Guide to running Generalised Phase Detection (GPD) algorithm

This guide is intended to illustrate the steps to run the GPD algorithm for phase detection on 3 component data. For more details on the technical aspects, please refer to the paper: (https://pubs.geoscienceworld.org/ssa/bssa/article/108/5A/2894/546740/Generalized-Seismic-Phase-Detection-with-Deep)The algorithm can be pulled from the github: https://github.com/interseismic/generalized-phase-detection

Input files: .json file

A json file is required, which specifies settings for the detection. The main parameters are:

"min_proba": 0.5: This parameter specifies how much confidence the user would like on the picks (between 0 – 1.0), the higher this number, the fewer detections will be made, but the confidence will be higher. Generally, 0.5 is reasonable

"freq_min": 1.0: This should be kept as is, since the training data were filtered in this range.

"freq_max": 20.0: This should be kept as is

"filter_data": true: This should be kept

"decimate_data": true: This should be kept

"model_file_S": "model_S_006_0.692474.pt": This model file should be given

"model_file_P": "model_P_022_0.691281.pt": This model file should be given

"batch_size": 4096: The number of waveform chunks which are moved to the GPU/CPU at once. Reduce this number if encountering "out of memory" type of issues.

"half_dur": 8.0: Should be kept as is

"delta":0.01: Should be kept as is

Input file: Waveform data

The raw data input file for mseed mode should specify the locations of the raw waveforms. Each line is 1 station for 1 day, each line consists of the path to the N, E, Z components in that order, e.g.:

```
/…/2018/139/IFDF_HHN.mseed /…/2018/139/IFDF_HHE.mseed
/…/2018/139/IFDF_HHZ.mseed

/…/2018/139/IFSC_EHN.mseed /…/2018/139/IFSC_EHE.mseed
/…/2018/139/IFSC_EHZ.mseed
```

The code is run as:

```
python gpd_run.py gpd.json [-P] -D cuda:0 input.in output.out
```

`[-P]` is optional to plot the picks, do not include for general use. The output file will be a list of picks

Guide to running Phaselink

This guide is intended to illustrate the steps to run the code from start to finish. For details on the mechanics of the engine, please refer to the paper (https://doi.org/10.1029/2018JB016674). Guide is intended for code pulled from github: https://github.com/interseismic/PhaseLink

Step 0 – Setting up 1D velocity model

Note: This version is for the "original" phaselink version

Objective: Phaselink requires a travel time lookup table to function, both to generate the initial training catalogue, and to run back-projection to ensure pick quality. The integrated travel time generator is contained in the `raytracer.tar.gz` file.

To generate the velocity model, edit the `hkun07.sealevel.txt` file to match the desired 1D velocity model, making sure the spacing is fine enough to resolve the desired earthquake depths and distances, and then run with `python make_TTtable.py` . The output files are `TT.*.pg` and `TT.*.sg` for P and S travel time lookup tables respectively. The generated `TT.*.pg` and `TT.*.sg` file paths will need to be inputted into the `phaselink.json` file.

Relevant parameters:

```
"tt_table": {
        "P": "TT/TT.hkun07.elev2km.pg",
        "S": "TT/TT.hkun07.elev2km.sg"
    },
```

Potential Error: Sometimes the fortran routine is not able to read the input velocity tex tfile properly, many times this is due to the text encoding of the velocity files. The easiest way to resolve this issue in windows is to copy into notepad, and save as ANSI.

Run as: `python make_TTtable.py`

Outputs: Travel time grids with format `*.pg and *.sg`

Step 1 – Generating training data

Objective: This code generates the training data for the neural network.

Relevant parameters:

"`t_win`": This controls the length of the time window of each training sample, and can generally be left at 120 seconds.

“**n_max_picks**”: Maximum number of picks in each training sample. This depends mostly on how many stations are in the network, and are expected to detect earthquake arrivals. I’ve used a rule of thumb of 4X number of stations as this parameter, e.g. if there are 32 stations, use 128 for **n_max_picks**.

“**ave_eve_sep**”: Mean time between events in the simulated catalogue. For 120 second window, 12 second separation seems to work well.

