the diagonal of the grid, and list of text items representing solution labels.

Return type tuple(seaborn.PairGrid, list(matplotlib.axes.Axes), list(matplotlib.text.Text))

1.5.2 Module contents

1.6 seismic.pick_harvester package

1.6.1 Submodules

1.6.2 seismic.pick_harvester.createEnsembleXML module

Description: This script was initially written for inserting new picks into the ISC catalogue. We now use a unified csv catalogue (that Babak has prepared) and this script merges existing picks with those picked by our parallel picker and creates self-consistent SC3ML files to be ingested into Seiscomp3.

CreationDate: 20/11/18 Developer: rakib.hassan@ga.gov.au

Revision History: LastUpdate: 20/11/18 RH

class seismic.pick_harvester.createEnsembleXML.**Arrival**(*net*, *sta*, *loc*, *cha*, *lon*, *lat*, *elev*, *phase*, *utctime*, *distance*)

Bases: object

cha

distance

elev

lat

loc

lon

net

phase

sta

utctime

class seismic.pick_harvester.createEnsembleXML.**Catalog**(*isc_coords_file*, *fdsn_inventory*, *our_picks*, *event_folder*, *output_path*, *discard_old_picks=False*)

Bases: object

get_id()

class seismic.pick_harvester.createEnsembleXML.**Event**

Bases: object

origin_list

preferred_magnitude

preferred_origin

public_id

class seismic.pick_harvester.createEnsembleXML.**FDSNInv**(*fn*, *host=None*, *port=None*)
Bases: object

getClosestStation(*lon*, *lat*, *maxdist=1000.0*)

rtp2xyz(*r*, *theta*, *phi*)

class seismic.pick_harvester.createEnsembleXML.**Magnitude**(*mag*, *mag_type*)
Bases: object

magnitude_type

magnitude_value

class seismic.pick_harvester.createEnsembleXML.**Origin**(*utctime*, *lat*, *lon*, *depthkm*)
Bases: object

arrival_list

depthkm

lat

lon

magnitude_list

utctime

class seismic.pick_harvester.createEnsembleXML.**OurPicks**(*fnList*, *phaseList*)
Bases: object

seismic.pick_harvester.createEnsembleXML.**setup_logger**(*name*, *log_file*, *level=20*)
Function to setup a logger; adapted from stackoverflow

1.6.3 seismic.pick_harvester.pick module

Description: Harvests picks from ASDF data-sets in parallel

References:

CreationDate: 13/09/18 Developer: rakib.hassan@ga.gov.au

Revision History: LastUpdate: 13/09/18 RH LastUpdate: dd/mm/yyyy Who Optional description

seismic.pick_harvester.pick.**dropBogusTraces**(*st*, *sampling_rate_cutoff=5*)

seismic.pick_harvester.pick.**extract_p**(*taupy_model*, *pickerlist*, *event*, *station_longitude*, *station_latitude*, *st*, *win_start=- 50*, *win_end=50*, *resample_hz=20*, *bp_freqmins=[0.5, 2.0, 5.0]*, *bp_freqmaxs=[5.0, 10.0, 10.0]*, *margin=None*, *max_amplitude=100000000.0*, *plot_output_folder=None*)

seismic.pick_harvester.pick.**extract_s**(*taupy_model*, *pickerlist*, *event*, *station_longitude*, *station_latitude*, *stn*, *ste*, *ba*, *win_start=- 50*, *win_end=50*, *resample_hz=20*, *bp_freqmins=[0.01, 0.01, 0.5]*, *bp_freqmaxs=[1, 2.0, 5.0]*, *margin=None*, *max_amplitude=100000000.0*, *plot_output_folder=None*)

seismic.pick_harvester.pick.**getWorkloadEstimate**(*fds*, *originTimestamps*)

seismic.pick_harvester.pick.**merge_results**(*output_path*)

1.6.4 seismic.pick_harvester.quality module

Description: Computes quality measures of picks using various methods

References:

CreationDate: 24/01/19

Developer: rakib.hassan@ga.gov.au

Revision History: LastUpdate: 24/01/19 RH LastUpdate: dd/mm/yyyy Who Optional description

seismic.pick_harvester.quality.**compute_quality_measures**(*trc*, *trc_filtered*, *scales*, *plotinfo=None*)

Computes quality measures for a given pick based on:

1. wavelet transforms
2. waveform complexity analysis (similar to Higuchi fractal dimensions)

Parameters

- **trc** – raw obspy trace centred on pick-time
- **trc_filtered** – filtered obspy trace centred on pick-time
- **scales** – scales for computing continuous wavelet transforms
- **plotinfo** – dictionary containing required plotting information (eventid, origintime, mag, net, sta, phase, ppsnr, pickid, outputfolder)

Returns

1. cwtsnr: quality measure based on wavelet analysis
2. dom_freq: dominant frequency of arrival energy
3. slope_ratio: quality measure based on waveform complexity analysis

1.6.5 seismic.pick_harvester.utils module

class seismic.pick_harvester.utils.**Catalog**(*event_folder*)

Bases: object

get_events()

get_preferred_origin_timestamps()

class seismic.pick_harvester.utils.**CatalogCSV**(*event_folder*)

Bases: object

get_events()

get_preferred_origin_timestamps()

class seismic.pick_harvester.utils.**Event**

Bases: object

class seismic.pick_harvester.utils.**EventParser**(*xml_filename*)

Bases: object

getEvents()

parseEvent(*e*)

parseMagnitude(*m*)

parseOrigin(*o*)

class seismic.pick_harvester.utils.**Magnitude**(*mag*, *mag_type*)
Bases: object

class seismic.pick_harvester.utils.**Origin**(*utctime*, *lat*, *lon*, *depthkm*)
Bases: object

class seismic.pick_harvester.utils.**ProgressTracker**(*output_folder*, *restart_mode=False*)
Bases: object

increment()

seismic.pick_harvester.utils.**recursive_glob**(*treeroot*, *pattern*)

seismic.pick_harvester.utils.**split_list**(*lst*, *npartitions*)

1.6.6 Module contents

1.7 seismic.receiver_fn package

1.7.1 Submodules

1.7.2 seismic.receiver_fn.bulk_rf_report module

Produce PDF report of network stations showing RF waveforms

1.7.3 seismic.receiver_fn.generate_rf module

Generate Receiver Functions (RF) from collection of 3-channel seismic traces.

seismic.receiver_fn.generate_rf.**event_waveforms_to_rf**(*input_file*, *output_file*, *config*)
Main entry point for generating RFs from event traces.

Config file consists of 3 sub-dictionaries. One named "filtering" for input stream filtering settings, one named "processing" for RF processing settings, and one named "system" for options on how the system will run the job. Each of these sub-dicts is described below:

```
"filtering":  # Filtering settings
{
  "resample_rate": float # Resampling rate in Hz
  "taper_limit": float   # Fraction of signal to taper at end, between 0 and 0.5
  "filter_band": (float, float) # Filter pass band (Hz). Not required for freq-
↪domain deconvolution.
  "channel_pattern": # Ordered list of preferred channels, e.g. 'HH*,BH*',
                     # where channel selection is ambiguous.
  "baz_range": (float, float) or [(float, float), ...] # Discrete ranges of␣
↪source back azimuth to use (degrees).
      # Each value must be between 0 and 360. May be a pair or a list of pairs␣
↪for multiple ranges.
}

"processing":  # RF processing settings
{
  "custom_preproc":
```

(continues on next page)

(continued from previous page)

```
  {
    "import": 'import custom symbols',  # statement to import required symbols
    "func": 'preproc functor'  # expression to get handle to custom preprocessing␣
↪functor
    "args": {}  # additional kwargs to pass to func
  }
  "trim_start_time": float # Trace trim start time in sec, relative to onset
  "trim_end_time": float # Trace trim end time in sec, relative to onset
  "rotation_type": str # Choice of ['zrt', 'lqt']. Rotational coordinate system
                       # for aligning ZNE trace components with incident wave␣
↪direction
  "deconv_domain": str # Choice of ['time', 'freq', 'iter']. Whether to perform␣
↪deconvolution
                       # in time or freq domain, or iterative technique
  "gauss_width": float # Gaussian freq domain filter width. Only required for␣
↪freq-domain deconvolution
  "water_level": float # Water-level for freq domain spectrum. Only required for␣
↪freq-domain deconvolution
  "spiking": float # Spiking factor (noise suppression), only required for time-
↪domain deconvolution
  "normalize": bool # Whether to normalize RF amplitude
}

"system":  # job run settings
{
  "parallel": bool # Use parallel execution
  "memmap": bool # Memmap input file for improved performance in data reading␣
↪thread.
                 # Useful when data input is bottleneck, if system memory permits.
  "temp_dir": str or path # Temporary directory to use for best performance
  "aggressive_dispatch": bool # Dispatch all worker jobs as aggressively as␣
↪possible to minimize
                              # chance of worker being starved of work. Uses more␣
↪memory.
}
```

Parameters

- **input_file** (*str or pathlib.Path*) – Event waveform source file for seismograms, generated using extract_event_traces.py script
- **output_file** (*str or pathlib.Path*) – Name of hdf5 file to produce containing RFs
- **config** (*dict*) – Dictionary containing job configuration parameters

Returns None

seismic.receiver_fn.generate_rf.**transform_stream_to_rf**(*ev_id*, *stream3c*, *config_filtering*, *config_processing*, ***kwargs*)

Generate P-phase receiver functions for a single 3-channel stream. See documentation for function event_waveforms_to_rf for details of config dictionary contents.

Parameters

- **ev_id** (*int or str*) – The event id
- **stream3c** (*rf.RFStream*) – Stream with 3 components of trace data

- **config_filtering** (*dict*) – Dictionary containing stream filtering settings
- **config_processing** (*dict*) – Dictionary containing RF processing settings
- **kwargs** (*dict*) – Keyword arguments that will be passed to filtering and deconvolution functions.

Returns RFstream containing receiver function if successful, None otherwise

Return type rf.RFStream or NoneType

seismic.receiver_fn.generate_rf.**transform_stream_to_rf_queue**(*oqueue*, *ev_id*, *stream3c*, *config_filtering*, *config_processing*, ***kwargs*)

Process using transform_stream_to_rf and queue results for further handling.

Parameters

- **oqueue** (*queue or multiprocessing.Manager.Queue*) – Output queue where filtered streams are queued
- **ev_id** – Event ID to pass on to transform_stream_to_rf
- **stream3c** – 3-channel stream to process
- **config_filtering** – Filtering settings
- **config_processing** – Processing settings
- **kwargs** (*dict*) – Keyword arguments that will be passed to filtering and deconvolution functions.

Returns True if stream containing receiver function is pushed into output queue oqueue, False otherwise

Return type bool

1.7.4 seismic.receiver_fn.plot_ccp module

Generate common conversion point (CCP) plot as per *C.Sippl, "Moho geometry along a north-south passive seismic transect through Central Australia", Technophysics 676 (2016), pp.56-69, DOI https://doi.org/10.1016/j.tecto.2016.03.031*

This code adapted from Christian Sippl's original code.

Workflow: extract_event_traces.py –> generate_rf.py –> rf_quality_filter.py –> plot_ccp.py (this script)

Example usage: python seismic/receiver_fn/plot_ccp.py –start-latlon -19.5 133.0 –end-latlon -19.5 140.0 –width 120 –channels T –stacked-scale 0.3 –title "Network OA CCP T-stacking (profile BS24-CF24)" /software/hiperseis/seismic/receiver_fn/DATA/OA-ZRT-cleaned.h5 /software/hiperseis/seismic/receiver_fn/DATA/OA-ZRT-T_CCP_stack_BS24-CF24_2km_spacing.png

seismic.receiver_fn.plot_ccp.**add_ccp_trace**(*trace*, *inc_p*, *matrx*, *matrx_entry*, *vmod*, *depstep*, *lenstep*, *sta_offset*, *az*)

project amplitudes from all RFs onto the profile... 2D rot:

seismic.receiver_fn.plot_ccp.**angular_distance**(*p1*, *p2*)

Compute the angular distance (in degrees) between two points p1 and p2 using Haversine formula.

Math reference: https://www.movable-type.co.uk/scripts/latlong.html

Parameters

- **p1** (*tuple(float, float)*) – (latitude, longitude) in degrees
- **p2** (*tuple(float, float)*) – (latitude, longitude) in degrees

seismic.receiver_fn.plot_ccp.**bearing**(*p1*, *p2*)
Compute bearing (forward azimuth) in degrees from p1 to p2.

Math reference: https://www.movable-type.co.uk/scripts/latlong.html

Parameters

- **p1** (*tuple(float, float)*) – (latitude, longitude) in degrees
- **p2** (*tuple(float, float)*) – (latitude, longitude) in degrees

seismic.receiver_fn.plot_ccp.**bounding_box**(*startpoint*, *endpoint*)
Compute a bounding box from start and end points.

Parameters

- **startpoint** (*pair of float*) – Coordinates of starting point
- **endpoint** (*pair of float*) – Coordinates of end point

Returns Bounding box (left, bottom, right, top)

Return type tuple(float, float, float, float)

seismic.receiver_fn.plot_ccp.**ccp_compute_station_params**(*rf_stream*, *startpoint*, *endpoint*, *width*, *bm=None*)

Determines which stations are between startpoint and endpoint great circle profile line, and within *width* distance of that profile line. Generates a dictionary of distance along profile line (*sta_offset*) and orthogonal distance from profile line (*dist*). For stations outside the region of interest, dictionary has the value None.

