ETH Zürich

ants

Ambient noise tool - Manual

Laura Ermert
laura.ermert@erdw.ethz.ch
Prof. Andreas Fichtner

With contributions from:
Evan Delaney
Myrna Staring

March 9, 2016

Contents

1 Introduction

1.1 About this manual

Hint: The yellow boxes contain various caveats and hints.

Example: The boxes in this friendly color provide an example use case. Important: All python commands of the example have to be called in the working directory ANT/. For a quick tour, once you have set up the scripts and installed all dependencies, you can walk through the manual using only the blue boxes as tutorial.

1.2 Requirements

- Python 2.7.x (running on 2.7.11) – it is easiest to get anaconda python, which contains many dependencies for obspy already
- openmpi (anaconda will provide this with, type conda install openmpi)
- mpi4py (try to install with pip, conda installation is faulty)
- obspy including dependencies

1.3 Overview: Download, processing, correlation

The tool has three main routines:

- **Download** is done by `par_download.py`. This downloads data from IRIS DMC. Input details must be provided in an xml file and an ascii file with the station list. Raw data are saved as MSEED files to `DATA/raw/latest/`. An alternative is to download data with an IRIS breq_fast request (slower but in some cases more reliable).
- **Preprocessing** is done by `ant_proc.py`. This includes downsampling, instrument correction and bandpass filtering. Input must be provided in a python input file (example at INPUT/input_correction.py). Preprocessed data are saved to `DATA/processed/<name_tag>` as MSEED files with a 'name tag' specific to this processing.
- **Correlation** is done by `ant_corr.py`. Input must be specified in a python input file and an ascii file with the channel list. Correlation stacks are saved as SAC files to `DATA/correlations/<name_tag>` with a 'name tag' specific to this correlation.

All three scripts have the possibility to update with additional data.

1.4 Setting up ANTS; antconfig.py file

ANTS can be obtained from github (https://github.com/echolite/ANTS). Once downloaded, please edit the default locations of the directories where correlations etc. are stored, in the `antconfig.py` file. A setup file **has to be run once** to initialize the folder structure:

Example: Run

```
python ants_setup.py
```

once to get the folder structure started.

1.5 Tags

Both processed and correlated data sets are organized by 'name tags', e.g. something like 'hum-jan05'.

The tag can identify a dataset all the way through processing, correlation and plotting.

Let's construct a first dataset called 'noisy'. Create a folder:

```
cd DATA/raw/

mkdir noisy
```

2 Download data from IRIS

2.1 Requirements

`download_fetchdata` is a python wrapper for the IRIS FetchData script which accesses the web service.

Although convenient, webservice requests sometimes fail to return data, although these data should be archived in the IRIS database. This affects both `FetchData` and obspy.fdsn client. In the case data look suspiciously incomplete, it may be better to use `breqfast (http://ds.iris.edu/SeismiQuery/breq_fast.phtml)`

Requirements to run `par_download.py`:

1. A reasonably current version of the FetchData perl script, which is already in the tool and can be updated if needed at

   `https://seiscode.iris.washington.edu/projects/ws-fetch-scripts/files.`

   If updated from the web, the script must be renamed to FetchData (i. e. remove the version number from the name), and it must be made executable by typing

   `chmod +x FetchData`

2. An xml input file specifying a number of input parameters at `INPUT/input_download.xml`, explained below.

3. A list of station ids to be downloaded at `INPUT/downloadlist.txt`, also explained in more detail below.

The data will be downloaded and stored as miniseed files in a subfolder of your data directory, called `raw/latest/`. A report detailing the download and which identifiers did not return data will be stored in `raw/latest/downloadreport.txt`.

2.2 How to run parallel download

With the station id list and input file provided, download can is done in parallel: Every processor just gets a share of the station ids to download. To run, you need open MPI and mpi4py. Once these are installed, just run

```
mpirun -np <nr. of cores> python par_download.py
```

in the directory where python script par_download.py is located.

