HiPerSeis Documentation

Release 1.0.0

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

Oct 10, 2023

CONTENTS

HiPerSeis source code can be found in Github : https://github.com/GeoscienceAustralia/hiperseis

CHAPTER

ONE

API REFERENCE BY TOPICS

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

Object to query IRIS or other client with a bounding box and time interval. Returns ASDF containing station information and data for interval.

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

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*, *min_length_sec*, *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

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*)

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.plot_data_quality module

1.1.8 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".

1.1.9 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*)

1.1.10 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

1.1.11 seismic.ASDFdatabase.utils module

1.1.12 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

1.3.3 seismic.gps_corrections.picks_reader_utils module

1.3.4 seismic.gps_corrections.relative_tt_residuals_plotter module

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
```

(continues on next page)

(continued from previous page)

```
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.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

1.4.4 seismic.inventory.engd2stxml module

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

1.4.7 seismic.inventory.inventory_merge module

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 pathlib.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

Create new instance of Instrument(sensor, response)

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*, *blacklisted_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.core.inventory.inventory.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 collections.namedtuple("Instrument", ['sensor', 'response'])

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

Load a stationxml file

Parameters
inventory_file (*str or path*) – Name of stationxml file to load

Returns
Station inventory as Obspy Inventory

Return type
obspy.core.inventory.inventory.Inventory

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

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

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

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

1.4.12 seismic.inventory.plotting module

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

Constructor

Parameters

source_inventory (*obspy.core.inventory.inventory.Inventory*) – Inventory from which to extract response

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: *ResponseFromInventory*

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

Constructor

Parameters
respFileName (*str*) – XML file to load

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

1.4.15 seismic.inventory.update_iris_inventory module

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

Output 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

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** (*Python dict of key-values pairs matching command line options, optional*) – Filtering options for network, station and channel codes, defaults to None

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

seismic.inversion.mcmc.plot_inversion module

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

seismic.inversion.wavefield_decomp.runners module

Batch execution interfaces for wavefield continuation methods and solvers.

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** (*str or pathlib.Path*) – File from which to load solution
- **job_timestamp** (*str or NoneType*) – Timestamp of job whose solution is to be loaded
- **logger** (*logging.Logger*) – Output logging instance

Returns
(solution, job configuration), job timestamp

Return type
(solution, dict), str

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** (*numpy.array*) – 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** (seismic.model_properties.LayerProps) – Mantle properties stored in class LayerProps instance.
- **Vp** (*numpy.array*) – Array of Vp values ordered by layer.
- **rho** (*numpy.array*) – Array of rho values ordered by layer.
- **flux_window** ((*float, float*)) – 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 obspy.Stream objects*) – Collection of waveform data to process
- **config** (*dict*) – Dict of job settings. See example files for fields and format of settings.
- **logger** (*logging.Logger or NoneType*) – Log message receiver. Optional, pass None for no logging.

Returns
Solution object based on scipy.optimize.OptimizeResult

Return type
scipy.optimize.OptimizeResult with additional custom attributes

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

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

Return type
(solution, dict)

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. In general soln_configs will be an ensemble of per-station solutions.

Parameters

- **soln_configs** (*list((solution, dict))*) – List of (solution, configuration) pairs to save (one per station).
- **input_file** (*str*) – Name of input file used for the job. Saved to job node for traceability
- **output_file** (*str or pathlib.Path*) – Name of output file to create
- **job_timestamp** (*str*) – Job timestamp that will be used to generate the top level job group
- **job_tracking** (*dict or dict-like*) – Dict containing job identification information for traceability
- **logger** (*logging.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** (*Callable(*args) -> float*) – Objective function to minimize. Takes unpacked args as function call arguments and returns a float.
- **bounds** (*scipy.optimize.Bounds*) – Bounds of the parameter space.
- **args** (*tuple or list*) – Any additional fixed parameters needed to completely specify the objective function.
- **x0** (*numpy.array with same shape as elements of bounds*) – Initial guess. If None, will be selected randomly and uniformly within the parameter bounds.
- **T** (*float*) – 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** (*int*) – Maximum number of minima to return
- **burnin** (*int*) – Number of random steps to discard before starting to accumulate statistics.
- **maxiter** (*int*) – Maximum number of steps to take (including burnin).
- **target_ar** (*float between 0 and 1*) – Target acceptance rate of point samples generated by stepping.
- **ar_tolerance** (*float*) – Tolerance on the acceptance rate before actively adapting the step size.
- **cluster_eps** (*float*) – Point proximity tolerance for DBSCAN clustering, in normalized bounds coordinates.
- **rnd_seed** (*int*) – Random seed to force deterministic behaviour
- **collect_samples** (*int or NoneType*) – If not None and integral type, collect collect_samples at regular intervals and return as part of solution.
- **logger** – Logger instance for outputting log messages.