Parameters

- **rf_stream** (*rf.rfstream.RFStream*) – RFStreams whose stations will be processed.
- **startpoint** (*tuple(float, float)*) – (latitude, longitude) in degrees of the profile line start point.
- **endpoint** (*tuple(float, float)*) – (latitude, longitude) in degrees of the profile line end point.
- **width** (*float*) – Width of ROI in km to either side of the profile line.
- **bm** (*mpl_toolkits.basemap.Basemap*) – Optional basemap upon which to plot stations coded according to whether they are within the ROI.

Returns Dictionary keyed by station code containing distances relative to profile line, or None if station is outside the ROI.

Return type dict

seismic.receiver_fn.plot_ccp.**ccp_generate**(*rf_stream*, *startpoint*, *endpoint*, *width*, *spacing*, *max_depth*, *channels=None*, *v_background='ak135'*, *station_map=True*)

Main function for processing RF collection and plotting common conversion point (CCP) stack of receiver functions (RFs) along a specific line between startpoint and endpoint.

Parameters

- **rf_stream** (*rf.rfstream.RFStream*) – Sequence of receiver function traces

- **startpoint** (*tuple(float, float)*) – Starting point for the transect line in latitude, longitude degrees
- **endpoint** (*tuple(float, float)*) – End point for the transect line in latitude, longitude degrees
- **width** (*float*) – Width (km) of region about the transect line from which to project RFs to the slice.
- **spacing** (*float*) – Size (km) of each discrete sample cell in the spatial slice beneath the transect.
- **max_depth** (*float*) – Maximum depth (km) of the slice beneath the transect
- **channels** (*list(str), optional*) – Filter for channels, defaults to None in which case only 'R' channel is used. Allowed values are in ['Z', 'R', 'T'].
- **v_background** (*str, optional*) – Assumed background 1D velocity model, defaults to 'ak135'
- **station_map** (*bool*) – Whether to generate figure for station map, defaults to True

Returns Normalized stack matrix, normalization factors, profile length, station metadata, map figure

Return type numpy.array, numpy.array, float, dict, matplotlib.pyplot.Figure

seismic.receiver_fn.plot_ccp.**cross_along_track_distance**(*p1*, *p2*, *p3*)

Compute the cross-track distance and the along-track distance of p3 in relation to great circle from p1 to p2.

Math reference: https://www.movable-type.co.uk/scripts/latlong.html

Parameters

- **p1** (*tuple(float, float)*) – (latitude, longitude) in degrees
- **p2** (*tuple(float, float)*) – (latitude, longitude) in degrees
- **p3** (*tuple(float, float)*) – (latitude, longitude) in degrees

Returns Cross-track distance (orthogonal to arc from p1 to p2) and along-track distance (along arc from p1 to p2) of the location p3.

Return type tuple(float, float)

seismic.receiver_fn.plot_ccp.**equirectangular_projection**(*x0*, *y0*, *x1*, *y1*)

Perform equirectangular projection of a pair of latitude, longitude coordinates to cartesian coordinates.

This length calculation uses the forward equirectangular projection (https://en.wikipedia.org/wiki/Equirectangular_projection). (See also https://www.movable-type.co.uk/scripts/latlong.html)

Parameters

- **x0** (*float*) – Point 0 longitude (deg)
- **y0** (*float*) – Point 0 latitude (deg)
- **x1** (*float*) – Point 1 longitude (deg)
- **y1** (*float*) – Point 1 latitude (deg)

Returns Lengths of sides of rectangle and the diagonal. The diagonal is the distance between points 0 and 1.

Return type float, float, float

seismic.receiver_fn.plot_ccp.**get_amplitude**(*trace*, *time*)

retrieve amplitude value

seismic.receiver_fn.plot_ccp.**matrx_lookup**(*xsz*, *sta_offset*, *h*, *depstep*, *lenstep*)
return index values for amplitude contrbution in profile matrix

seismic.receiver_fn.plot_ccp.**plot_ccp**(*matrx*, *length*, *max_depth*, *spacing*, *vlims=None*, *metadata=None*, *title=None*, *colormap='seismic'*)
Plot results of CCP stacking procedure.

Parameters

- **matrx** (*numpy.array*) – [description]
- **length** (*float*) – Length (km) of the transect line
- **max_depth** (*float*) – Maximum depth (km) of the slice to plot
- **spacing** (*float*) – Grid spacing (km)
- **vlims** (*tuple(float, float), optional*) – Min and max values for the color scale, defaults to None
- **metadata** (*dict, optional*) – Metadata generated by ccp_compute_station_params(), defaults to None
- **title** (*str, optional*) – Title text to add to the plot, defaults to None
- **colormap** (*str*) – Color map to use for the CCP intensity shading.

Returns Figure handle

Return type matplotlib.pyplot.Figure

seismic.receiver_fn.plot_ccp.**run**(*rf_stream*, *start_latlon*, *end_latlon*, *width*, *spacing*, *max_depth*, *channels*, *background_model='ak135'*, *stacked_scale=None*, *title=None*, *colormap=None*)
Run CCP generation on a given dataset of RFs.

Parameters

- **rf_stream** (*rf.RFStream*) – Set of RFs to use for CCP plot
- **start_latlon** (*tuple(float, float)*) – Starting (latitude, longitude) coordinates of line transect
- **end_latlon** (*tuple(float, float)*) – End (latitude, longitude) coordinates of line transect
- **width** (*float*) – Width of transect (km)
- **spacing** (*float*) – Discretization size (km) for RF ray sampling
- **max_depth** (*float*) – Maximum depth of slice below the transect line (km)
- **channels** (*str, comma separated*) – String of comma-separated component IDs to source for the RF amplitude
- **background_model** (*str*) – 1D background model to assume
- **stacked_scale** (*float*) – Max value to represent on color scale of CCP plot
- **title** (*str*) – Title to place at top of CCP plot
- **colormap** (*str*) – Color map to use for the CCP intensity shading. Suggest 'seismic', 'coolwarm' or 'jet'.

Returns Figure handles: Main figure, map figure, dict of station parameters

Return type (matplotlib.pyplot.Figure, matplotlib.pyplot.Figure, dict)

seismic.receiver_fn.plot_ccp.**setup_ccp_profile**(*length*, *spacing*, *maxdep*)
Construct the grid for a CCP stacking profile

Parameters

- **length** (*float*) – Length (km) of the profile
- **spacing** (*float*) – Grid spacing (km)
- **maxdep** (*float*) – Maximum depth (km) of the slice to plot

Returns Zeroed matrix and mesh coordinates

Return type numpy.array, numpy.array, numpy.array

1.7.5 seismic.receiver_fn.plot_ccp_batch module

seismic.receiver_fn.plot_ccp_batch.**gravity_subplot**(*hf*, *metadata*, *grav_map*)
Custom plot modifier to add a gravity subplot along the CCP transect line.

Parameters

- **hf** (*matplotlib.pyplot.Figure*) – Figure containing main CCP plot
- **metadata** (*dict*) – Dictionary from CCP plotting code containing metadata of the transect and station data
- **grav_map** (*Instance of 2D interpolation class from scipy.interpolator*) – Interpolator callable that takes a iterable of 2D coordinates and interpolates a gravity value at each, based on external data source.

seismic.receiver_fn.plot_ccp_batch.**moho_annotator**(*hf*, *metadata*)
Custom plot annotator to add markers to the main figure showing locations of Moho estimates from other sources.

Parameters

- **hf** (*matplotlib.pyplot.Figure*) – Figure containing main CCP plot
- **metadata** (*dict*) – Dictionary from CCP plotting code containing metadata of the transect and station data

seismic.receiver_fn.plot_ccp_batch.**run_batch**(*transect_file*, *rf_waveform_file*, *fed_db_file*, *amplitude_filter=False*, *similarity_filter=False*, *stack_scale=0.4*, *width=30.0*, *spacing=2.0*, *max_depth=200.0*, *channel='R'*, *output_folder=''*, *colormap='seismic'*, *annotators=None*)
Run CCP generation in batch mode along a series of transects.

Parameters

- **transect_file** (*str or Path*) – File containing specification of network and station locations of ends of transects
- **rf_waveform_file** (*str or Path*) – HDF5 file of QA'd receiver functions for the network matching the transect file
- **fed_db_file** (*str or Path*) – Name of file with which to initialize FederatedASDFDataBase

- **amplitude_filter** (*bool*) – Whether to use amplitude-based filtering of waveforms beform plotting.
- **similarity_filter** (*bool*) – Whether to use RF waveform similarity filtering of waveforms beform plotting.
- **stack_scale** (*float*) – Max value to represent on color scale of CCP plot
- **width** (*float*) – Width of transect (km)
- **spacing** (*float*) – Discretization size (km) for RF ray sampling
- **max_depth** (*float*) – Maximum depth of slice below the transect line (km)
- **channel** (*str length 1*) – Channel component ID to source for the RF amplitude

Returns None

1.7.6 seismic.receiver_fn.plot_spatial_map module

Use cartopy to plot point dataset onto map.

seismic.receiver_fn.plot_spatial_map.**plot_spatial_map**(*point_dataset*, *projection_code*, *title=None*, *feature_label=None*)

Make spatial plot of point dataset with filled contours overlaid on map.

Parameters

- **point_dataset** – Name of point dataset file. Should be in format produced by script *pointsets2grid.py*
- **projection_code** – EPSG projection code, e.g. 3577 for Australia
- **title** – Title string for top of plot
- **feature_label** – Label for the color bar of the plotted feature

Returns

1.7.7 seismic.receiver_fn.pointsets2grid module

Blend multiple datasets on different lon/lat point sets together onto a common grid.

Uses weighted Gaussian interpolation method of Kennett, but simplified to ignore the individual point weighting.

Multiple datasets with per-dataset settings are passed in using a JSON configuration file with layout illustrated by the following example:

```
{
    "source_files":
    {
        "file1":
        {
            "weighting": 2.5,
            "scale_length_km": 10.0,
            "scale_length_cutoff": 3.5
        },
        "file2":
        {
            "weighting": 0.5,
```

(continues on next page)

(continued from previous page)

```
            "scale_length_km": 20.0
        },
        "file3":
        {
            "weighting": 1.0,
            "scale_length_km": 10.0,
            "scale_length_cutoff": 3
        }
    },
    "proj_crs_code": 3577,
    "output_spacing_km": 10.0
}
```

Reference: B. L. N. Kennett 2019, "Areal parameter estimates from multiple datasets", Proc. R. Soc. A. 475:20190352, http://dx.doi.org/10.1098/rspa.2019.0352

Requires: - pyepsg - cartopy

1.7.8 seismic.receiver_fn.rf_3dmigrate module

Description:

Implements migration algorithm as described in Frassetto et al. (2010): Improved imaging with phase-weighted common conversion point stacks of receiver functions (GJI)

References:

CreationDate: 3/15/18 Developer: rakib.hassan@ga.gov.au

Revision History: LastUpdate: 3/15/18 RH LastUpdate: dd/mm/yyyy Who Optional description

class seismic.receiver_fn.rf_3dmigrate.**Geometry**(*start_lat_lon*, *azimuth*, *lengthkm*, *nx*, *widthkm*, *ny*, *depthkm*, *nz*, *debug=False*)

Bases: object

generateGrids()

class seismic.receiver_fn.rf_3dmigrate.**Migrate**(*geometry*, *stream*, *debug=False*, *output_folder='/tmp'*)

Bases: object

execute()

seismic.receiver_fn.rf_3dmigrate.**rtp2xyz**(*r*, *theta*, *phi*)

Convert spherical to cartesian coordinates

Parameters

- **r** ([type]) – [description]
- **theta** ([type]) – [description]
- **phi** ([type]) – [description]

Returns [description]

Return type [type]

seismic.receiver_fn.rf_3dmigrate.**xyz2rtp**(*x*, *y*, *z*)

Convert cartesian to spherical coordinates

Parameters

- **x** ([type]) – [description]
- **y** ([type]) – [description]
- **z** ([type]) – [description]

Returns [description]

Return type [type]

1.7.9 seismic.receiver_fn.rf_deconvolution module

Custom receiver function deconvolution methods and support functions.

seismic.receiver_fn.rf_deconvolution.**iter_deconv_pulsetrain**(*numerator*, *denominator*, *sampling_rate*, *time_shift*, *max_pulses=1000*, *tol=0.001*, *gwidth=2.5*, *only_positive=False*, *log=None*, ***kwargs*)

Iterative deconvolution of source and response signal to generate seismic receiver function. Adapted to Python by Andrew Medlin, Geoscience Australia (2019), from Chuck Ammon's (Saint Louis University) *iterdeconfd* Fortran code, version 1.04.

Note this is not really a frequency-domain deconvolution method, since the deconvolution is based on generating a time-domain pulse train which is filtered and convolved with source signal in time domain in order to try to replicate observation. Results should be the same even if some of the spectral techniques used in functions (such as *gauss_filter*) were replaced by non-spectral equivalents.

Parameters

- **numerator** (*numpy.array(float)*) – The observed response signal (e.g. R, Q or T component)
- **denominator** (*numpy.array(float)*) – The source signal (e.g. L or Z component)
- **sampling_rate** (*float*) – The sampling rate in Hz of the numerator and denominator signals
- **time_shift** (*float*) – Time shift (sec) from start of input signals until expected P-wave arrival onset.
- **max_pulses** (*int*) – Maximum number of delta function pulses to synthesize in the unfiltered RF (up to 1000)
- **tol** (*float*) – Convergence tolerance, iteration stops if change in error falls below this value
- **gwidth** (*float*) – Gaussian filter width in the normalized frequency domain.
- **only_positive** (*bool*) – If true, only use positive pulses in the RF.
- **log** (*logging.Logger*) – Log instance to log messages to.