2.3 The xml input

An example inputfile is provided. Here is a short explanation of the parameters:

- verbose(0 or 1): If 1, the -v flag is set for FetchData. This prints some information about data availability, download speed and the downloaded data volume to the terminal window.
- exdir: The directory where FetchData perlscript is located
- user: An email address. This is at the moment irrelevant, but included hoping that the script can be extended to download data via arclink from the European Datacenter, Orfeus. The arclink protocol requires users to provide an email adress, so that if the data center has trouble coping with the request, they can directly interact with users.
- ids: Path to the file with the station ids of the stations you would like to download data from. The purpose of this file will be discussed in the next paragraph.
- time: Contains three parameters, starttime, endtime and minlen. The first two are the start-end endtimes for the period when you would like to request data. The format is: YYYY-MM-DD,HH:MM:SS. The last is the minimum length of segments you would like to retrieve in seconds.
- region: Contains four parameters, lat_min, lat_max, lon_min, lon_max: Select a geographic region, if needed.

2.4 The station id list

This list provides the selection of station channels from which data are downloaded. Again, an example file is provided. Each entry has to have the format

```
network.station..channel
```

and entries have to be separated by returns. You can wildcard single letters with '?' which is very convenient to specify channels, for example: LH?. Also, you can still wildcard all stations in a network with one ??? entry for the three-letter station names and one ???? for those with four letters.
Don't use * as wildcard as it appears not to work.
Although it seems tedious to compile an explicit list of all the stations, this is an easy way to control what your data set will contain. You can look up and copy station lists, and the id list gives you the flexibility to add specific stations from other networks without having to download all the network's data. Moreover the list lends itself to easy parallelization.

2.4.1 A note on location

It is important to note is that neither during download nor during correlation, the scripts filter for location! So whatever is between the two points separating station identifier and channel identified will simply be ignored.
This is because there is no well-fixed convention on what locations are called. For example, a station might have an STS-1 on location – and an STS-2 on 00. Also, a station might or might not have a location designated as –. As it would be way too tedious to look this up for every single station, the download script simply downloads the specified channel from all available locations. If you want to exclude certain locations, just remove the raw data.

2.5 Downloading RESP files

SEED Resp files containing the instrument response parameters for all times when that instrument was recording will be downloaded to `DATA/resp/RESP.NET.STA.LOC.CHA`. You can use `python download_resp.py INPUT/input_download.xml` or request the resp files from IRIS via breq_fast.

Continuing the example, we now download data:

1. Compile station id list. Let's say we want to download data from stations G.COYC, G.CAN, II.BFO and AF.WIN and we want all three LH channels (which typically have a sampling rate of 1 Hz). So the id list would look like:

   ```
   G.CAN..LH?

   ...
   ```

   You find a similar text file under

   ```
   INPUT/donwloadlist.txt
   ```

2. Edit the xml input file. You will find an example under

   ```
   INPUT/input_download.xml
   ```

3. To download, type

   ```
   cd ANT/

   mpirun -np 2 python par_download.py
   ```

4. After some patient waiting, the script finishes and you can check your raw data directory. It should now contain data files similar to:

   ```
   ls DATA/raw/latest/

   G.CAN..LH?.2012.001.00.00.00.2012.032.00.00.00.mseed

   G.COYC..LH?.2012.001.00.00.00.2012.032.00.00.00.mseed

   II.BFO..LH?.2012.001.00.00.00.2012.032.00.00.00.mseed
   ```

   Additionally, the report file `downloadreport.txt` informs you that download for AF.WIN did not work.

5. Move the data files to the directory previously created:

```
mv * DATA/raw/noisy/
```

Reasons why a selection did not return data can be the following

- the station wasn't yet operational at the period you requested
- the station was recording at that time, but recordings aren't archived at IRIS
- the station was recording, but the channel you are requesting was not sampled
- as previously describedm in some cases none of the above reasons apply but no data is downloaded. Try a breq_fast request.