Returns
OptimizeResult containing solution(s) and solver data.

Return type
scipy.optimize.OptimizeResult with additional attributes

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.**WfContinuationSuFluxComputer**(*station_*
f_s,
time_wi
cut_win

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.

Constructor

Parameters

- **station_event_dataset** – Iterable container of obspy.Stream objects.
- **f_s** – Processing sample rate. Usuually much less than the sampling rate of the input raw seismic traces.
- **time_window** – Time window about onset to use for wave continuation processing.
- **cut_window** – Shorter time segment within time_window to from which to extract primary arrival waveform and its multiples.

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(seismic.model_properties.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(seismic.model_properties.LayerProps)*) – List of LayerProps through which to propagate the surface seismograms.

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

Return type
numpy.array

`times()`

Get array of discrete time values of the time axis

Returns
Array of evenly spaced time values

Return type
numpy.array

seismic.inversion.wavefield_decomp.wfd_plot module

Module contents

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

1.6.3 seismic.pick_harvester.pick module

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.`**`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.utils.**Catalog**(*event_folder*)
Bases: object

get_events()

get_preferred_origin_timestamps()

class seismic.pick_harvester.utils.**CatalogCSV**(*event_folder*, *events_only=True*)
Bases: object

get_events()

get_preferred_origin_timestamps()

class seismic.pick_harvester.utils.**Event**
Bases: object

origin_list

preferred_magnitude

preferred_origin

public_id

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

magnitude_type

magnitude_value

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

arrival_list

depthkm

lat

lon

magnitude_list

utctime

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

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*, *network_list='*'*, *station_list='*'*, *only_corrections=False*)

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 "correction" for rotating/swapping/negating channel data for one or more named stations with potential orientation discrepancies. 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":
  {
    "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
```

(continues on next page)

(continued from previous page)

```
                        # 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
}