Returns RF trace, pulse train, expected response signal, predicted response signal, quality of fit statistic

Return type numpy.array(float), numpy.array(float), numpy.array(float), numpy.array(float), float

seismic.receiver_fn.rf_deconvolution.**rf_iter_deconv**(*response_data*, *source_data*, *sr*, *tshift*, *min_fit_threshold=80.0*, *normalize=0*, ***kwargs*)

Adapter function to rf library. To use, add arguments *deconvolve='func', func=rf_iter_deconv* to *rf.RFStream.rf()* function call.

Parameters

- **response_data** (*list of numpy.array(float)*) – List of response signals for which to compute receiver function
- **source_data** (*numpy.array(float)*) – Source signal to use for computing the receiver functions
- **sampling_rate** (*float*) – Sampling rate of the input signals (Hz)
- **time_shift** (*float*) – Time shift (seconds) from start of signal to onset
- **min_fit_threshold** (*float*) – Minimum percentage of fit to include trace in results, otherwise will be returned as empty numpy array.
- **normalize** (*int or None*) – Component of stream to use for normalization, usually component 0 (the vertical component). Set to None to disable normalization.

Returns Receiver functions corresponding to the list of input response signals.

Return type list of numpy.array(float)

1.7.10 seismic.receiver_fn.rf_h5_file_event_iterator module

Helper class to iterate over 3-channel event traces in h5 file generated by rf library, without loading all the traces into memory. This is a scalable solution for very large files.

class seismic.receiver_fn.rf_h5_file_event_iterator.**IterRfH5FileEvents**(*h5_filename*, *memmap=False*, *channel_pattern=None*)

Bases: object

Helper class to iterate over events in h5 file generated by extract_event_traces.py and pass them to RF generator. This class avoids having to load the whole file up front via obspy which is slow and not scalable.

Due to the expected hierarchy structure of the input H5 file, yielded event traces are grouped by station ID.

Data yielded per event can easily be hundreds of kB in size, depending on the length of the event traces. rf library defaults to window of (-50, 150) sec about the P wave arrival time.

1.7.11 seismic.receiver_fn.rf_h5_file_station_iterator module

Helper class to iterate over station events in h5 file generated by rf library, without loading all the traces into memory. This is a scalable solution for very large files.

class seismic.receiver_fn.rf_h5_file_station_iterator.**IterRfH5StationEvents**(*h5_filename*, *memmap=False*)

Bases: object

Helper class to iterate over stations in h5 file generated by extract_event_traces.py and pass them to RF generator. This class avoids having to load the whole file up front via obspy which is slow and not scalable.

Data yielded per station can easily be many MB in size.

1.7.12 seismic.receiver_fn.rf_handpick_tool module

Simple tool for hand picking functions from RF plots. Selected plots event IDs are exported to text file for later use as RF filter in other tools and workflows.

seismic.receiver_fn.rf_handpick_tool.**main**()
Main entry function for RF picking tool.

seismic.receiver_fn.rf_handpick_tool.**on_release**(*event*, *target_axes*, *select_mask*, *background*, *rect_selector*)
Event handler when mouse button released.

Parameters

- **target_axes** (*matplotlib.axes.Axes*) – The original axes in which the RectangleSelector began. May differ from event.inaxes.
- **event** (*matplotlib.backend_bases.MouseEvent*) – Button up event for end of rectangle area selection
- **select_mask** (*numpy.array(bool)*) – Boolean mask of selected state of each receiver function in the plot
- **background** (*Return type from fig.canvas.copy_from_bbox*) – Background raster from initial render of RFs
- **rect_selector** (*matplotlib.widgets.RectangleSelector*) – Widget for selecting rectangular region to toggle RF selection

seismic.receiver_fn.rf_handpick_tool.**on_select**(*e_down*, *e_up*, *select_mask*)
Event handler for RectangleSelector

Parameters

- **e_down** (*matplotlib.backend_bases.MouseEvent*) – Button down event for start of rectangle
- **e_up** (*matplotlib.backend_bases.MouseEvent*) – Button up event for end of rectangle
- **select_mask** (*numpy.array(bool)*) – Boolean mask of selected state of each receiver function in the plot

1.7.13 seismic.receiver_fn.rf_inversion_export module

Export RFs to file format for external inversion code to run on.

seismic.receiver_fn.rf_inversion_export.**rf_inversion_export**(*input_h5_file*, *output_folder*, *network_code*, *component='R'*, *resample_freq=6.25*, *trim_window=-5.0, 20.0*, *moveout=True*)
Export receiver function to text format for ingestion into Fortran RF inversion code.

Parameters

- **input_h5_file** (*str or Path*) – Input hdf5 file containing receiver function data
- **output_folder** (*str or Path*) – Folder in which to export text files, one per channel per station. Will be appended with network code.
- **network_code** (*str*) – Network to which this RF data belongs, used to disambiguate and track folders.
- **component** (*str, optional*) – The channel component to export, defaults to 'R'
- **resample_freq** (*float, optional*) – Sampling rate (Hz) of the output files, defaults to 6.25 Hz
- **trim_window** (*tuple, optional*) – Time window to export relative to onset, defaults to (-5.0, 20.0). If data needs to be resampled, the samples are anchored to the start of this time window.
- **moveout** (*bool, optional*) – Whether to apply moveout correction prior to exporting, defaults to True

1.7.14 seismic.receiver_fn.rf_network_dict module

Class encapsulating a collection of receiver functions for stations of one network.

class seismic.receiver_fn.rf_network_dict.**NetworkRFDict**(*rf_stream*)

Bases: object

Collection of RFs for a given network indexed by station code, channel code.

keys()

Accessor for the top level keys (station codes) of the network in an iterable container.

Returns Iterable of top level keys to the dictionary

Return type Python iterable

1.7.15 seismic.receiver_fn.rf_plot_utils module

Utility plotting functions for consistent and convenient plotting of RFs.

seismic.receiver_fn.rf_plot_utils.**plot_hk_stack**(*k_grid*, *h_grid*, *hk_stack*, *title=None*, *save_file=None*, *num=None*, *clip_negative=True*)

Plot H-k stack using data generated by function rf_stacking.computed_weighted_stack().

Parameters

- **k_grid** (*Two-dimensional numpy.array*) – Grid of k-values
- **h_grid** (*Two-dimensional numpy.array*) – Grid of H-values
- **hk_stack** (*Two-dimensional numpy.array*) – Grid of stacked RF sample values produced by function rf_stacking.computed_weighted_stack()
- **title** (*str, optional*) – Title to add to the plot, defaults to None
- **save_file** (*str or Path, optional*) – File name in which to save the plot, defaults to None
- **num** (*int, optional*) – Number of RFs used to produce the stack, defaults to None

- **clip_negative** (*bool, optional*) – Clip negative stack regions to zero, defaults to True

Returns Handle to the figure created for the plot

Return type matplotlib.pyplot.figure.Figure

seismic.receiver_fn.rf_plot_utils.**plot_iir_filter_response**(*filter_band_hz*, *sampling_rate_hz*, *corners*)

Plot one-way bandpass filter response in the frequency domain. If filter is used as zero-phase, the attenuation will be twice what is computed here.

Parameters

- **filter_band_hz** (*tuple(float) of length 2 (i.e. pair)*) – Pair of frequencies corresponding to low cutoff and high cutoff freqs
- **sampling_rate_hz** (*float*) – The sampling rate in Hz
- **corners** (*int*) – The order of the filter

Returns Figure object

Return type matplotlib.figure.Figure

seismic.receiver_fn.rf_plot_utils.**plot_iir_impulse_response**(*filter_band_hz*, *sampling_rate_hz*, *corners*, *zero_phase=False*, *N=1000*, *blip_period=1.0*)

Plot bandpass filter response to standard waveforms in the time domain - impulse (delta function, step function, square wave pulse. By default filter is applied one-way. Set *zero_phase=True* to plot two-way filter response.

Parameters

- **filter_band_hz** (*tuple(float) of length 2 (i.e. pair)*) – Pair of frequencies corresponding to low cutoff and high cutoff freqs
- **sampling_rate_hz** (*float*) – The sampling rate in Hz
- **zero_phase** (*bool*) – If True, plot two-way signal response (zero phase), otherwise plot one-way signal response.
- **N** (*int*) – Number of samples in the input test signals.
- **blip_period** (*float*) – Period of the 'blip' test signal (square wave pulse).

seismic.receiver_fn.rf_plot_utils.**plot_rf_stack**(*rf_stream*, *time_window=- 10.0, 25.0*, *trace_height=0.2*, *stack_height=0.8*, *save_file=None*, ***kwargs*)

Wrapper function of rf.RFStream.plot_rf() to help do RF plotting with consistent formatting and layout.

Parameters

- **rf_stream** (*rf.RFStream*) – RFStream to plot
- **time_window** (*tuple, optional*) – Time window to plot, defaults to (-10.0, 25.0)
- **trace_height** (*float, optional*) – Height of a single trace (reduce to cram RFs closer together), defaults to 0.2
- **stack_height** (*float, optional*) – Height of mean (stacked) RF at top of plot, defaults to 0.8

- **save_file** (*str to valid file path, optional*) – File to save resulting image into, defaults to None

seismic.receiver_fn.rf_plot_utils.**plot_rf_wheel**(*rf_stream*, *max_time=15.0*, *deg_per_unit_amplitude=45.0*, *plt_col='C0'*, *title=''*, *figsize=10, 10*, *cluster=True*, *cluster_col='#ff4000'*, *layout=None*, *fontscaling=1.0*)

Plot receiver functions around a polar plot with source direction used to position radial RF plot.

Parameters

- **rf_stream** (*rf.RFStream or list(rf.RFStream)*) – Collection of RFs to plot. If passed as a list, then each stream in the list will be plotted on separate polar axes.
- **max_time** (*float, optional*) – maximum time relative to onset, defaults to 25.0
- **deg_per_unit_amplitude** (*float, optional*) – Azimuthal scaling factor for RF amplitude, defaults to 20
- **plt_col** (*str, optional*) – Plot color for line and positive signal areas, defaults to 'C0'
- **title** (*str, optional*) – Title for the overall plot, defaults to ''
- **figsize** (*tuple, optional*) – Size of figure area, defaults to (12, 12)
- **cluster** (*bool*) – Whether to add overlaid mean RF where there are many RFs close together.
- **cluster_col** (*matplotlib color specification*) – Color of clustered stacked overlay plots.
- **layout** (*tuple(int, int)*) – Arrangement of polar plots in grid. If None, then arranged in a column.

Returns Figure object

Return type matplotlib.figure.Figure

seismic.receiver_fn.rf_plot_utils.**plot_station_rf_overlays**(*db_station*, *title=None*, *time_range=None*)

Plot translucent overlaid RF traces for all traces in each channel, and overplot the mean signal of all the traces per channel.

Parameters

- **db_station** (*dict({str, list(RFTrace)})*) – Dictionary with list of traces per channel for a given station.
- **title** (*str, optional*) – Plot title, defaults to None

Returns Mean trace signal per channel

Return type list(np.array)

1.7.16 seismic.receiver_fn.rf_plot_vespagram module

Plot vespagrams from receiver function data

1.7.17 seismic.receiver_fn.rf_process_io module

Helper for asynchronous writing of RF data to file.

seismic.receiver_fn.rf_process_io.**async_write**(*rfstream_queue*, *outfile_name*, *max_buffered=100*, *metadata=''*)

Monitors asynchronous queue for data, removes from queue to buffer, then flushes buffer intermittently and when queue termination signal is put.

When None is received on the queue, this is taken as the signal to terminate monitoring the queue.

Parameters

- **rfstream_queue** (*multiprocessing.Manager.Queue*) – Queue into which RFStreams are pushed for writing to file.
- **outfile_name** (*str or Path*) – Name of file into which queued RFStream results are periodically written.
- **max_buffered** (*int, optional*) – Maximum number of RFStreams to buffer before flushing to file, defaults to 100
- **metadata** (*str, optional*) – Metadata string to write to root attribute

1.7.18 seismic.receiver_fn.rf_quality_filter module

Filter out invalid RFs, and compute a variety of quality metrics for the remaining RFs. These metrics can be combined in various ways downstream to perform different kinds of filtering. Quality metrics are stored in the stats of each trace.

seismic.receiver_fn.rf_quality_filter.**compute_max_coherence**(*orig_stream*, *f1*, *f2*)

Finding coherence between two signals in frequency domain f1 and f2 - normalised min and max frequencies, f1 < f2 <= ~0.5 returns array of indexes for coherent traces with median

Suggested minimum level of coherence for good results: 0.6

seismic.receiver_fn.rf_quality_filter.**compute_rf_quality_metrics**(*station_id*, *station_stream3c*, *similarity_eps*)

Top level function for adding quality metrics to trace metadata.

Parameters

- **station_id** (*str*) – Station ID
- **station_stream3c** (*list(rf.RFStream) with 3 components*) – 3-channel stream
- **similarity_eps** (*float*) – Distance threshold used for DBSCAN clustering

Returns Triplet of RF streams with Z, R or Q, and T components with populated quality metrics. Otherwise return None in case of failure.

seismic.receiver_fn.rf_quality_filter.**get_rf_stream_components**(*stream*)
Identify the RF component types and return them.

Parameters **stream** (*rf.RFStream*) – Stream containing mixed RF components.

