Manual for the NCCPy Package

1. Overview

This NCCPy package is an earthquake relocation program that aims to improve the resolution of earthquake locations. It first utilizes waveform cross-correlation with grid search approaches to determine relative locations between pairs of earthquake hypocenters (their initiation points) and centroids (their center of masses), then pin down final locations of all earthquakes via inversions. Using small events as anchors by assuming that their hypocenters and centroids are close together, we could obtain the relative locations between hypocenters and centroids for all events.

The package is rewritten and modified from the *hyponcc* package, originally a centroid relocation package, and is written in Fortran by Dr. Satoshi Ide and Dr. Kazuaki Ohta.

For detailed reference of the algorithms used in this package, including detailed formulations, implementations, and robustness tests, please refer to:

Chang, T.-W., & Ide, S. (2021). Hypocenter hotspots illuminated using a new cross-correlation-based hypocenter and centroid relocation method. *Journal of Geophysical Research: Solid Earth*, *126*, e2021JB021991. https://doi.org/10.1029/2021JB021991

Other studies focusing on the usages of the original *hyponcc* package:

Ohta, K. & Ide, S. (2008). A precise hypocenter determination method using network correlation coefficients and its application to deep low-frequency earthquakes. *Earth Planets Space*, *60*, 877–882. https://doi.org/10.1186/BF03352840

Ohta, K. & Ide, S. (2011). Precise hypocenter distribution of deep low-frequency earthquakes and its relationship to the local geometry of the subducting plate in the Nankai subduction zone, Japan. *Journal of Geophysical Research: Solid Earth*, *116*, B01308. https://doi.org/10.1029/2010JB007857

Chang, T.-W., & Ide, S. (2019). Empirical relocation of large subduction-zone earthquakes via the teleseismic network correlation coefficient method. *Earth Planets Space*, *71*, 79. https://doi.org/10.1186/s40623-019-1057-z

Version 0.6.0 (27 Jan, 2022)

Author: Ta-Wei Chang

Reachable at: twchangsci@gmail.com

2. Usages

2.1. After the download:

- To be able to run the package from any desired directories, one will need to append the path to which the package is stored onto $PATH after downloading it. To do this, (for Unix and Linux users,) add:

  ```
  export PATH="$PATH:(path to package)"
  ```

  in your starting script of your Shell (e.g. ~/.bash_profile).

- For simple calling and usage of the package, one is advised to modify **the first line of** `NccPy_main.py` to the path of your Python. (If you do not know, try inputting "which python" in Terminal.)

- For Mac users, to avoid from bugs associated with downloaded programs, one is advised to go to the *Source* directory, and perform this to `NccPy_main.py`:

  ```
  xattr -d com.apple.quarantine NccPy_main.py
  ```

2.2. Files/ data required:

- **Original catalog** to start the relocation from
  - ➡ In the format of:
    YYYY MM DD hh mm ss lat lon dep mag

- **Seismogram data** in SAC format (event data with information such as origin time given)
  - ➡ Require both velocity (for centroid) and acceleration (for hypocenter) data
  - ➡ Path will be: ~/Data/(Acc/Vel)/(YYYYMMDDhhmmss)/(SAC files),
    where the time in the directory names exactly following the original catalog, with "ss" being rounded-down second. Do keep the directory names as "Acc" and "Vel".
  - ➡ Some preliminary sampling frequency screening is recommended (the programs expect the sampling frequencies of all files to match that of the first SAC file in the directory, and will remove all data with different sampling frequencies)

- **Parameter file** (detailed later on):
  - ➡ Comes in two formats: simplified and complete
  - ➡ The simplified: for quick-and-dirty runs; auto-generates during the initializing mode:

    ```
    NccPy_main.py
    ```

  - ➡ The complete: for customization of parameters; will be auto-generated during actual quick-and-dirty runs, or could be outputted upon request (Sec. 2.3.3)

2.3. To run the package:

Just type:

```
NccPy_main.py
```

All the instructions will be displayed on the screen, with a template of parameter input file prepared for you.

2.3.1. Running the Demo:

To demonstrate how the package could be executed, a demo case with all necessities is prepared in the *Demo* directory.