“**station_file**”: Linked file should contain the network, station name, latitude, longitude of each station

“**n_train_samp**”: Number of training samples. 100 000 is enough to initially test if the network is setup at all, 1 000 000 for most use cases, 10 000 000 for final catalogue, or if there are many stations.

"**lat_min**", “**lat_max**”, ”**long_min**”, ”**long_max**”, “**z_min**”, “**z_max**”: Edges of the volume in which to simulate earthquakes. Use the furthest extent at which earthquakes are expected.

“**n_fake**”: Number of fake picks to generate per training sample. “**n_max_picks**”/2 seems to work well, but may try “**n_max_picks**”/1 or “**n_max_picks**”/4 if network is not working well.

“**max_event_depth**”: Set to same as “**z_max**”

"**min_hypo_dist**": The code drops some source-receiver ray paths to simulate scenario where not all picks are made. This parameter is the minimum distance at which ray paths may be dropped. This evidently depends on many factors, rule of thumb may be 25% of the diagonal dimension of the simulated volume. I use 15km here for the Dallas-Fort Worth basin

"**max_hypo_dist**": Similar to “**min_hypo_dist**”, but the maximum distance, i.e. ray paths beyond this distance are definitely dropped. Maybe 50% of the diagonal dimension? I use 40 km for the Dallas-Fort Worth basin.

"**max_pick_error**": Maximum error that is added to the simulated phase arrival time. Depends on confidence on the picks. I use 0.5 seconds here, may be increased for larger regions, or if velocity structure is more uncertain.

"**training_dset_X**": File name for X training data that will be created, file extension .npy

"**training_dset_Y**": File name for Y training data that will be created, file extension .npy

Run as: **python phaselink_dataset.py params.json**

Outputs: **training_dset_X, training_dset_Y** which contain the X and Y training data for the neural network.

Step 2 – Training the network:

Objective: This step takes the training dataset created in step 1, and trains the neural network weights in epochs. Outputs a set of model weights with `.pt` extension for each epoch.

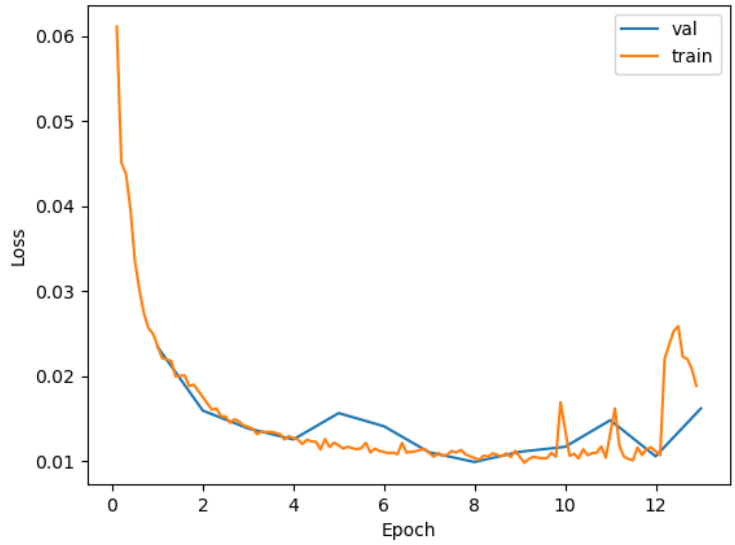
Step 2.1: Trained network models are saved to `phaselink_model` directory. Please create one prior to running.

Relevant parameters:

"`device`": Device that will be used, e.g. “cuda:0”

“`model_file`”: Once the network is trained, choose the model with lowest validation loss, e.g. "phaselink _model/model_007_**0.008235**.pt" where the validation loss is bolded.

If using edited version, will also output loss curves which are useful to diagnose training process. A good loss curve looks like the following, where validation and training loss are similar. In this case, the best model would be around epoch 8.



Run as: `python phaselink_train.py params.json`

Outputs: trained model file in `phaselink_model` directory

Step 3 – Using the trained network to identify earthquakes

Objective: Applies trained neural network model to a set of phase arrival picks, and generates clusters corresponding to earthquake events.