Returns (RF component type, primary RF component (R or Q), transverse RF component (T), source component (Z or L))

Return type (str, rf.RFStream, rf.RFStream, rf.RFStream)

seismic.receiver_fn.rf_quality_filter.**rf_group_by_similarity**(*swipe*, *similarity_eps*)
Cluster waveforms by similarity

Parameters

- **swipe** (*numpy.array*) – Numpy array of RF rowwise
- **similarity_eps** (*float*) – Tolerance on similarity between traced to be considered in the same group.

Returns Index of the group for each trace. -1 if no group is found for a given trace.

Return type numpy.array

seismic.receiver_fn.rf_quality_filter.**rf_quality_metrics_queue**(*oqueue*, *station_id*, *station_stream3c*, *similarity_eps*, *drop_z=True*)
Produce RF quality metrics in a stream and queue the QC'd components for downstream processing.

Parameters

- **oqueue** (*queue or multiprocessing.Manager.Queue*) – Output queue where filtered streams are queued
- **station_id** (*str*) – Station ID
- **station_stream3c** (*list(rf.RFStream) with 3 components*) – 3-channel stream
- **similarity_eps** (*float*) – Distance threshold used for DBSCAN clustering

seismic.receiver_fn.rf_quality_filter.**spectral_entropy**(*stream*)
Compute the spectral entropy of a trace

Parameters **trace** (*rf.RFTrace*) – Single channel seismic trace

Returns Spectral entropy of the trace waveform

Return type float

1.7.19 seismic.receiver_fn.rf_stacking module

seismic.receiver_fn.rf_stacking.**compute_hk_stack**(*cha_data*, *V_p=None*, *h_range=None*, *k_range=None*, *root_order=1*, *include_t3=True*)
Compute H-k stacking array on a dataset of receiver functions.

Parameters

- **cha_data** (*iterable(rf.RFTrace)*) – List or iterable of RF traces to use for H-k stacking.

- **V_p** (*float, optional*) – P-wave velocity in crustal layer, defaults to None in which case it is inferred from trace metadata
- **h_range** (*numpy.array [1D], optional*) – Range of h values (Moho depth) values to cover, defaults to np.linspace(20.0, 70.0, 251)
- **k_range** (*numpy.array [1D], optional*) – Range of k values to cover, defaults to np.linspace(1.4, 2.0, 301)
- **root_order** (*int, optional*) – Exponent for nth root stacking as per K.J.Muirhead (1968), defaults to 1
- **include_t3** (*bool, optional*) – If True, include the t3 (PpSs+PsPs) multiple in the stacking, defaults to True

Returns k-grid values [2D], h-grid values [2D], H-k stack in series of 2D layers having one layer per multiple

Return type numpy.array [2D], numpy.array [2D], numpy.array [3D]

seismic.receiver_fn.rf_stacking.**compute_theoretical_phase_times**(*tr*, *H*, *k*, *V_p*, *include_t3=True*)

seismic.receiver_fn.rf_stacking.**compute_weighted_stack**(*hk_components*, *weighting=0.5, 0.5, 0.0*)

Given stack components from function *compute_hk_stack*, compute the overall weighted stack.

Parameters

- **hk_components** (*np.array*) – H-k stack layers returned from *compute_hk_stack*
- **weighting** (*tuple, optional*) – Weightings for (t1, t2, t3) layers respectively, defaults to (0.5, 0.5, 0.0)

Returns Weighted stack in H-k space

Return type numpy.array

seismic.receiver_fn.rf_stacking.**find_global_hk_maximum**(*k_grid*, *h_grid*, *hk_weighted_stack*)

Given the weighted stack computed from function *compute_weighted_stack* and the corresponding k-grid and h-grid, find the location in H-k space of the global maximum.

Parameters

- **k_grid** (*Two-dimensional numpy.array*) – Grid of k-values
- **h_grid** (*Two-dimensional numpy.array*) – Grid of H-values
- **hk_weighted_stack** (*Two-dimensional numpy.array*) – Grid of stacked RF sample values produced by function rf_stacking.computed_weighted_stack()

Returns Location of global maximum on the H-k grid of the maximum stack value.

Return type tuple(float, float)

seismic.receiver_fn.rf_stacking.**find_local_hk_maxima**(*k_grid*, *h_grid*, *hk_stack*, *min_rel_value=0.5*)

Given the weighted stack computed from function *compute_weighted_stack* and the corresponding k-grid and h-grid, find the locations in H-k space of all local maxima above a certain threshold.

Parameters

- **k_grid** (*Two-dimensional numpy.array*) – Grid of k-values
- **h_grid** (*Two-dimensional numpy.array*) – Grid of H-values

- **hk_stack** (*Two-dimensional numpy.array*) – Grid of stacked RF sample values produced by function rf_stacking.computed_weighted_stack()
- **min_rel_value** (*float, optional*) – Minimum value required relative to the largest value in the stack, defaults to 0.5

Returns List of tuples containing parameters of local maxima solutions, with values in the following order: (H, k, stack_value, row_index, col_index)

Return type list(tuple(float, float, float, int, int))

seismic.receiver_fn.rf_stacking.**infer_Vp_from_traces**(*cha_data*, *log=None*)

1.7.20 seismic.receiver_fn.rf_synthetic module

Helper functions for producing synthetic pseudo-Receiver function traces

seismic.receiver_fn.rf_synthetic.**convert_inclination_to_distance**(*inclinations*, *model='iasp91'*, *nominal_source_depth_km=10.0*)

Helper function to convert range of inclinations to teleseismic distance in degrees.

Parameters

- **inclinations** (*numpy.array(float)*) – Array of inclination angles in degrees
- **model** (*str, optional*) – Name of model to use for ray tracing, defaults to "iasp91"
- **nominal_source_depth_km** (*float, optional*) – Assumed depth of source events, defaults to 10.0

Returns Array of teleseismic distances in degrees corresponding to input inclination angles.

Return type numpy.array(float)

seismic.receiver_fn.rf_synthetic.**generate_synth_rf**(*arrival_times*, *arrival_amplitudes*, *fs_hz=100.0*, *window_sec=-10, 30*, *f_cutoff_hz=2.0*)

Simple generator of synthetic R component receiver function with pulses at given arrival times.

Parameters

- **arrival_times** (*iterable of float*) – Iterable of arrival times as numerical values in seconds
- **arrival_amplitudes** (*iterable of float*) – Iterable of arrival amplitudes
- **fs_hz** (*float, optional*) – Sampling rate (Hz) of output signal, defaults to 100.0
- **window_sec** (*tuple, optional*) – Time window over which to create signal (sec), defaults to (-10, 30)
- **f_cutoff_hz** (*float, optional*) – Cutoff frequency (Hz) for low-pass filtering to generate realistic result, defaults to 2.0

Returns Array of times and corresponding signal amplitudes

Return type numpy.array, numpy.array

seismic.receiver_fn.rf_synthetic.**synthesize_rf_dataset**(*H*, *V_p*, *V_s*, *inclinations*, *distances*, *ds*, *log=None*, *include_t3=False*, *amplitudes=None*, *baz=0.0*)

Synthesize RF R-component data set over range of inclinations and distances and get result as a rf.RFStream instance.

Parameters

- **H** (*float*) – Moho depth (km)
- **V_p** (*float*) – P body wave velocity in uppermost layer
- **V_s** (*float*) – S body wave velocity in uppermost layer
- **inclinations** (*numpy.array(float)*) – Array of inclinations for which to create RFs
- **distances** (*numpy.array(float)*) – Array of teleseismic distances corresponding to inclinations
- **ds** (*float*) – Final sampling rate (Hz) for the downsampled output signal
- **log** (*logger, optional*) – Logger to send output to, defaults to None
- **include_t3** (*bool, optional*) – If True, include the third expected multiple PpSs+PsPs
- **amplitudes** (*list(float), optional*) – Custom amplitudes to apply to the multiples
- **baz** (*float, optional*) – Back azimuth for metadata

Returns Stream containing synthetic RFs

Return type rf.RFStream

1.7.21 seismic.receiver_fn.rf_util module

Utility functions to help with RF processing and analysis.

seismic.receiver_fn.rf_util.**choose_rf_source_channel**(*rf_type*, *db_station*)

Choose source channel for RF analysis.

Parameters

- **rf_type** (*str*) – The RF rotation type, should be either 'ZRT' or 'LQT'
- **db_station** (*dict(str, list(rf.RFTrace))*) – Dict of traces for a given station keyed by channel code.

Returns Channel code of the primary RF source channel

Return type str

seismic.receiver_fn.rf_util.**compute_extra_rf_stats**(*stream*)

Compute extra statistics for each trace and add it to the RFTrace.stats structure.

Parameters **stream** (*rf.RFStream*) – RFStream to augment with additional metadata statistics.

seismic.receiver_fn.rf_util.**compute_rf_snr**(*rf_stream*)

Compute signal to noise (S/N) ratio of the RF itself about the onset pulse (key 'snr'). This SNR is a ratio of RMS amplitudes. Stores results in metadata of input stream traces.

In the LQT rotation case when rotation is working ideally, the onset pulse of the rotated transverse signals should be minimal, and a large pulse at t = 0 indicates lack of effective rotation of coordinate system, so for 'snr' we use a long time window after onset pulse, deliberately excluding the onset pulse, to maximize contribution to the SNR from the multiples after the onset pulse.

Parameters **rf_stream** (*rf.RFStream*) – R or Q component of Receiver Function

Returns SNR for each trace in the input stream

Return type numpy.array

seismic.receiver_fn.rf_util.**compute_vertical_snr**(*src_stream*)

Compute the SNR of the Z component (Z before deconvolution) including the onset pulse (key 'snr_prior'). Stores results in metadata of input stream traces. This SNR is a ratio of max envelopes.

Some authors compute this prior SNR on signal after rotation but before deconvolution, however that doesn't make sense for LQT rotation where the optimal rotation will result in the least energy in the L component. For simplicity we compute it on Z-component only which is a reasonable estimate for teleseismic events.

Parameters **src_stream** (*rf.RFStream or obspy.Stream*) – Seismic traces before RF deconvolution of raw stream.

seismic.receiver_fn.rf_util.**filter_crosscorr_coeff**(*rf_stream*, *time_window=-2, 25*, *threshold_cc=0.7*, *min_fraction=0.15*, *apply_moveout=False*)

For each trace in the stream, compute its correlation coefficient with the other traces. Return only traces matching cross correlation coefficient criteria based on C.Sippl (2016) [see http://dx.doi.org/10.1016/j.tecto.2016.03.031]

Parameters

- **rf_stream** (*rf.RFStream*) – Stream of RF traces to filter, should be **for a single component of a single station**
- **time_window** (*tuple, optional*) – Time window to filter by, defaults to (-2, 25)
- **threshold_cc** (*float, optional*) – Threshold cross-correlation coefficient, defaults to 0.70. Denoted Xi in Sippl, who used value 0.80.
- **min_fraction** (*float, optional*) – Minimum fraction of coefficients above threshold_cc, defaults to 0.15. Denoted tau in Sippl, who used value 0.15.
- **apply_moveout** (*bool*) – Whether to apply moveout correction to Ps phase prior to computing correlation coefficients.

Returns Filtered stream of RF traces

Return type rf.RFStream

seismic.receiver_fn.rf_util.**filter_station_streams**(*db_station*, *freq_band=None, None*)

Perform frequency filtering on all channels' traces for a given station. Returns a copy of db_station with streams containing filtered results.

seismic.receiver_fn.rf_util.**filter_station_to_mean_signal**(*db_station*, *min_correlation=1.0*)

Filter out streams which are not 'close enough' to the mean signal, based on simple correlation score. The term "correlation" here really just means a similarity dot product (projection of individual trace onto the mean).

seismic.receiver_fn.rf_util.**find_rf_group_ids**(*stream*)

For the given stream, which is expected to have an rf_group attribute in its traces' metadata, determine the unique set of group ids that the traces contain.

Parameters **stream** (*obspy.core.trace.Trace*) – Stream containing traces with rf_group ids associated with them.

Returns Set of rf_group ids found in the traces

Return type set(int)

seismic.receiver_fn.rf_util.**label_rf_quality_simple_amplitude**(*rf_type*, *traces*, *snr_cutoff=2.0*, *rms_amp_cutoff=0.2*, *max_amp_cutoff=1.0*)

Add RF quality label for a collection of RFs based on simple amplitude criteria computed by quality filter script. Adds quality label in-place.

Parameters

- **rf_type** (*str*) – The RF rotation type, should be either 'ZRT' or 'LQT'
- **traces** (*Iterable collection of rf.RFTrace*) – Iterable collection of rf.RFTrace
- **snr_cutoff** (*float, optional*) – Minimum signal SNR, defaults to 2.0
- **rms_amp_cutoff** (*float, optional*) – Maximum accepted RMS amplitude of signal, defaults to 0.2
- **max_amp_cutoff** (*float, optional*) – Maximum accepted amplitude of signal, defaults to 1.0

seismic.receiver_fn.rf_util.**phase_weights**(*stream*)

Phase weighting takes all the traces in a stream and computes a weighting for each sample in the stream between 0 and 1. The weighting represents how consistent is the phase angle of the signal at the same point in the time series across all streams.

If phase weighting to accentuate higher multiples than Ps, then moveout should be applied first before calling this function.

See https://doi.org/10.1111/j.1365-246X.1997.tb05664.x

Note: this function should not be applied to streams with mixed components.

Parameters **stream** (*Iterable container of obspy.Traces*) – Stream containing one or more traces from which phase coherence weightings will be generated.

Returns Array of normalized weighting factors with same length as traces in stream.