Before executing it, please take a moment to browse through the three (only) necessities: original catalog, waveform data, and (simplified) parameter file. These will be the minimum required input when applying to your own datasets as well. The parameter file is prepared here for demo purposes; in actual runs, templates could be auto-generated for your convenience, as detailed in Sec. 2.3.2.

To execute, simply navigate to the *Demo* directory and enter this in your Terminal or equivalent:

```
NccPy_main.py -run_all para_input_light.txt
```

Note on the data included: the waveform data are originally obtained from Hi-net, which is maintained by National Research Institute for Earth Science and Disaster Resilience (NIED) (https://doi.org/10.17598/NIED.0003). They are processed into SAC format and to include arrival time information given by Japan Meteorological Agency (JMA) (https://www.data.jma.go.jp/svd/eqev/data/bulletin/deck_e.html). The earthquake hypocenter catalog is obtained from the Seismological Bulletin of Japan published by JMA (https://www.data.jma.go.jp/svd/eqev/data/bulletin/hypo_e.html).

2.3.2. Quick-and-dirty run with your own dataset

As simple as with the demo case in Sec. 2.3.1, with the catalog and the seismogram data ready, running the package takes only 3 simple steps (after navigating to your working directory):

I. Use the initializing mode: tells you all you need on the splash screen + outputs a for-you-to-edit simplified parameter file (called "para_input_light.txt")

```
NccPy_main.py
```

II. Edit the simplified parameter file accordingly:

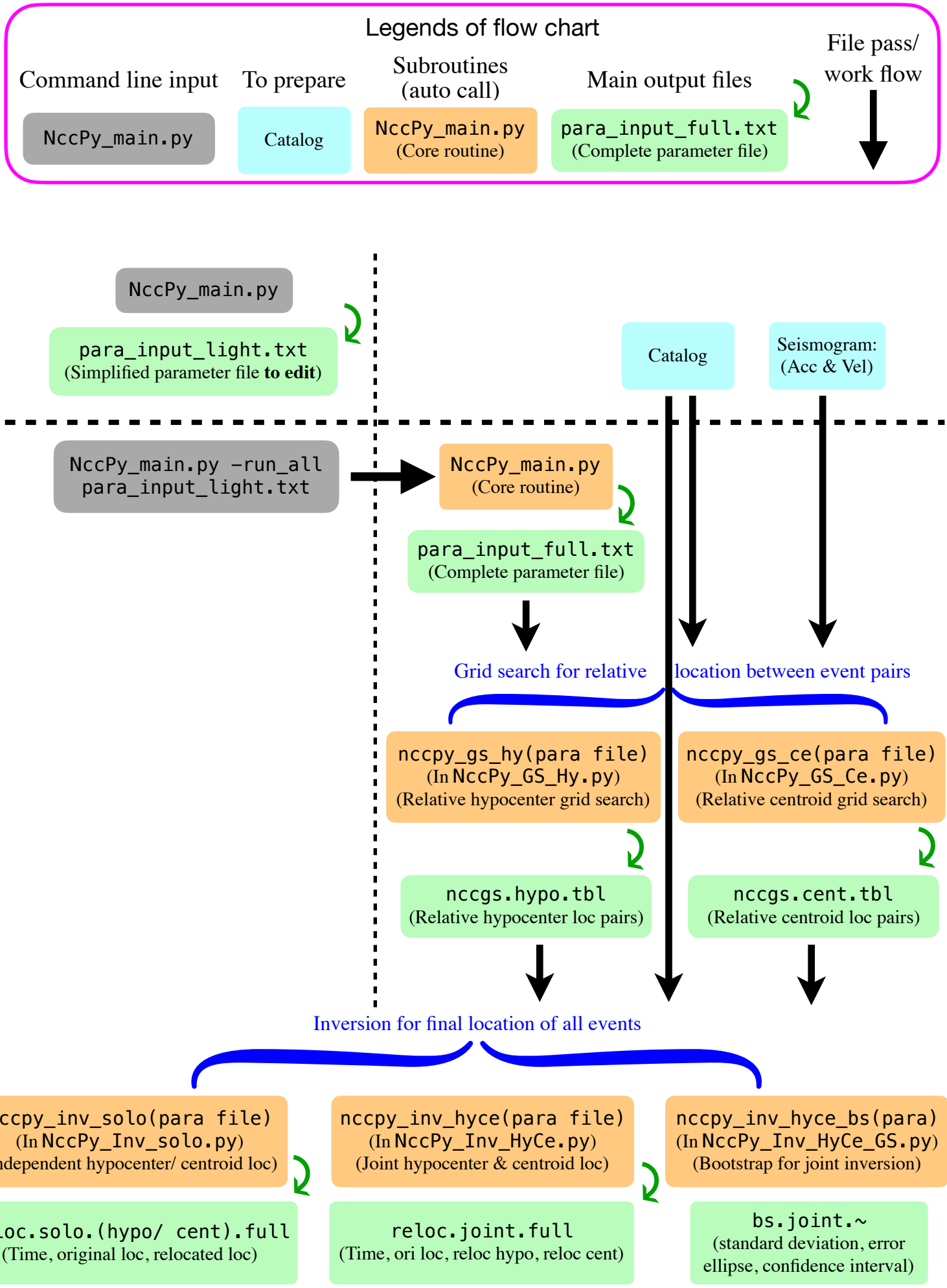
Line 1: path to original catalog
Line 2: path to waveform data files (the parent directory of Vel / Acc)
Line 3: automatically set all parameters? (0: no; 1: yes) (setting it as 0 will naturally require the complete parameter file, which we skip for now)