"correction": # corrections to be applied to data for named stations prior to RF␣
↪computation
{
  "plot_dir": str # path to folder where plots related to orientation corrections␣
↪are to be saved
  "swap_ne": list # list of NET.STA.LOC for which N and E channels are to be␣
↪swapped, e.g ["OA.BL27."],
  "rotate": list # list of NET.STA.LOC that are to be rotated to maximize P-arrival␣
↪energy on                          the primary RF component, e.g ["OA.BL27."]
  "negate": list # list of NET.STA.LOC.CHA that are to be negated, e.g ["OA.BL27..
↪HHZ"]
}
```

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

1.7.4 seismic.receiver_fn.plot_ccp module

1.7.5 seismic.receiver_fn.plot_ccp_batch module

1.7.6 seismic.receiver_fn.plot_spatial_map module

Use cartopy to plot moho grid and gradient onto a map.

`seismic.receiver_fn.plot_spatial_map.`**`from_params`**(*params*)

Create plots from WorkflowParameters as part of moho workflow.

`seismic.receiver_fn.plot_spatial_map.`**`plot_spatial_map`**(*grid_data*, *gradient_data*, *methods_datasets=None*, *projection_code=None*, *title=None*, *feature_label=None*, *bounds=None*, *scale=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
- **bounds** – 4 element tuple of (L, R, B, T) for limiting plot extent
- **scale** – 2 element tuple of (vmin, vmax) for limiting colormap scale

Returns

1.7.7 seismic.receiver_fn.pointsets2grid module

Blend data from multiple methods together onto a common grid.

Multiple methods with per-method settings are passed in using a JSON configuration file.

See *config_moho_workflow_example.json* and the wiki page at https://github.com/GeoscienceAustralia/hiperseis/wiki/Blending-and-Plotting-Point-Datasets for an explanation of the config.

Bounds are optional. If provided, the interpolation grid will be limited to this extent. If not provided, the interpolation grid is bounded by the maximum extent of the aggregate datasets.

The gridded data will be written to output directory as 'moho_grid.csv'. If output directory is not provided, the current working directory will be used.

The gridded gradient will be written to output directory as 'moho_gradient.csv'.

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

`seismic.receiver_fn.pointsets2grid.`**make_grid**(*params*)

Run multi point dataset weighted averaging over Gaussian interpolation functions to produce aggregate dataset. Source data coordinates are assumed to be in lon/lat (WGS84). First 2 rows of output file contain grid dimensions, followed by CSV data.

1.7.8 seismic.receiver_fn.rf_3dmigrate module

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*)

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.

Yields (station_id, event_id, event_time, stream)

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.

Initializer

Parameters

- **h5_filename** (*str or pathlib.Path*) – Name of file containing event seismograms in HDF5 format, indexed in seismic.stream_io.EVENTIO_H5INDEX format
- **memmap** (*bool*) – If True, memmap the open file. Can improve tractability of handling very large files.
- **channel_pattern** (*str*) – [OPTIONAL] Ordered list of preferred channels, e.g. 'HH*,BH*'. Channel mask to use to select channels returned.

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.

This class yields 3-channel traces per station per event.

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

Initializer

Parameters

- **h5_filename** (*str or pathlib.Path*) – Name of file containing event seismograms in HDF5 format, indexed in seismic.stream_io.EVENTIO_H5INDEX format
- **memmap** (*bool*) – If True, memmap the open file. Can improve tractability of handling very large files.

1.7.12 seismic.receiver_fn.rf_handpick_tool module

1.7.13 seismic.receiver_fn.rf_inversion_export module

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. Note that location codes are not taken into account.

Initialize from rf.RFStream

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

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

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*, *Vp=6.5*, *h_range=None*, *k_range=None*, *weights=array([0.5, 0.4, 0.1])*, *root_order=1*, *semblance_weighted=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.
- **Vp** (*float, optional*) – Average crustal Vp for computing H-k stacks
- **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)
- **weights** (*numpy.array [1D], optional*) – numpy array of length 3, containing weights for the three phases (Ps, PpPs and (PpSs + PsPs))
- **root_order** (*int, optional*) – Exponent for nth root stacking as per K.J.Muirhead (1968), defaults to 1

Returns
k-grid values [2D], h-grid values [2D], H-k stack values [2D]

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

seismic.receiver_fn.rf_stacking.**compute_sediment_hk_stack**(*cha_data*, *H_c*, *k_c*, *Vp=6.5*, *h_range=None*, *k_range=None*, *root_order=9*)

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.
- **H_c** (*float, optional*) – Crustal thickness estimate from H-k stack

- **k_c** (*float, optional*) – Crustal Vp/Vs ratio estimate from H-k stack
- **Vp** (*float, optional*) – Average crustal Vp for computing H-k stacks
- **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

Returns
k-grid values [2D], h-grid values [2D], H-k stack values [2D]

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

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_sum*, *max_number=3*)

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_by_distance**(*rf_stream*, *min_dist*, *max_dist*)

Discard RFs that fall outside the distance range(min_dist, max_dist) @param rf_stream: RFStream @param min_dist: minimum angular distance @param max_dist: maximum angular distance @return: trimmed RFStream

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_invalid_radial_component**(*rf_stream*)

Filter out invalid radial RFs with amplitudes > 1 or troughs around onset time :param rf_stream: Stream of RF traces to filter, should be **for a single component of a single station** :type rf_stream: rf.RFStream :return: Filtered RF stream :rtype: 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.Stream*) – 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.Trace*) – 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

seismic.receiver_fn.rf_util.**trim_hdf_keys**(*hdf_key_list: [<class 'str'>], networks_string: str, stations_string: str*) → [<class 'str'>]

Trims a list of hdf_keys, filtering out unwanted networks and stations. @param hdf_key_list: @param networks_string: a space-separated list of networks. '*' includes all. @param stations_string: a space-separated list of stations or a text file

with station names in each row, w/wo location codes. '*' includes all.

@return: trimmed list

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: *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',))
```