Return type numpy.array

seismic.receiver_fn.rf_util.**read_h5_rf**(*src_file*, *network=None*, *station=None*, *loc=''*, *root='/waveforms'*)

Helper function to load data from hdf5 file generated by rf library or script *rf_quality_filter.py*. For faster loading time, a particular network and station may be specified.

Parameters

- **src_file** (*str or Path*) – File from which to load data
- **network** (*str, optional*) – Specific network to load, defaults to None
- **station** (*str, optional*) – Specific station to load, defaults to None
- **root** (*str, optional*) – Root path in hdf5 file where to start looking for data, defaults to '/waveforms'

Returns All the loaded data in a rf.RFStream container.

Return type rf.RFStream

seismic.receiver_fn.rf_util.**rf_to_dict**(*rf_data*)

Convert RF data loaded from function read_h5_rf() into a dict format for easier addressing of selected station and channel RF traces.

Parameters **rf_data** (*rf.RFStream*) – RFStream data

Returns Nested dicts to find traces by station then channel code, with attached metadata.

Return type *seismic.receiver_fn.rf_network_dict.NetworkRFDict*

seismic.receiver_fn.rf_util.**signed_nth_power**(*arr*, *order*)

As per DOI https://doi.org/10.1038/217533a0. Muirhead, K.J. "Eliminating False Alarms when detecting Seismic Events Automatically"

Parameters

- **arr** (*numpy.array*) – Compute n-th power of input array, preserving sign of original data.
- **order** (*float or int*) – Order of the power to compute

Returns Input array raised to nth power.

Return type numpy.array

seismic.receiver_fn.rf_util.**signed_nth_root**(*arr*, *order*)

As per DOI https://doi.org/10.1038/217533a0. Muirhead, K.J. "Eliminating False Alarms when detecting Seismic Events Automatically"

Parameters

- **arr** (*numpy.array*) – Compute n-th root of input array, preserving sign of original data.
- **order** (*float or int*) – Order of the root to compute

Returns Input array raised to 1/nth power.

Return type numpy.array

1.7.22 Module contents

1.8 seismic.synthetics package

1.8.1 Subpackages

seismic.synthetics.backends package

Submodules

seismic.synthetics.backends.backend_syngine module

Backend for making synthetic seismograms using Syngine.

class seismic.synthetics.backends.backend_syngine.**SynthesizerSyngine**(*station_latlon*, *earth_model='iasp91'*)

Bases: *seismic.synthetics.backends.synthesizer_base.Synthesizer*

Class to synthesize seismograms using online Syngine service.

To write resultant stream to HDF5 format, add 'ignore' option: *synth_stream.write('test_synth.h5', 'h5', ignore=('mseed',))*

synthesize(*src_latlon*, *fs*, *time_window*)
See documentation for *synthesize()*

seismic.synthetics.backends.backend_syngine.**synthesizer**()

seismic.synthetics.backends.backend_tws module

Backend for making synthetic seismograms using Telewavesim.

class seismic.synthetics.backends.backend_tws.**SynthesizerMatrixPropagator**(*station_latlon*, *layer-props*)
Bases: *seismic.synthetics.backends.synthesizer_base.Synthesizer*

Class to synthesize seismograms from a 1D model description using Kennett's matrix propagator method.

property kappa

synthesize(*src_latlon*, *fs*, *time_window*)
See documentation for *synthesize()*

seismic.synthetics.backends.backend_tws.**synthesizer**()

seismic.synthetics.backends.synthesizer_base module

Base class for seismogram synthesis class

class seismic.synthetics.backends.synthesizer_base.**Synthesizer**(*station_latlon*)
Bases: object

Base class for seismogram synthesizers.

compute_event_stats(*src_lat*, *src_lon*, *eventid_base*, *src_depth_m=0*, *earth_model='iasp91'*, *phase='P'*, *origin_time=None*)
Compute trace stats fields for a source single event.

Parameters

- **src_lat** – Source latitude
- **src_lon** – Source longitude
- **eventid_base** – Base string for event id
- **src_depth_m** – Source depth in metres
- **earth_model** – String name of earth model to use for ray tracing
- **phase** – Which phase is being modelled
- **origin_time** – Timestamp of the source event

Returns stats dict

property station_latlon

abstract synthesize(*src_latlon*, *fs*, *time_window*)
Function signature for function to compute synthetic dataset of obspy streams.

Parameters

- **src_latlon** – Iterable of source (lat, lon) locations
- **fs** – Sampling rate in Hz
- **time_window** – Pair of time values relative to onset

Returns obspy.Stream containing ZNE velocity seismogram

Module contents

1.8.2 Submodules

1.8.3 seismic.synthetics.synth module

Entry point for making synthetic seismograms.

`seismic.synthetics.synth.`**`example_usage`**`()`

`seismic.synthetics.synth.`**`synthesize_dataset`**`(`*method*, *output_file*, *net*, *sta*, *src_latlon*, *fs*, *time_window*, ***kwargs*`)`

User function for creating a synthetic seismogram dataset of obspy streams in HDF5 format. Datasets generated can be loaded into class NetworkEventDataset.

Parameters

- **method** – 'propmatrix' or 'syngine'
- **output_file** – Destination file in which to write resultant streams in HDF5 format.
- **net** – Network code of receiver
- **sta** – Station code of receiver
- **src_latlon** – Iterable of source event coordinates
- **fs** – Sampling rate
- **time_window** – Time window about onset. First value should be < 0, second should be > 0

Returns Whether the dataset was successfully created.

1.8.4 Module contents

1.9 seismic.traveltime package

1.9.1 Submodules

1.9.2 seismic.traveltime.cluster_grid module

Description: Define a 3D grid discretization of the Earth

CreationDate: 2/09/2018

Developer:

fei.zhang@ga.gov.au

class seismic.traveltime.cluster_grid.**Grid2**(*ndis=2*, *param_file=None*)
Bases: object

A non-uniform Grid Model of Earth for eventsource–>station rays clustering and sorting.

Reference to the inversion Fortran code model parameter file param1x1 in this directory:

- Global: 5x5 degree and a list of 1+16 depths
- Region: 1x1 degree and a list of 1+29 depths

find_block_number(*lat*, *lon*, *z*)
find the index-number of an events/station in a non-uniform grid each spatial point (lat, lon, z) is mapped to a uniq block_number.

Parameters

- **lat** – latitude (-90,90)
- **lon** – longitude (0,360)
- **z** – depth meter

Returns a tuple of 4 block number AND and the block's centre coordinate(int_block_n, xc,yc, zcm)

generate_grid3D()
Loop over all the reference depths, both regional and global depth list.

Returns pandas dataframe

generate_latlong_grid(*depthmeters=0.0*)
create a csv file containing: (block_number, lat,long, depthm=0, xc,yc,zc)

Returns Pandas data frame

get_depth_index(*z*, *dep_meters*)
given a point with depth z meters, and a np-array of refined depth in meters, find the index which z fall into.

Parameters

- **z** – an event depth in meters
- **dep_meters** – an array of numbers corresponding to a refined depth discretization.

Returns depth index and the cell block centre depth in metres.

is_point_in_region(*lat*, *lon*)
test if the event or station point is in the region box?

Parameters

- **lat** – latitude
- **lon** – longitude

Returns boolean T or F

show_properties()
print the properties of the grid definition

class seismic.traveltime.cluster_grid.**UniformGrid**(*nx=360*, *ny=180*, *dz=10000*)
Bases: object

This is the original (simple) uniform grid model for Earth.

It is replaced by the more generic non-uniform model defined above as class Grid2

find_block_number(*lat*, *lon*, *z*)
find the 3D-block number in this grid

Parameters

- **lat** – lattitude
- **lon** – longitude
- **z** – elevation

Returns int block number

1.9.3 seismic.traveltime.events_stations_rays_visualization2 module

seismic.traveltime.events_stations_rays_visualization2.**plot1**(*mypdf*)

Parameters **mypdf** – which dataframe to view in the following??

Returns

seismic.traveltime.events_stations_rays_visualization2.**plot_geopandas**(*mypdf*)
Geopandas and Maps :param mypdf: :return:

seismic.traveltime.events_stations_rays_visualization2.**plot_gmt**(*mypdf*)
GMT-python plotting map figure saved into PNG file. But not displaying on screen

Require to conda install gmt-python module in anaconda python gmt version-6 will be installed. :param mypdf: :return:

seismic.traveltime.events_stations_rays_visualization2.**plot_rays**(*mypdf*)

Parameters **mypdf** –

Returns

seismic.traveltime.events_stations_rays_visualization2.**read_csv2pdf**(*csvfile*)
Read in a csv file into a pandas dataframe. Make sure the column names match the csv files. delimiter/separator is whitespace or comma

seismic.traveltime.events_stations_rays_visualization2.**sort_plus**(*inputcsv=None*)

1.9.4 seismic.traveltime.gather_events module

Parse multiple events xml files to gather all seismic events-arrivals (Refactored From the original seismic.cluster.cluster.py)

output CSV files containing the following columns:

```
['source_block', 'station_block', 'residual', 'event_number',
 SOURCE_LONGITUDE, SOURCE_LATITUDE, 'source_depth', STATION_LONGITUDE, STATION_
↪LATITUDE,
 'observed_tt', 'calculated_locations2degrees', xml_distance, STATION_CODE, 'SNR', 'P_
↪or_S']
```

How to Run:

```
export ELLIPCORR=/g/data1a/ha3/fxz547/Githubz/passive-seismic/ellip-corr/
cd  passive-seismic/tempworks
# python ../seismic/traveltime/gather_events.py  -v DEBUG gather /g/data/ha3/fxz547/
↪Githubz/passive-seismic/testdata/
```

(continues on next page)

(continued from previous page)

```
# python ../seismic/traveltime/gather_events.py  -v DEBUG gather /g/data/ha3/fxz547/
↪Githubz/passive-seismic/some_events_xml/
#  OR
#<fzhang@ubuntu16qos>/Softlab/Githubz/passive-seismic/tempworks (master)
# $ python2 ../seismic/traveltime/gather_events.py  -v DEBUG gather ./events_xmls_test
```

class seismic.traveltime.gather_events.**ArrivalWriter**(*rank*, *wave_type*, *output_file*)

Bases: object

Convenience class for writing arrival data

close()

write(*cluster_info*)

seismic.traveltime.gather_events.**getSNR**(*arrival*)

From the arrival get the SNR value. This algorithm depend on how the snr value is coded in the xml file :param arrival: :return: a float SNR value

seismic.traveltime.gather_events.**get_paths_from_csv**(*csvfile*)

Parse a text/csv file to extract a list of paths, where events xml files are stored, to be gathered.

Parameters **csvfile** – csv file

Returns list_of_paths

seismic.traveltime.gather_events.**process_event**(*event*, *stations*, *grid*, *wave_type*, *counter*)

Parameters

- **event** – obspy.core.event.Event class instance
- **stations** – dict stations dict
- **grid** – can be None; Grid class instance
- **wave_type** – str Wave type to generate inversion inputs. See *gather* function.
- **counter** – int event counter in this process

seismic.traveltime.gather_events.**process_many_events**(*event_xmls*, *grid*, *stations*, *wave_type*, *output_file*, *seed=1*)

seismic.traveltime.gather_events.**recursive_glob**(*dirname*, *ext='*.xml'*)

Under the dirname recursively find all files with extension ext. Return a list of the full-path to the files of interest.

See: https://stackoverflow.com/a/2186565/3321542

Parameters

- **dirname** – a single dir OR a list of dirs.
- **ext** – eg, ".xml"

Returns a list of path2files

1.9.5 seismic.traveltime.mpiops module

seismic.traveltime.mpiops.**array_split**(*arr*, *process=None*)
Convenience function for splitting array elements across MPI processes

Parameters

- **arr** (*ndarray*) – Numpy array
- **process** (*int*) – Process for which array members are required. If None, MPI.comm.rank is used instead. (optional)

:return List corresponding to array members in a process. :rtype: list

seismic.traveltime.mpiops.**run_once**(*f*, **args*, ***kwargs*)
Run a function on one node and then broadcast result to all.

Parameters

- **f** (*str*) – The function to be evaluated. Can take arbitrary arguments and return anything or nothing
- **args** (*str*) – Other positional arguments to pass on to f (optional)
- **kwargs** (*str*) – Other named arguments to pass on to f (optional)

Returns The value returned by f.

Return type unknown

1.9.6 seismic.traveltime.parametrisation module

Class for parsing Alexei's tomographic parametrisation file as an object.

Created on Wed Apr 03 13:52:28 2019

@author: Marcus W. Haynes

class seismic.traveltime.parametrisation.**grid**(*in_file='param'*)
Bases: object

extract_1D(*lon*, *lat*, *values*, *ref=None*, *method='linear'*)
Extracts arbitrary 1D vertical seismic velocity profiles from an inversion model.