2.6 Downloading stationxml files

Station xml files (https://www.fdsn.org/xml/station/) are just another Metadata container, and in the case of ANTS are used for obtaining station coordinates. They can be downloaded conveniently with the obspy.clients.fdsn client if available. If they aren't available there, try to obtain them from your data provider.

A convenience script is available to download station xml files. In the ANTS directiry, try running:

```
python download_staxml.py <path-to-downloadlist>
```

where the downloadlist is just a .txt file containing station names in the pattern NET.STA. (You can reuse the downloadlist.txt from downloading time series data.)

3 Preprocessing

Preprocessing input has to be specified in a python input file `INPUT/input_correction.py`. Preprocessing contains the steps: Filling in short gaps; windowing; taper/mean removal; antialias filter and decimation; instrument correction. All fields in the input file have to be filled in.

Before starting a processing run, quickly review whether

- All the channels you target have been successfully downloaded? LHZ, LHZ, LHN, BHE,...?
- Resp files have been downloaded? There must be one resp file per channel! Otherwise, instrument correction will not work (see above for resp files.)

An example input file for processing is provided in

`INPUT/input_correction.py`

Review this file and run

```
mpirun -np 2 python ant_proc.py
```

which will cut data into 131072 second windows, decimate them to 1 Hz sampling rate, and correct them to ground velocity. The resulting data should be found in `DATA/processed/noisy/`.

After preprocessing, the resulting output can be listed with the script `UTIL/listproc.py`. This helps to identify large gaps. It is invoked as

```
python UTIL/listproc.py < directory to check > < name for output file > < prepname >
```

...where the 'prepname' is the name tag of the preprocessing run.

4 Correlation

Correlation is the central piece of this tool. Two correlations are available: 'Classical' cross-correlation and phase cross-correlation [1, 2]. They can be obtained seperately or in one run. For the classical cross-correlation, you can choose to normalize (and get a proper correlation) or not to normalize (and keep a covariance; trace energy will be saved in the sac header for later optional use). The steps of the correlation routine are:

- Compiling a list of possible correlations between all stations listed in an ascii input file. The user has to specify this station list file, choose the channels, choose the component (vertical or horizontal) and indicate whether or not to calculate autocorrelations.
  The user also has to specify the number of correlations that are calculated with one round of reading trace data ('npairs'). If npairs is 3, a core loads data to calculate 3 correlations. The more npairs are assigned, the more memory each core needs.
  If 'update' is set to 1, the tool checks whether each correlation is already saved and correlates only those that are not.
- After the code has found the possible correlations, each rank is assigned a bunch of them. From then on the cores work independently.
- Data is loaded and cut into windows of length specified with the parameter winlen. The windows overlap by the nr of seconds specified in olap.
- After cutting into windows, data is antialias-filtered and downsampled, if the sampling rate specified in Fs is below the original sampling rate.
- Before correlation, the data are bandpass filtered if `apply_bandpass` is set to `True`.
- In addition, a small choice of pre-processing such as spectral whitening and running mean normalization in time domain are provided.

Before running, an ascii file with a list of stations to be correlated has to be provided. The path to this file has to be entered in the input file as `idlist` and can either be created manually (just type NETWORK.STATION.LOCATION. for the files you'd like to include in the correlation) or using a little script called

`UTIL/prepare_corr_ids`

(this script will list the names of stations from which data are available in a specified directory). An example correlation list file is already provided at

`INPUT/correlationlist.txt`

An example input file to set up the correlation run itself is provided in

`INPUT/input_correlation.py`

This can be run by invoking

```
mpirun -np 2 python ant_corr.py
```

The routine will find a list of possible correlations that the two cores will then independently calculate in windows, and stack. They should be saved as SAC files in the folder

`DATA/correlations/noisy/`

if everything goes well. You can look at the data e.g. with ipython:

```
ipython

In[1]: from obspy import read

In[2]: tr = read('DATA/correlations/noisy/*.SAC')

In[3]: tr.plot()
```

...and maybe you can see a signal.