Initialization

Parameters

- **station_latlon** – See documentation for *synthesize()*
- **earth_model** (*str*) – String naming which standard earth model to use.

synthesize(*src_latlon*, *fs*, *time_window*)

See documentation for *synthesize()*

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

Getter for backend Synthesizer class

Returns
Class name

Return type
SynthesizerSyngine

seismic.synthetics.backends.backend_tws module

Backend for making synthetic seismograms using Telewavesim.

class seismic.synthetics.backends.backend_tws.**SynthesizerMatrixPropagator**(*station_latlon*, *layerprops*)

Bases: *Synthesizer*

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

Initialization

Parameters

- **station_latlon** – See documentation for *synthesize()*
- **layerprops** (*list*(seismic.model_properties.LayerProps)) – List of LayerProps. Last layer should be mantle properties.

property kappa

Return ratio of Vp/Vs for each layer

Returns
k value per layer

Return type
numpy.array

synthesize(*src_latlon*, *fs*, *time_window*)

See documentation for *synthesize()*

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

Getter for backend Synthesizer class

Returns
Class name

Return type
SynthesizerMatrixPropagator

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.

Initialization

Parameters
station_latlon (*tuple(float, float) or str*) – Either a tuple of (lat, lon) coordinates, or a station code in the format 'NET.STA' string.

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** (*float*) – Source latitude
- **src_lon** (*float*) – Source longitude
- **eventid_base** (*str*) – Base string for event id
- **src_depth_m** (*float*) – Source depth in metres
- **earth_model** (*str*) – String name of earth model to use for ray tracing
- **phase** (*str*) – Which phase is being modelled
- **origin_time** (*obspy.UTCDateTime*) – Timestamp of the source event. If empty, will be a random offset from now.

Returns
Stats dictionary

Return type
dict

property station_latlon

Get (latitude, longitude) location of receiving station

Returns
Location (latitude, longitude) of receiving station

Return type
tuple(float, float)

abstract synthesize(*src_latlon*, *fs*, *time_window*)

Function signature for function to compute synthetic dataset of obspy streams.

Parameters

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

Returns
obspy.Stream containing ZNE velocity seismogram

Return type
obspy.Stream

Module contents

1.8.2 Submodules

1.8.3 seismic.synthetics.synth module

Entry point for making synthetic seismograms.

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** (*str*) – 'propmatrix' or 'syngine'
- **output_file** (*str*) – Destination file in which to write resultant streams in HDF5 format.
- **net** (*str*) – Network code of receiver
- **sta** (*str*) – Station code of receiver
- **src_latlon** (*iterable of pairs*) – Iterable of source (lat, lon) locations
- **fs** (*float*) – Sampling rate
- **time_window** (*tuple(float, float)*) – Time window about onset. First value should be < 0, second should be > 0

Returns
Whether the dataset was successfully created.

Return type
bool

1.8.4 Module contents

1.9 seismic.traveltime package

1.9.1 Submodules

1.9.2 seismic.traveltime.cluster_grid module

1.9.3 seismic.traveltime.events_stations_rays_visualization2 module

1.9.4 seismic.traveltime.gather_events module

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

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

1.9.11 seismic.traveltime.zone_rays module

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

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*)

1.10.7 seismic.xcorqc.generate_test_data module

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

1.10.8 seismic.xcorqc.utils module

1.10.9 seismic.xcorqc.validate_xcorr_setup module

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

1.10.10 seismic.xcorqc.xcorqc module

1.10.11 seismic.xcorqc.xcorr_station_clock_analysis module

1.10.12 Module contents

CHAPTER

TWO

SEISMIC PACKAGE FULL HIERARCHY

2.1 seismic package

2.1.1 Subpackages

2.1.2 Submodules

2.1.3 seismic.analyze_station_orientations module

2.1.4 seismic.extract_event_traces module

2.1.5 seismic.model_properties module

Helper classes to encapsulate model properties.

class seismic.model_properties.**LayerProps**(*vp*, *vs*, *rho*, *thickness*)

Bases: object

Helper class to contain layer bulk material properties

Constructor for given properties

Parameters

- **vp** (*float*) – P-wave body wave velocity
- **vs** (*float*) – S-wave body wave velocity
- **rho** (*float*) – Bulk material density
- **thickness** (*float*) – 1D (vertical) thickness of the layer.

property H

Get layer thickness

property Vp

Get P-wave body wave velocity

property Vs

Get S-wave body wave velocity

property rho

Get bulk material density

2.1.6 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'*, *root='/waveforms'*)

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.