Input::

- lon - float or list of desired longitude locations.
- lat - float or list of desired latitude locations.
- values - array of inversion values corresponding to the grid class.
- ref [optional] - the reference model to convert velocity perturbations to absolute values, if desired. Needs to be a 2D array with depth in first column and velocity in second column.
- method [optional] - method used to interpolate values to location

save()
Write the parametrisation file to disk

1.9.7 seismic.traveltime.parse_param module

Created on Wed Oct 31 14:15:44 2018

@author: u81234

class seismic.traveltime.parse_param.**grid3**
Bases: object

parse_parametrisation(*param_file='./example_param.txt'*)
Reads in the local and global grid parameters from an external parametrisation file (by default we load './param')

1.9.8 seismic.traveltime.plotviews module

class seismic.traveltime.plotviews.**Region**(*upperlat*, *bottomlat*, *leftlon*, *rightlon*)
Bases: tuple

property bottomlat
Alias for field number 1

property leftlon
Alias for field number 2

property rightlon
Alias for field number 3

property upperlat
Alias for field number 0

1.9.9 seismic.traveltime.pslog module

class seismic.traveltime.pslog.**ElapsedFormatter**
Bases: object

format(*record*)

seismic.traveltime.pslog.**configure**(*verbosity*)

seismic.traveltime.pslog.**warn_with_traceback**(*message*, *category*, *filename*, *lineno*, *line=None*)
copied from: http://stackoverflow.com/questions/22373927/get-traceback-of-warnings

1.9.10 seismic.traveltime.sort_rays module

Description: Read in event-arrival seismic rays and sort them according to a discretization model of Earth.

The input file header assumed to be:

col_names=['source_block', 'station_block', 'residual', 'event_number','source_longitude','source_latitude','source_depth', 'station_longitude','station_latitude', 'observed_tt', 'locations2degrees', 'station_code','SNR', 'P_or_S']

The output CSV file will be feed into an inversion program which will derive travel-time tomography images.

Developer: fei.zhang@ga.gov.au

seismic.traveltime.sort_rays.**apply_filters**(*csv_data*, *phase*)
Apply filters to the rows (rays) according to phase P or S. Remove un-reliable data. :param csv_data: input pandas dataframe :param phase: P or S :return: pandas dataframe

seismic.traveltime.sort_rays.**compute_ellipticity_corr**(*arrival_phase*, *ev_latitude*, *ev_longitude*, *ev_depth_km*, *sta_latitude*, *sta_longitude*, *degrees_to_source*)

Utility function to compute ellipticity correction.

Parameters

- **arrival_phase** – P or S
- **ev_latitude** – event lat
- **ev_longitude** – event long
- **ev_depth_km** – event depth in km
- **sta_latitude** – station lat
- **sta_longitude** – station long
- **degrees_to_source** – degree to source

Returns ellipticity correction float value

seismic.traveltime.sort_rays.**filter_S_by_P**(*inpdf*, *pevents_csv*)

Filter the S picks by P events :param inpdf: input pdf :param pevents_csv: Pevents csv file with 3 columns: ['#eventID', 'net','sta'] :return: a new csv pdf after the filtering

seismic.traveltime.sort_rays.**get_columns_dict**(*jsonfile*)

seismic.traveltime.sort_rays.**is_manual_pick**(*x*)

seismic.traveltime.sort_rays.**is_ray_in**(*ray_id*, *P_RAYS_ID_LIST*)

check if an ray_id is in the P_RAYS_ID_LIST :param ray_id: "net_sta_evtid" :param P_RAYS_ID_LIST: a sequence of Ray_id of P waves. :return: 1 if in; 0 if not in

seismic.traveltime.sort_rays.**sort**(*input_file*, *sorted_file*, *residual_cutoff*)

Sort and filter the arrivals based on the source and station block number. There are two stages of filtering:

- Filter based on the arrival time residual value: defult value for the cutoff is 5 for P-Wave, 10 for S-Wave
- Filter based on median of observed travel time.

If there are multiple source and station block combinations, we keep the row corresponding to the median observed travel time (observed_tt).

Parameters

- **input_file** – output file from the gather stage (eg, outfile_P.csv)
- **sorted_file** – str, optional optional sorted output file path. Default: sorted.csv.
- **residual_cutoff** – float residual seconds, arrivals are rejected if the residual is larger than the cutoff

Returns A pandas df

seismic.traveltime.sort_rays.**sort2**(*input_file*, *sorted_file*, *residual_cutoff*)

Sort and filter the arrivals based on the source and station block number. If there are multiple source and station block combinations, the row corresponding to the highest SNR value is kept. (This function is for experiment only, not used.)

Parameters

- **input_file** – output file from the gather stage (eg, outfile_P.csv)
- **sorted_file** – str, optionaloptional sorted output file path. Default: sorted.csv.

- **residual_cutoff** – float arrivals are rejected if the residual is larger than the cutoff.

Returns pandas_df

seismic.traveltime.sort_rays.**sort_csv_in_grid**(*inputcsv*, *outputcsv*, *phase*, *mygrid*, *column_name_map*)

Read in a csv file, re-grid each row according to a given Grid model. Write into output csv file with re-calculated block_numbers re-named columns

Parameters

- **inputcsv** – path to an input csv file
- **outputcsv** – path to output file
- **phase** – P or S
- **mygrid** – instance of Earth Grid model
- **column_name_map** – column map dictionary as in csv_columns.json file

Returns outfile

seismic.traveltime.sort_rays.**translate_csv**(*in_csvfile*, *out_csvfile*)

Read in a csv file, re-grid each row according to a new Grid model. Write into another csv file with re-calculated block_numbers and six new columns of Grid cell centers.

Parameters

- **in_csvfile** – path to an input csv file
- **out_csvfile** – path to an output csv file

Returns out_csvfile

1.9.11 seismic.traveltime.zone_rays module

class seismic.traveltime.zone_rays.**Region**(*upperlat*, *bottomlat*, *leftlon*, *rightlon*)

Bases: tuple

property bottomlat
Alias for field number 1

property leftlon
Alias for field number 2

property rightlon
Alias for field number 3

property upperlat
Alias for field number 0

1.9.12 Module contents

1.10 seismic.xcorqc package

1.10.1 Submodules

1.10.2 seismic.xcorqc.analytic_plot_utils module

Utility functions supporting plotting for cross-correlation visualizations.

seismic.xcorqc.analytic_plot_utils.**distance**(*origin*, *destination*)

Compute the distance in km between origin coordinates and destination coordinates. The coordinates are (latitude, longitude) couplets in units of degrees.

Parameters

- **origin** (*tuple(float, float)*) – Coordinates of origin point
- **destination** (*tuple(float, float)*) – Coordinates of destination point

Returns Epicentral distance between origin and destination in kilometres

Return type float

seismic.xcorqc.analytic_plot_utils.**drawBBox**(*min_lon*, *min_lat*, *max_lon*, *max_lat*, *base_map*, ***kwargs*)

Draw bounding box on a basemap

Parameters

- **min_lon** (*float*) – Minimum longitude
- **min_lat** (*float*) – Minimum latitude
- **max_lon** (*float*) – Maximum longitude
- **max_lat** (*float*) – Maximum latitude
- **base_map** (*mpl_toolkits.basemap.Basemap*) – Basemap on which to draw the bounding box

seismic.xcorqc.analytic_plot_utils.**timestamps_to_plottable_datetimes**(*time_series*)

Convert a series of float (or equivalent) timestamp values to matplotlib plottable datetimes.

Parameters **time_series** (*iterable container*) – Series of timestamps

Returns Equivalent series of plottable timestamps

Return type numpy.array('datetime64[ms]') with millisecond resolution

1.10.3 seismic.xcorqc.client_data module

seismic.xcorqc.client_data.**main**()

1.10.4 seismic.xcorqc.correlator module

Description: Generates cross-correlations for data from staion-pairs in parallel

References:

CreationDate: 11/07/18

Developer: rakib.hassan@ga.gov.au

Revision History: LastUpdate: 11/07/18 RH LastUpdate: dd/mm/yyyy Who Optional description

class seismic.xcorqc.correlator.**Dataset**(*asdf_file_name*, *netsta_list='*'*)

Bases: object

get_closest_stations(*netsta*, *other_dataset*, *nn=1*)

get_unique_station_pairs(*other_dataset*, *nn=1*)

seismic.xcorqc.correlator.**process**(*data_source1*, *data_source2*, *output_path*, *interval_seconds*, *window_seconds*, *window_overlap*, *window_buffer_length*, *resample_rate=None*, *taper_length=0.05*, *nearest_neighbours=1*, *fmin=None*, *fmax=None*, *netsta_list1='*'*, *netsta_list2='*'*, *pairs_to_compute=None*, *start_time='1970-01-01T00:00:00'*, *end_time='2100-01-01T00:00:00'*, *instrument_response_inventory=None*, *instrument_response_output='vel'*, *water_level=50*, *clip_to_2std=False*, *whitening=False*, *whitening_window_frequency=0*, *one_bit_normalize=False*, *read_buffer_size=10*, *ds1_zchan=None*, *ds1_nchan=None*, *ds1_echan=None*, *ds2_zchan=None*, *ds2_nchan=None*, *ds2_echan=None*, *corr_chan=None*, *envelope_normalize=False*, *ensemble_stack=False*, *restart=False*, *dry_run=False*, *no_tracking_tag=False*)

DATA_SOURCE1: Text file containing paths to ASDF files

DATA_SOURCE2: Text file containing paths to ASDF files

OUTPUT_PATH: Output folder

INTERVAL_SECONDS: Length of time window (s) over which to compute cross-correlations; e.g. 86400 for 1 day

WINDOW_SECONDS: Length of stacking window (s); e.g 3600 for an hour. INTERVAL_SECONDS must be a multiple o
WINDOW_SECONDS; no stacking is performed if they are of the same size.

1.10.5 seismic.xcorqc.fft module

seismic.xcorqc.fft.**ndflip**(*a*)

Inverts an n-dimensional array along each of its axes

1.10.6 seismic.xcorqc.generate_dispersion_curves module

Description: Runs Rhys Hawkins' code in parallel to generate dispersion curves based on cross-correlations of station-pairs. Note that this script call shell scripts that are expected to be in the current working directory.

todo: remove dependence on shell scripts.

References:

CreationDate: 10/01/20

Developer: rakib.hassan@ga.gov.au

Revision History: LastUpdate: 10/01/20 RH LastUpdate: dd/mm/yyyy Who Optional description

`seismic.xcorqc.generate_dispersion_curves.`**`kill`**(*proc_pid*)

`seismic.xcorqc.generate_dispersion_curves.`**`runprocess`**(*cmd*, *get_results=False*)

`seismic.xcorqc.generate_dispersion_curves.`**`split_list`**(*lst*, *npartitions*)

1.10.7 seismic.xcorqc.generate_test_data module

`seismic.xcorqc.generate_test_data.`**`generateStationTestData`**(*sta*)

1.10.8 seismic.xcorqc.utils module

class `seismic.xcorqc.utils.`**`ProgressTracker`**(*output_folder*, *restart_mode=False*)
Bases: `object`

`increment`()

`seismic.xcorqc.utils.`**`drop_bogus_traces`**(*st*, *sampling_rate_cutoff=1*)
Removes spurious traces with suspect sampling rates. :param st: Obspy Stream :param sampling_rate_cutoff: sampling rate threshold :return: Input stream is updated inplace

`seismic.xcorqc.utils.`**`getStationInventory`**(*master_inventory*, *inventory_cache*, *netsta*)

`seismic.xcorqc.utils.`**`get_stream`**(*fds*, *net*, *sta*, *cha*, *start_time*, *end_time*, *baz=None*, *trace_count_threshold=200*, *logger=None*, *verbose=1*)

`seismic.xcorqc.utils.`**`rtp2xyz`**(*r*, *theta*, *phi*)

`seismic.xcorqc.utils.`**`split_list`**(*lst*, *npartitions*)

`seismic.xcorqc.utils.`**`xyz2rtp`**(*x*, *y*, *z*)

1.10.9 seismic.xcorqc.validate_xcorr_setup module

`seismic.xcorqc.validate_xcorr_setup.`**`test_setup`**()

1.10.10 seismic.xcorqc.xcorqc module

Description: Cross-correlation functionality

References:

CreationDate: 29/06/17

Developer: laurence.davies@ga.gov.au

Revision History: LastUpdate: 29/06/17 LD First commit of xcor code. LastUpdate: 13/07/17 LD Fixed xcor filtering issue when traces have different sample rates. LastUpdate: 11/08/17 RH Implement ASDF-based cross-correlation workflow LastUpdate: 11/07/18 RH Implemented parallel cross-correlator LastUpdate: 19/07/18 RH Implemented cross-correlation approaches described in Habel et al. 2018

LastUpdate: dd/mm/yyyy Who Optional description

seismic.xcorqc.xcorqc.**IntervalStackXCorr**(*refds*, *tempds*, *start_time*, *end_time*, *ref_net_sta*, *temp_net_sta*, *ref_sta_inv*, *temp_sta_inv*, *instrument_response_output*, *water_level*, *ref_cha*, *temp_cha*, *baz_ref_net_sta*, *baz_temp_net_sta*, *resample_rate=None*, *taper_length=0.05*, *buffer_seconds=864000*, *interval_seconds=86400*, *window_seconds=3600*, *window_overlap=0.1*, *window_buffer_length=0*, *flo=None*, *fhi=None*, *clip_to_2std=False*, *whitening=False*, *whitening_window_frequency=0*, *one_bit_normalize=False*, *envelope_normalize=False*, *ensemble_stack=False*, *outputPath='/tmp'*, *verbose=1*, *tracking_tag=''*)

This function rolls through two ASDF data sets, over a given time-range and cross-correlates waveforms from all possible station-pairs from the two data sets. To allow efficient, random data access asdf data sources, an instance of a SeisDB object, instantiated from the corresponding Json database is passed in (tempds_db) – although this parameter is not mandatory, data-access from large ASDF files will be slow without it.

Station-ids to be processed from the two data-sources can be specified as lists of strings, while wildcards can be used to process all stations. Data is fetched from the sources in chunks to limit memory usage and data-windows with gaps are discarded.