5 Getting squared amplitude ratios

Once we have obtained a set of correlations, we can take different measurements on them. One simple but informative measurement is the ratio of signal energy of a window on the causal as compared to the same window on the acausal branch. Conceptually, the difference in amplitude between these windows is due to the anisotropic distribution of sources.
To obtain this measurement, we go through two steps: 1) Selection of the appropriate propagation velocity and window length and 2) Running the measurement.
If you need to get a rough estimate of the propagation velocity, and have no idea yet, you can plot a record section using the script UTIL/plot_section.py. You can then manually fit the desired velocity by trying out different values for the r_speed parameter.

Suppose you'd like to fit a propagation speed to your correlations, examples for which are stored in sac-format in folder DATA/examples/. You can get a rough estimate via

```
ipython

In[1]: import UTIL.plot_section as ps

In[2]: ps.plot_section('DATA/correlations/noisy/*.SAC',ps_nu=0, annotate=True,/
...prefilter=(0.005,0.02,3),dist_scaling=1e6, scaling=True,r_speed=3700,/
...verbose=True, savepng=True, savesvg=False,maxlag=12000)
```

This will plot a section of your data located at DATA/correlations/mydata, with the distances between the stations on the y axis and scaled by the value dist_scaling (just play around with different values until you see your signal at a convenient amplitude). The parameter r_speed controls the slope of the gray dotted line, which you can then fit to match the center of the signal window you'd like to select. The remaining parameters are:

- **ps_nu**: if set to any value greater than 0, will be used as an exponent for the phase weighted stack. You can test ps_nu=2 and will see that it suppresses uncorrelated noise.
- **annotate** switches annotations with the station names on and off
- **prefilter** applies a Butterworth zerophase bandpassfilter (lower corner, upper corner, order)
- **scaling** controls whether scaling with distance is switched on or off altogether
- **verbose** repeats the names of the files it is trying to plot
- **maxlag** limits the x-axis
- Plots are saved if either **savepng** or **savesvg** are set to True

Whether you try to fit the propagation velocity to the front of the wavegroup and then choose only a window after that, or fit it to the middle and then choose the window on either side, is not terribly important, as long as you hit the window you would like to select.

If you already know what the propagation velocity should be, for example from previous studies or a regional velocity model, you can go directly ahead and choose a window width and review it with the following script.

Once a propagation speed is estimated, we apply it to choose windows and take a measurement in them. For the example above, a very low-frequency band has been selected, so if you'd like to include for example 3 times the longest period on either side of the estimated arrival time, you'll choose windows of 600 seconds length on either side. All the input for the measurement is specified in `INPUT/input_measurement.py`.
To measure, run

```
python measr_asym.py measure
```

This will work on all files matching the pattern 'DATA/correlations/noisy/*ccc.noisy.SAC' (or whatever you have specified as input) and store the resulting measurement in two files called noisy_asymmetry.msr1.txt and noisy_asymmetry.msr2.txt.
Since parameter **doplot** is set to True, graphs showing the traces will be plotted, with the selected signal and noise windows, and some information on the measurement. In this way you can manually adapt the windows to your data and purposes. The following parameters are:

- **window**: You can select Hann or Boxcar windows.
- **g_speed**: the propagation speed at which the signal is expected
- **w1, w2**: Length of window in seconds before/after the arrival determined by g_speed
- **prefilter** applies a Butterworth zerophase bandpassfilter (lower corner, upper corner, order)
- **ps_nu**: if set to any value greater than 0, will be used as an exponent for the phase weighted stack.In the prepared example, only the linear stack is available, so this must be 0.
- **verbose** repeats the names of the file it is trying to take a measurement on

Bibliography

[1] Martin Schimmel. Phase cross-correlations: Design, comparisons, and applications. *Bulletin of the Seismological Society of America*, 89(5):1366–1378, 1999.

[2] Martin Schimmel, Eleonore Stutzmann, and J Gallart. Using instantaneous phase coherence for signal extraction from ambient noise data at a local to a global scale. *Geophysical Journal International*, 184(1):494–506, 2011.