Initialize from data source (file or obspy.Stream). Traces are COPIED into the dataset in order to leave input object intact, since many obspy functions mutate traces in-place.

All streams in the input data source stream_src are expected to belong to the same network. This is checked as the data is ingested. A discrepant network code is an error condition.

Parameters

- **stream_src** (*str, pathlib.Path or obspy.Stream*) – Source of input streams. May be a file name or an Obspy Stream
- **network** (*str*) – Network code of streams to load. If stream_src is an Obspy Stream, the streams will be filtered to match this network code.
- **station** (*str*) – Station code of streams to load. If stream_src is an Obspy Stream, the streams will be filtered to match this station code.
- **location** (*str*) – [OPTIONAL] Location code of streams to load. Leave as default (empty string) if location code is empty in the data source.
- **ordering** (*str*) – Channel ordering to be applied to the data after loading. The channel labelling must be consistent with the requested ordering - rotation to the coordinate system implied by the ordering is *NOT* applied.

Raises

AssertionError – If discrepant network code is found in input data

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).

Return type
Iterable(tuple)

`by_station()`

Iterate over station sub-dictionaries.

Returns
Iterable over the stations, each element consisting of pair containing (station code, event dict).

Return type
Iterable(tuple)

`curate(curator)`

Curate the dataset according to a callable curator. Modifies collection in-place to remove streams that do not satisfy the curation criteria of the callable. Curator call signature must be consitent with:

```
callable(station_code, event_id, stream) -> bool
```

The callable returns a boolean indicating whether to keep the 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.

Parameters
event_id (*str*) – ID of event to look up

Returns
Station index for given event, if event ID is found, otherwise None

Return type
SortedDict or NoneType

`num_events()`

Get number of events in the dataset.

Returns
Number of events

Return type
int

`num_stations()`

Get number of stations in the dataset.

Returns
Number of stations

Return type
int

prune(*items*, *cull=True*)

Remove a given sequence of (station, event) pairs from the dataset.

Parameters

- **items** (*Iterable(tuple)*) – Iterable of (station, event) pairs
- **cull** (*boolean*) – 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.

Parameters
station_code (*str*) – Station to get

Returns
Event index for station, if station is found

Return type
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

Return type
boolean

2.1.7 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

2.1.8 seismic.stream_io module

Helper functions for seismic stream IO.

seismic.stream_io.**get_obspyh5_index**(*src_file*, *seeds_only=False*, *root='/waveforms'*)

Scrape the index (only) from an obspyh5 file.

Parameters

- **src_file** (*str or pathlib.Path*) – Name of file to extract index from
- **seeds_only** (*bool*) – If True, only get the seed IDs of the traces. Otherwise (default), get full index.

Returns
Sorted dictionary with index of waveforms in the file

Return type
sortedcontainers.SortedDict

seismic.stream_io.**iter_h5_stream**(*src_file*, *headonly=False*, *root='/waveforms'*)

Iterate over hdf5 file containing streams in obspyh5 format.

Parameters

- **src_file** (*str or pathlib.Path*) – Path to file to read
- **headonly** (*bool*) – 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.**remove_group**(*hdf_fn*, *net_sta_loc*, *root='/waveforms'*, *logger=None*)

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.**safe_h5_root**(*src_file*, *root*)

seismic.stream_io.**safe_iter_event_data**(*events*, *inventory*, *get_waveforms*, *use_rfstats=True*, *phase='P'*, *request_window=None*, *pad=10*, *pbar=None*, ***kwargs*)

Return iterator yielding three component streams per station and event.