Relevant parameters:

"**gpd_file**": Input text file where each line corresponds to a pick with "NETWORK STATION PHASE UTCDATETIME PICK_PROBABILITY PICK_WIDTH". This is created from running the GPD code.

"**outfile**": File name for the output file, which follows "n_min_det" condition and can be read into NonLinLoc

"**outfile_minPS**": File name for the output file, which follows "**n_min_det_P**" and "**n_min_det_S**" condition and can be read into NonLinLoc

"**n_min_nucl**": Minimum number of picks to initially nucleate a cluster. Rule of thumb might be 2/3 of **n_min_det**

"**n_min_merge**": Merge clusters if they share at least this number of picks in common. **n_min_merge = 1 or 2** works here.

"**n_min_det**": Minimum number of picks in a cluster after back-projection. 12 will generally yield good results, however if the network is small a lower number should be used.

"**n_min_det_P**": Same as above, but specifying specific minimum number of P picks for a cluster

"**n_min_det_S**": Same as above, but for S picks.

"**batch_size**": Number of samples sent to the GPU at one time, depends on GPU VRAM. If you are throwing GPU memory errors, use half this number. Repeat if necessary.

"**pr_min**": Minimum GPD probability, usually 0.5. Set to 0 if you want to use all picks.

"**trig_dur_min**": Keep at 0

"**n_x_nodes**", "**n_y_nodes**", "**n_z_nodes**": Number of nodes for back-project. Generally 1km spacing (number of nodes = **lat_max** – **lat_min** in km) is good, unless back-project is too slow. **These grid nodes do not need to be the same as in step 0!** They can be significantly subsampled, 500 m or so is fine.

"**t_repick**": Time window outside of first and last pick in which non-nucleated picks are considered. 10 seconds is probably ok here.

"**min_sep**": Maximum error between predicted and actual travel time. 0.5 to 1 second is ok here, may need to be higher if large distance or velocity structure is uncertain.

"**max_pick_dist**": Maximum distance at which picks are considered. Diagonal dimension of search volume may be used here. E.g. 50 km

"**min_pick_dist**": Ray paths less than this distance are not considered. Helps to remove events corresponding to noise sources at surface. E.g. 0.5 km

" **prebuilt_TT** ": True if using pre-built hdf5 files for the velocity mode, false otherwise the csv velocity model is required regardless.

"**prebuilt_TT_hdf5": "Decatur_TT.hdf5**" : hdf5 containing pre-built velocity model, should contain sub-files in the format:

```
TT_123.hdf5:
                S_TT (4D float array: X,Y,Z,station ID)
                P_TT (4D float array: X,Y,Z,station ID)
                STA_NAME (1D string array: station names)
                GRIDLOCS (3D float array: co-ordinates of X,Y,Z
grid cells)
```

Run as: **python phaselink_eval.py params.json**

Outputs: **outfile** containing clusters of picks corresponding to earthquake events separated by extra newlines . **outfile_params** also outputs params file back, using the same name convention as the outfile name.

Step 4 – Run NonLinLoc

Only needed if running original code, not needed if using the "ImplicitDepth" or "ExplicitDepth" versions

Step 5 – QC of events by visual inspection

Objective: Takes each event after it has been located through Nonlincloc, and then plots all the waveforms for each event sequentially. Used to verify that the clusters correctly correspond to earthquakes. Generally <1% false positive rate is desired.

Relevant parameters:

"**nlloc_loc_path**": Directory that contains Nonlinloc run files, e.g. "/usr/NLLruns/2005_6_2_9_minsep10"