Cross-correlation results are written out for each station-pair, in the specified folder, as NETCDF4 files. Panoply (https://www.giss.nasa.gov/tools/panoply/), already installed on the NCI VDIs can be used to interrogate these results.

Parameters

- **refds** (FederatedASDFDataSet) – FederatedASDFDataSet containing reference-station data
- **tempds** (FederatedASDFDataSet) – FederatedASDFDataSet containing temporary-stations data
- **ref_net_sta** (*str*) – Network.Station for the reference Dataset.
- **temp_net_sta** (*str*) – Network.Station for the temporary Dataset.
- **ref_sta_inv** (*Inventory*) – Inventory containing instrument response for station
- **temp_sta_inv** (*Inventory*) – Inventory containing instrument response for station
- **instrument_response_output** (*str*) – Output of instrument response correction; can be either 'vel' or 'disp'
- **water_level** (*float*) – Water-level used during instrument response correction

- **ref_cha** (*str*) – Channel name for the reference Dataset
- **temp_cha** (*str*) – Channel name for the temporary Dataset
- **baz_ref_net_sta** (*float*) – Back-azimuth of ref station from temp station in degrees
- **baz_temp_net_sta** (*float*) – Back-azimuth of temp station from ref station in degrees
- **resample_rate** (*float*) – Resampling rate (Hz). Applies to both data-sets
- **taper_length** (*float*) – Taper length as a fraction of window length
- **buffer_seconds** (*int*) – The amount of data to be fetched per call from the ASDF-DataSets, because we may not be able to fetch all the data (from start_time to end_time) at once. The default is set to 10 days and should be a multiple of interval_seconds.
- **interval_seconds** (*int*) – The interval in seconds, over which cross-correlation windows are stacked. Default is 1 day.
- **window_seconds** (*int*) – Length of cross-correlation window in seconds. Default is 1 hr.
- **window_overlap** (*float*) – Window overlap fraction. Default is 0.1.
- **window_buffer_length** (*float*) – Buffer length as a fraction of 'window-seconds' around actual data windows of interest. This helps exclude effects of tapering and other edge artefacts from data windows before cross-correlation. Default is 0
- **flo** (*float*) – Lower frequency for Butterworth bandpass filter
- **fhi** (*float*) – Upper frequency for Butterworth bandpass filter
- **clip_to_2std** (*bool*) – Clip data in each window to +/- 2 standard deviations
- **whitening** (*bool*) – Apply spectral whitening
- **whitening_window_frequency** (*float*) – Window frequency (Hz) used to determine length of averaging window for smoothing spectral amplitude
- **one_bit_normalize** (*bool*) – Apply one-bit normalization to data in each window
- **envelope_normalize** (*bool*) – Envelope via Hilbert transforms and normalize
- **ensemble_stack** (*bool*) – Outputs a single CC function stacked over all data for a given station-pair
- **verbose** (*int*) – Verbosity of printouts. Default is 1; maximum is 3.
- **tracking_tag** (*str*) – File tag to be added to output file names so runtime settings can be tracked
- **outputPath** (*str*) – Folder to write results to

Param start_time: Start-time (UTCDateTime format) for data to be used in cross-correlation

Param end_time: End-time (UTCDateTime format) for data to be used in cross-correlation

Returns 1: 1d np.array with time samples spanning [-window_samples+dt:window_samples-dt] 2: A dictionary of 2d np.arrays containing cross-correlation results for each station-pair. Rows in each 2d array represent number of interval_seconds processed and columns represent stacked samples of length window_seconds. 3: A dictionary of 1d np.arrays containing number of windows processed, within each interval_seconds period, for each station-pair. These Window-counts could be helpful in assessing robustness of results.

seismic.xcorqc.xcorqc.**setup_logger**(*name*, *log_file*, *level=20*)

Function to setup a logger; adapted from stackoverflow

seismic.xcorqc.xcorqc.**taper**(*tr*, *taperlen*)

seismic.xcorqc.xcorqc.**whiten**(*a*, *sampling_rate*, *window_freq=0*)

Applies spectral whitening to trace samples. When window_freq=0, all frequency bins are normalized by their amplitudes, i.e. all frequency bins end up with an amplitude of 1. When window_freq is nonzero, a smoothed amplitude spectrum (smoothing window length is as computed below) is used to normalize the frequency bins.

Parameters

- **a** – trace samples
- **sampling_rate** – sampling rate
- **window_freq** – smoothing window length (Hz)

Returns spectrally whitened samples

seismic.xcorqc.xcorqc.**xcorr2**(*tr1*, *tr2*, *sta1_inv=None*, *sta2_inv=None*, *instrument_response_output='vel'*, *water_level=50.0*, *window_seconds=3600*, *window_overlap=0.1*, *window_buffer_length=0*, *interval_seconds=86400*, *taper_length=0.05*, *resample_rate=None*, *flo=None*, *fhi=None*, *clip_to_2std=False*, *whitening=False*, *whitening_window_frequency=0*, *one_bit_normalize=False*, *envelope_normalize=False*, *verbose=1*, *logger=None*)

seismic.xcorqc.xcorqc.**zeropad**(*tr*, *padlen*)

seismic.xcorqc.xcorqc.**zeropad_ba**(*tr*, *padlen*)

1.10.11 seismic.xcorqc.xcorr_station_clock_analysis module

Functions for computing estimated GPS clock corrections based on station pair cross-correlation and plotting in standard layout.

class seismic.xcorqc.xcorr_station_clock_analysis.**XcorrClockAnalyzer**(*src_file*, *time_window*, *snr_threshold*, *pcf_cutoff_threshold*)

Bases: object

Helper class for bundling of preprocessed cross-correlation data before plotting or subsequent processing.

do_clustering(*coeffs*)

Do DBSCAN clustering on the corrections.

Parameters **coeffs** (*tuple(float, float, float)*) – Triplet of distance coefficients, corresponding to the sensitivity of the clustering to point separation along 1) x-axis (time), 2) y-axis (correction) and 3) slope (drift rate)

Returns Results of sklearn.cluster.dbscan (refer to third party documentation)

do_spline_regression(*group_ids*, *regression_degree*)

Do univariate spline regression on each cluster of points.

Parameters

- **group_ids** – Cluster IDS generated from do_clustering()
- **regression_degree** – Desired degree of curve fit for each cluster, one for each non-negative cluster ID

Returns dict of regressors that can be applied to arbitrary time values

do_spline_resampling(*group_ids*, *regressors*, *sampling_period_seconds*)
Using pre-computed regressors, resample every cluster at a prescribed frequency

Parameters

- **group_ids** –
- **regressors** –
- **sampling_period_seconds** –

Returns

get_corrections_time_series()

plot_clusters(*ax*, *ids*, *coeffs*, *stn_code=''*)
Plot the distinct clusters color coded by cluster ID, with underlying corrections shown in gray.

Parameters

- **ax** –
- **ids** –
- **coeffs** –

Returns

plot_regressors(*ax*, *ids*, *regressors*, *stn_code=''*)
Plot regressor functions on top of original data

Parameters

- **ax** –
- **ids** –
- **regressors** –
- **stn_code** –

Returns

plot_resampled_clusters(*ax*, *ids*, *resampled_corrections*, *stn_code=''*)
Plot resampled regressor functions per cluster on top of original data

Parameters

- **ax** –
- **ids** –
- **regressors** –
- **stn_code** –

Returns

seismic.xcorqc.xcorr_station_clock_analysis.**batch_process_folder**(*folder_name*, *dataset*, *time_window*, *snr_threshold*, *pearson_cutoff_factor=0.5*, *save_plots=True*)
Process all the .nc files in a given folder into graphical visualizations.

Parameters

- **folder_name** (*str*) – Path to process containing .nc files
- **dataset** (*FederatedASDFDataset*) – Dataset to be used to ascertain the distance between stations.
- **time_window** (*float*) – Lag time window to plot (plus or minus this value in seconds)
- **snr_threshold** (*float*) – Minimum signal to noise ratio for samples to be included into the clock lag estimate
- **save_plots** – Whether to save plots to file, defaults to True
- **save_plots** – bool, optional

seismic.xcorqc.xcorr_station_clock_analysis.**batch_process_xcorr**(*src_files*, *dataset*, *time_window*, *snr_threshold*, *pearson_cutoff_factor=0.5*, *save_plots=True*, *underlay_rcf_xcorr=False*, *force_save=False*)

Process a batch of .nc files to generate standard visualization graphics. PNG files are output alongside the source .nc file. To suppress file output, set save_plots=False.

Parameters

- **src_files** (*Iterable of str*) – List of files to process
- **dataset** (*FederatedASDFDataset*) – Dataset to be used to ascertain the distance between stations.
- **time_window** (*float*) – Lag time window to plot (plus or minus this value in seconds)
- **snr_threshold** (*float*) – Minimum signal to noise ratio for samples to be included into the clock lag estimate
- **save_plots** – Whether to save plots to file, defaults to True
- **save_plots** – bool, optional
- **underlay_rcf_xcorr** – Show the individual correlation of row sample with RCF beneath the computed time lag, defaults to False
- **underlay_rcf_xcorr** – bool, optional

Returns List of files for which processing failed, and associated error.

Return type list(tuple(str, str))

seismic.xcorqc.xcorr_station_clock_analysis.**plot_estimated_timeshift**(*ax*, *x_lag*, *y_times*, *correction*, *annotation=None*, *row_rcf_crosscorr=None*)

seismic.xcorqc.xcorr_station_clock_analysis.**plot_pearson_corr_coeff**(*ax*, *rcf*, *ccf_masked*, *y_times*)

seismic.xcorqc.xcorr_station_clock_analysis.**plot_reference_correlation_function**(*ax*, *x_lag*, *rcf*, *rcf_corrected*, *snr_threshold*)

seismic.xcorqc.xcorr_station_clock_analysis.**plot_snr_histogram**(*ax*, *snr*, *time_window*, *nbins=10*)

seismic.xcorqc.xcorr_station_clock_analysis.**plot_stacked_window_count**(*ax*, *x_nsw*, *y_times*)

seismic.xcorqc.xcorr_station_clock_analysis.**plot_xcorr_file_clock_analysis**(*src_file*, *asdf_dataset*, *time_window*, *snr_threshold*, *pearson_correlation_factor*, *show=True*, *under-lay_rcf_xcorr=False*, *pdf_file=None*, *png_file=None*, *title_tag=''*, *settings=None*)

seismic.xcorqc.xcorr_station_clock_analysis.**plot_xcorr_time_series**(*ax*, *x_lag*, *y_times*, *xcorr_data*, *use_formatter=False*)

seismic.xcorqc.xcorr_station_clock_analysis.**read_correlator_config**(*nc_file*)

Read the correlator settings used for given nc file.

Parameters **nc_file** (*str*) – File name of the .nc file containing the cross-correlation data.

Returns Pandas Series with named fields whose values are the runtime settings used for the .nc file

Return type pandas.Series

seismic.xcorqc.xcorr_station_clock_analysis.**station_codes**(*filename*)

Convert a netCDF4 .nc filename generated by correlator to the corresponding station codes in the format `NETWORK.STATION`

Assumed format: `NET1.STAT1.NET2.STA2.*.nc`

Parameters **filename** (*str*) – The `.nc` filename from which to extract the station and network codes

Returns Station code for each station (code1, code2)

Return type tuple(str, str)

seismic.xcorqc.xcorr_station_clock_analysis.**station_distance**(*federated_ds*, *code1*, *code2*)

Determine the distance in km between a pair of stations at a given time.

Parameters

- **federated_ds** (*seismic.ASDFdatabase.FederatedASDFDataSet*) – Federated dataset to query for station coordinates
- **code1** (*str*) – Station and network code in the format NETWORK.STATION for first station
- **code2** (*str*) – Station and network code in the format NETWORK.STATION for second station

Returns Distance between stations in kilometers

Return type float

1.10.12 Module contents

HiPerSeis source code can be found in Github : https://github.com/GeoscienceAustralia/hiperseis

CHAPTER

TWO

FULL PACKAGE HIERARCHY

CHAPTER

THREE

SEISMIC

3.1 seismic package

3.1.1 Subpackages

3.1.2 Submodules

3.1.3 seismic.analyze_station_orientations module

Analyze a data set of seismic arrival events on a per-station basis and try to detect and estimate any station orientation error.

The event waveform dataset provided as input to this script (or to NetworkEventDataset if calling directly into function *analyze_station_orientations*) is generated by the script *seismic/extract_event_traces.py*.

In future, consider moving this script to the *inventory* module and applying corrections to the station inventory xml (to the azimuth tag).

Reference: Wilde-Piórko, M., Grycuk, M., Polkowski, M. et al. On the rotation of teleseismic seismograms based on the receiver function technique. J Seismol 21, 857-868 (2017). https://doi.org/10.1007/s10950-017-9640-x

seismic.analyze_station_orientations.**analyze_station_orientations**(*ned*, *curation_opts*, *config_filtering*, *config_processing*, *save_plots_path=None*)

Main processing function for analyzing station orientation using 3-channel event waveforms. Uses method of Wilde-Piorko https://doi.org/10.1007/s10950-017-9640-x

One should not worry about estimates that come back with error of less than about 20 degrees from zero, since this analysis provides only an estimate.

Parameters

- **ned** – NetworkEventDataset containing waveforms to analyze. Note: the data in this dataset will be modified by this function.
- **curation_opts** – Seismogram curation options. Safe default to use is *DEFAULT_CURATION_OPTS*.
- **config_filtering** – Seismogram filtering options for RF computation. Safe default to use is *DEFAULT_CONFIG_FILTERING*.