Parameters

- **events** – list of events or *~obspy.core.event.Catalog* instance
- **inventory** – *~obspy.core.inventory.inventory.Inventory* instance with station and channel information
- **get_waveforms** – Function returning the data. It has to take the arguments network, station, location, channel, starttime, endtime.
- **phase** – Considered phase, e.g. 'P', 'S', 'PP'
- **request_window** (*tuple (start, end)*) – requested time window around the onset of the phase
- **pad** (*float*) – add specified time in seconds to request window and trim afterwards again
- **pbar** – tqdm instance for displaying a progressbar
- **kwargs** – all other kwargs are passed to *~rf.rfstream.rfstats()*

Returns
three component streams with raw data

Example usage with progressbar:

```
from tqdm import tqdm
from rf.util import iter_event_data
with tqdm() as t:
    for stream3c in iter_event_data(*args, pbar=t):
        do_something(stream3c)
```

seismic.stream_io.**write_h5_event_stream**(*dest_h5_file*, *stream*, *index='waveforms/{network}.{station}.{location}/{event_time.datetime:%Y-%m-%dT%H:%M:%S}/{channel}_{starttime.datetime:%Y-%m-%dT%H:%M:%S}_{endtime.datetime:%Y-%m-%dT%H:%M:%S}'*, *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

2.1.9 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.Stream or rf.RFStream*) – Stream containing one or more traces

Returns
None

seismic.stream_processing.**back_azimuth_filter**(*back_azi*, *back_azi_range*)

Check if back azimuth *back_azi* is within range. Inputs must be in the range [0, 360] degrees.

Parameters

- **back_azi** (*int or float*) – Value to check
- **back_azi_range** (*List or array of 2 floats, min and max back azimuth*) – Pair of angles in degrees.

Returns
True if back_azi is within back_azi_range, False otherwise.

Return type
bool

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** (*obspy.Stream or rf.RFStream*) – 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 *rf_station_orientations.py*

Returns
Stream with modified back azimuth

Return type
Same as type(stream)

seismic.stream_processing.**negate_channel**(*_event_id*, *stream*, *channel*)

Negate the data in the given channel of the stream

Parameters

- **_event_id** – Ignored
- **stream** (*obspy.Stream*) – Stream containing channel to flip

- **channel** (*str*) – Single character string indicating which component to flip

Returns
Stream with channel data negated

Return type
obspy.Stream

seismic.stream_processing.**scalarize**(*_obj*, *_stats*)

seismic.stream_processing.**scalarize**(*val: Number*, *_stats*)

seismic.stream_processing.**scalarize**(*d: dict*, *stats*)

seismic.stream_processing.**scalarize**(*filename: str*, *stats*)

Fallback scalarize function for non-specialized type

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*) – 1D array of times
- **y** (*numpy.array*) – 1D array of sample values
- **t_new** (*numpy.array*) – 1D array of new times to interpolate onto

Returns
1D array of new interpolated sample values

Return type
numpy.array

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** (*obspy.Stream*) – Stream whose N and E channels are to be swapped

Returns
Stream with channel swapping applied

Return type
obspy.Stream

seismic.stream_processing.**zerophase_resample**(*item*, *resample_hz*)

seismic.stream_processing.**zne_order**(*tr*)

Channel ordering sort key function for ZNE ordering

Parameters
tr (*obspy.Trace or rf.RFTrace*) – Trace whose ordinal is to be determined.

Returns
Numeric index indicating ZNE sort order of traces in a stream

Return type
int

seismic.stream_processing.**zrt_order**(*tr*)

Channel ordering sort key function for ZRT ordering

Parameters
tr (*obspy.Trace or rf.RFTrace*) – Trace whose ordinal is to be determined.

Returns
Numeric index indicating ZRT sort order of traces in a stream

Return type
int

2.1.10 seismic.stream_quality_filter module

Helper functions for curating and quality controlling stream objects.

seismic.stream_quality_filter.**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** (seismic.network_event_dataset.NetworkEventDataset) – NetworkEventDataset containing seismograms to curate. Data will be modified by this function.
- **curation_opts** (*dict*) – Dict containing curation options.
- **logger** (*logging.Logger*) – Logger for emitting log messages
- **rotate_to_zrt** (*bool*) – Whether to automatically rotate to ZRT coords.

Returns
None, curation operates directly on data_all

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** (*logging.Logger*) – Logger in which to log messages

Returns
True if checks pass, False otherwise

Return type
bool

2.1.11 seismic.units_utils module

Constants and utility functions for unit conversion.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

D

E

F

G

N

O

P

Q

R

S