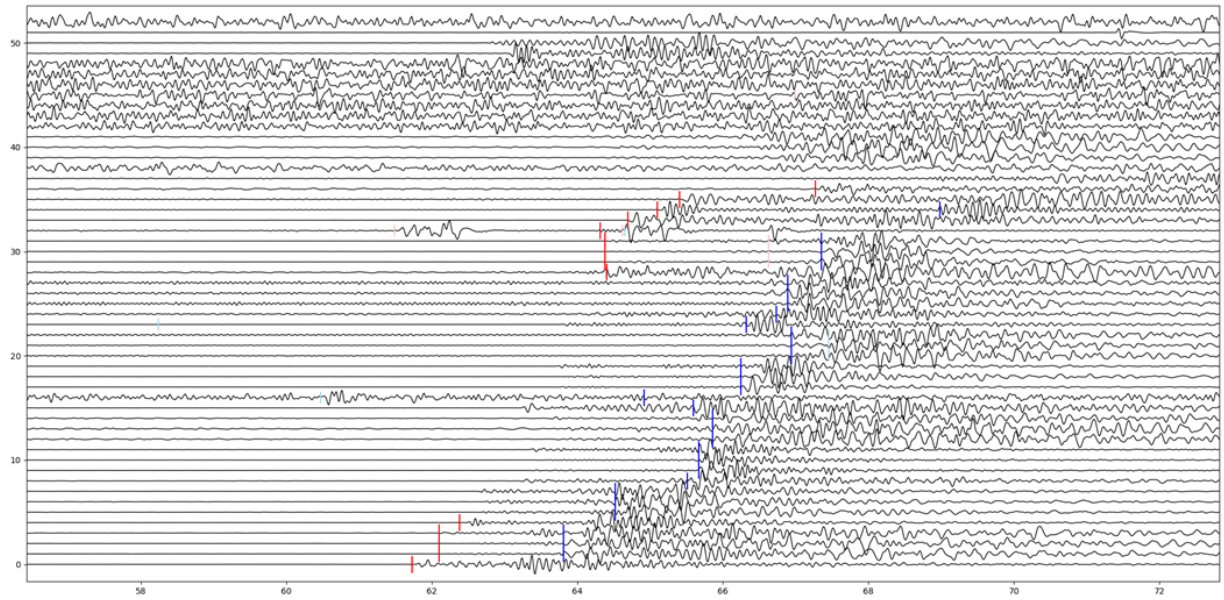
"**nlloc_sum_file**": File header name for summary Nonlinloc run file, e.g. "Phaselink_NLL_.sum.grid0.loc.hyp"

"**wf_path**": Directory containing all waveforms mseed files. Waveforms within should be in format of **/year/julian_day/waveforms.mseed**, e.g."/usr/DFW/Waveforms",

"**plot_sorted**": Set true if you want to view stations by alphabetical order, false to view by arrival time.

Run as: **python phaselink_plot.py params.json**

Output: Displays plots of each event, evaluate to determine whether picks are as expected. Example:



Appendix: Packages and modules required

Load module anaconda/5.2.0-3.6 to activate anaconda environment, then load module powerai/1.5.4 to run codes

Specific anaconda environment used (clone to use) is found at:
/data/gpfs/Users/l0526615/.conda/envs/obspy2

The following packages and versions are used in running phaselink

```
apex                      0.1                      pypi_0    pypi
cudatoolkit               10.1.243              h6bb024c_0
cupy-cuda101              7.2.0                    pypi_0    pypi
cython                    0.29.21                  pypi_0    pypi
geopy                     1.20.0                     py_0    conda-forge
h5py                      2.10.0          nompi_py37h513d04c_100    conda-
forge
hdf5                      1.10.5          nompi_h3c11f04_1104    conda-forge
matplotlib                3.1.1                    py37_1    conda-forge
```

```
matplotlib-base           3.1.1            py37he7580a8_1    conda-forge
mpi                       1.0                       mpich    conda-forge
mpi4py                    3.0.3            py37hcf07815_0    conda-forge
mpich                     3.3.2                hc856adb_0    conda-forge
numba                     0.48.0           py37hb3f55d8_0    conda-forge
numpy                     1.15.4          py37h8b7e671_1002    conda-forge
obspy                     1.1.1            py37h3010b51_1    conda-forge
pandas                    0.25.1           py37hb3f55d8_0    conda-forge
pyproj                    1.9.5.1         py37h2944ce7_1006    conda-forge
pytorch                   1.4.0           py3.7_cuda10.1.243_cudnn7.6.3_0
pyt
scikit-learn              0.21.3           py37hcdab131_0    conda-forge
scipy                     1.5.1            py37ha3d9a3c_0    conda-forge
torchaudio                0.4.0                      py38    pytorch
torchvision               0.5.0                 py38_cu101    pytorch
```