- **config_processing** – Seismogram RF processing options. Safe default to use is *DEFAULT_CONFIG_PROCESSING*.
- **save_plots_path** – Optional folder in which to save plot per station of mean arrival RF amplitude as function of correction angle

Returns Dict of estimated orientation error with net.sta code as the key.

seismic.analyze_station_orientations.**process_event_file**(*src_h5_event_file*, *curation_opts=None*, *config_filtering=None*, *config_processing=None*, *dest_file=None*, *save_plots_path=None*)

Use event dataset from an HDF5 file to analyze station for orientation errors.

Parameters

- **src_h5_event_file** – HDF5 file to load. Typically one created by *extract_event_traces.py* script
- **curation_opts** – Seismogram curation options. Safe default to use is *DEFAULT_CURATION_OPTS*.
- **config_filtering** – Seismogram filtering options for RF computation. Safe default to use is *DEFAULT_CONFIG_FILTERING*.
- **config_processing** – Seismogram RF processing options. Safe default to use is *DEFAULT_CONFIG_PROCESSING*.
- **dest_file** – File in which to save results in JSON format
- **save_plots_path** – Optional folder in which to save plot per station of mean arrival RF amplitude as function of correction angle

Returns None

3.1.4 seismic.extract_event_traces module

Use waveform database and station inventory to extract raw traces for all seismic events within a given magnitude and time range.

seismic.extract_event_traces.**asdf_get_waveforms**(*asdf_dataset*, *network*, *station*, *location*, *channel*, *starttime*, *endtime*)

Custom waveform getter function to retrieve waveforms from FederatedASDFDataSet.

Parameters

- **asdf_dataset** (*seismic.ASDFdatabase.FederatedASDFDataSet*) – Instance of FederatedASDFDataSet to query
- **network** (*str*) – Network code
- **station** (*str*) – Station code
- **location** (*str*) – Location code
- **channel** (*str*) – Channel code
- **starttime** (*str in UTC datetime format*) – Start time of the waveform query
- **endtime** (*str in UTC datetime format*) – End time of the waveform query

Returns Stream containing channel traces

Return type obspy.Stream of obspy.Traces

seismic.extract_event_traces.**get_events**(*lonlat*, *starttime*, *endtime*, *cat_file*, *distance_range*, *magnitude_range*, *early_exit=True*)

Load event catalog (if available) or create event catalog from FDSN server.

Parameters

- **lonlat** (*tuple(float, float)*) – (Longitude, latitude) of reference location for finding events
- **starttime** (*obspy.UTCDateTime or str in UTC datetime format*) – Start time of period in which to query events
- **endtime** (*obspy.UTCDateTime or str in UTC datetime format*) – End time of period in which to query events
- **cat_file** (*str or Path*) – File containing event catalog, or file name in which to store event catalog
- **distance_range** (*tuple(float, float)*) – Range of distances over which to query seismic events
- **magnitude_range** (*tuple(float, float)*) – Range of event magnitudes over which to query seismic events.
- **early_exit** (*bool, optional*) – If True, exit as soon as new catalog has been generated, defaults to True

Returns Event catalog

Return type obspy.core.event.catalog.Catalog

seismic.extract_event_traces.**is_url**(*resource_path*)

Convenience function to check if a given resource path is a valid URL

Parameters **resource_path** (*str*) – Path to test for URL-ness

Returns True if input is a valid URL, False otherwise

Return type bool

seismic.extract_event_traces.**timestamp_filename**(*fname*, *t0*, *t1*)

Append pair of timestamps (start and end time) to file name in format that is compatible with filesystem file naming.

Parameters

- **fname** (*str or path*) – File name
- **t0** (*obspy.UTCDateTime*) – first timestamp
- **t1** (*obspy.UTCDateTime*) – second timestamp

3.1.5 seismic.model_properties module

class seismic.model_properties.**LayerProps**(*vp*, *vs*, *rho*, *thickness*)
Bases: object

Helper class to contain layer bulk material properties

property H

property Vp

property Vs

property rho

3.1.6 seismic.network_event_dataset module

Class encapsulating a collection of event waveforms for stations of one network.

class seismic.network_event_dataset.**NetworkEventDataset**(*stream_src*, *network=None*, *station=None*, *location=''*, *ordering='ZNE'*)
Bases: object

Collection of 3-channel ZNE streams with traces aligned to a fixed time window about seismic P-wave arrival events, for a given network.

Two indexes are provided. One indexes hierarchically by station code and event ID, yielding a 3-channel ZNE stream per event, so that you can easily gather all traces for a given station by iterating over events.

The other index indexes hierarchically by event ID and station code, yielding a 3-channel ZNE stream per station. Using this index you can easily gather all traces for a given event across multiple stations.

Preferably each input trace will already have an 'event_id' attribute in its stats. If not, an event ID will be invented based on station identifiers and time window.

apply(*_callable*)
Apply a callable across all streams. Use to apply uniform processing steps to the whole dataset.

Parameters **_callable** (*Any callable compatible with the call signature.*) – Callable object that takes an obspy Stream as input and applies itself to that Stream. Expect that stream may be mutated in-place by the callable.

Returns None

by_event()
Iterate over event sub-dictionaries

Returns Iterable over the discrete events, each element consisting of pair containing (event id, station dict).

by_station()
Iterate over station sub-dictionaries

Returns Iterable over the stations, each element consisting of pair containing (station code, event dict).

curate(*curator*)
Curate the dataset according to a callable curator. Curator call signature takes station code, event id and stream as input, and returns boolean whether to keep Stream or not.

Parameters **curator** (*Callable*) – Function or callable delegate to adjudicate whether to keep each given stream.

Returns None

event(*event_id*)
Accessor for stations for a given event. :param event_id: ID of event to look up :return: Station index for given event, if event ID is found :rtype: SortedDict

num_events()
Get number of events in the dataset. :return: Number of events

num_stations()
Get number of stations in the dataset. :return: Number of stations

prune(*items*, *cull=True*)
Remove a given sequence of (station, event) pairs from the dataset.

Parameters

- **items** – Iterable of (station, event) pairs
- **cull** – If True, then empty entries in the top level index will be removed.

Returns None

station(*station_code*)
Accessor for events for a given station. :param station_code: Station to get :return: Event index for station, if station is found :rtype: SortedDict

write(*output_h5_filename*, *index_format='event'*)
Write event dataset back out to HDF5 file.

Parameters

- **output_h5_filename** (*str or path*) – Output file name
- **index_format** (*str*) – Format to use for index. Must be 'event' (default) or 'standard' (obspy default)

Returns True if file was written

3.1.7 seismic.plot_network_event_dataset module

Bulk plotting helper functions based on NetworkEventDataset

seismic.plot_network_event_dataset.**plot_ned_seismograms**(*ned*, *output_file*, *channel_order='ZNE'*)
Plot seismograms in NetworkEventDataset to PDF file. If dataset is very large, this may take a long time to run.

Parameters

- **ned** – NetworkEventDataset containing waveforms.
- **output_file** – Output file name

3.1.8 seismic.stream_io module

Helper functions for seismic stream IO.

seismic.stream_io.**get_obspyh5_index**(*src_file*, *seeds_only=False*)

seismic.stream_io.**iter_h5_stream**(*src_file*, *headonly=False*)

Iterate over hdf5 file containing streams in obspyh5 format.

Parameters

- **src_file** – str or path to file to read
- **headonly** – Only read trace stats, do not read actual time series data

Yield obspy.Stream containing traces for a single seismic event.

seismic.stream_io.**read_h5_stream**(*src_file*, *network=None*, *station=None*, *loc=''*, *root='/waveforms'*)

Helper function to load stream data from hdf5 file saved by obspyh5 HDF5 file IO. Typically the source file is generated using *extract_event_traces.py* script. For faster loading time, a particular network and station may be specified.

Parameters

- **src_file** (*str or Path*) – File from which to load data
- **network** (*str, optional*) – Specific network to load, defaults to None
- **station** (*str, optional*) – Specific station to load, defaults to None
- **root** (*str, optional*) – Root path in hdf5 file where to start looking for data, defaults to '/waveforms'

Returns All the loaded data in an obspy Stream.

Return type obspy.Stream

seismic.stream_io.**sac2hdf5**(*src_folder*, *basenames*, *channels*, *dest_h5_file*, *tt_model_id='iasp91'*)

Convert collection of SAC files from a folder into a single HDF5 stream file.

Parameters

- **src_folder** (*str or Path*) – Path to folder containing SAC files
- **basenames** (*list of str*) – List of base filenames (file name excluding extension) to load.
- **channels** (*List of str*) – List of channels to load. For each base filename in basenames, there is expected to be a file with each channel as the filename extension.
- **dest_h5_file** (*str or Path*) – Path to output file. Will be created, or overwritten if already exists.
- **tt_model_id** (*str*) – Which travel time earth model to use for synthesizing trace metadata. Must be known to obspy.taup.TauPyModel

Returns None

seismic.stream_io.**write_h5_event_stream**(*dest_h5_file*, *stream*, *mode='a'*, *ignore=()*)

Write stream to HDF5 file in event indexed format using obspy.

Parameters

- **dest_h5_file** (*str or pathlib.Path*) – File in which to write the stream.
- **stream** (*obspy.Stream*) – The stream to write

- **mode** (*str*) – Write mode, such as 'w' or 'a'. Use 'a' to iteratively write multiple streams to one file.
- **ignore** (*Any iterable of str*) – List of headers to ignore when writing attributes to group. Passed on directly to obspyh5.writeh5

3.1.9 seismic.stream_processing module

Utility stream processing functions.

seismic.stream_processing.**assert_homogenous_stream**(*stream*, *funcname*)
Verify that the given stream does not contain mixture of stations or channels/components.

Parameters **stream** (*obspy.core.stream.Stream or rf.RFStream*) – Stream containing one or more traces

seismic.stream_processing.**back_azimuth_filter**(*baz*, *baz_range*)
Check if back azimuth *baz* is within range. Inputs must be in the range [0, 360] degrees.

Parameters

- **baz** (*int or float*) – Value to check
- **baz_range** (*List or array of 2 floats, min and max back azimuth*) – Pair of angles in degrees.

Returns True if baz is within baz_range, False otherwise.

seismic.stream_processing.**correct_back_azimuth**(*_event_id*, *stream*, *baz_correction*)
Apply modification to the back azimuth value in the stream stats

Parameters

- **_event_id** – Ignored
- **stream** – Stream to which correction is applied
- **baz_correction** – Any object with a registered *scalarize* function for generating an angle correction for a trace in degrees. E.g. could be a numeric value, a dictionary of correction values, or a file produced by script *analyze_station_orientations.py*

Returns Stream with modified back azimuth

seismic.stream_processing.**negate_channel**(*_event_id*, *stream*, *channel*)
Negate the data in the given channel of the stream

Parameters

- **_event_id** – Ignored
- **stream** – Stream containing channel to flip
- **channel** – Single character string indicating which component to flip

Returns Stream with channel data negated

seismic.stream_processing.**scalarize**(*_obj*, *_stats*)
seismic.stream_processing.**scalarize**(*val: numbers.Number*, *_stats*)
seismic.stream_processing.**scalarize**(*d: dict*, *stats*)
seismic.stream_processing.**scalarize**(*filename: str*, *stats*)

seismic.stream_processing.**sinc_resampling**(*t*, *y*, *t_new*)
Resample signal y for known times t onto new times t_new. Sampling rates do not need to match and time windows do not need to overlap. t_new should not have a lower sampling rate than t.

Parameters

- **t** – numpy.array of times
- **y** – numpy.array of sample values
- **t_new** – numpy.array of new times to interpolate onto

Returns numpy.array of new interpolated sample values

seismic.stream_processing.**swap_ne_channels**(*_event_id*, *stream*)
Swap N and E channels on a stream. Changes the input stream.

Parameters

- **_event_id** – Ignored
- **stream** – Stream whose N and E channels are to be swapped

Returns Stream with channel swapping applied

seismic.stream_processing.**zne_order**(*tr*)
Channel ordering sort key function

Parameters **tr** (*obspy.Trace or RFTrace*) – Trace whose ordinal is to be determined.

Returns Numeric index indicating ZNE sort order of traces in a stream

seismic.stream_processing.**zrt_order**(*tr*)
Channel ordering sort key function

Parameters **tr** (*obspy.Trace or RFTrace*) – Trace whose ordinal is to be determined.

Returns Numeric index indicating ZRT sort order of traces in a stream

3.1.10 seismic.stream_quality_filter module

Helper functions for curating and quality controlling stream objects.

seismic.stream_quality_filter.**curate_stream3c**(*ev_id*, *stream3c*, *logger=None*)
Apply quality curation criteria to a stream. Modifies the stream in-place if required. Traces in stream must be in ZNE order. Each trace in the stream is expected to have metadata with starttime, endtime, channel and inclination.

The following checks are made. If any of these checks fails, the function returns False: * Inclination value is not NaN * The stream has 3 channels * Each trace in the stream has the same number of samples * None of the traces have any NaN values in the time series data * None of the traces have zero variance

The following cleanups are attempted on the stream: * All 3 traces have the same time range

Parameters

- **ev_id** (*int or str*) – The event id
- **stream3c** (*obspy.Stream*) – Stream with 3 components of trace data
- **logger** (*logger.Logger object*) – Logger in which to log messages

Returns True if checks pass, False otherwise

3.1.11 seismic.units_utils module

Constants and utility functions for unit conversion.

3.1.12 Module contents

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

D

E

F

G

M

N

O

P

Q

R

S

T

U

V

W

X

Z