III. Run the whole package

```
NccPy_main.py -run_all para_input_light.txt
```

That's it! Be patient, and wait for your results!

One point to note, however, is that the auto-filling of parameters is currently optimized for near-field (< 10°, ≤ 100 km) applications on up-to moderate-sized (M ≤ ~5.5) events. For other applications, tuning of parameters is strongly recommended. Also, for event-station distances > 10° and/ or depths > 100 km, a re-calculation of travel-time table is required.


Legends of flow chart
Command line input
To prepare
Subroutines (auto call)
Main output files
File pass/ work flow
NccPy_main.py
Catalog
NccPy_main.py (Core routine)
para_input_full.txt (Complete parameter file)
NccPy_main.py
para_input_light.txt (Simplified parameter file to edit)
Catalog
Seismogram: (Acc & Vel)
NccPy_main.py -run_all para_input_light.txt
NccPy_main.py (Core routine)
para_input_full.txt (Complete parameter file)
Grid search for relative
location between event pairs
nccpy_gs_hy(para file) (In NccPy_GS_Hy.py) (Relative hypocenter grid search)
nccpy_gs_ce(para file) (In NccPy_GS_Ce.py) (Relative centroid grid search)
nccgs.hypo.tbl (Relative hypocenter loc pairs)
nccgs.cent.tbl (Relative centroid loc pairs)
Inversion for final location of all events
nccpy_inv_solo(para file) (In NccPy_Inv_solo.py) (Independent hypocenter/ centroid loc)
nccpy_inv_hyce(para file) (In NccPy_Inv_HyCe.py) (Joint hypocenter & centroid loc)
nccpy_inv_hyce_bs(para) (In NccPy_Inv_HyCe_GS.py) (Bootstrap for joint inversion)
reloc.solo.(hypo/ cent).full (Time, original loc, relocated loc)
reloc.joint.full (Time, ori loc, reloc hypo, reloc cent)
bs.joint.~ (standard deviation, error ellipse, confidence interval)

2.3.3. For more advanced users:

It is recommended for most users to run through the quick-and-dirty procedures once before tuning the parameters. Among others, one reason is that by doing so, the complete parameter file to be used by the subroutines will be auto-generated and implemented. For more advanced users seeking customization of the parameters, however, the following customization are also prepared for your convenience:

❖ To obtain a complete parameter file for customization:

```
NccPy_main.py -para_custom
```

(Optional, but recommended) If catalog file is available, some parameters, such as the grid dimensions, could be auto-filled:

```
NccPy_main.py -para_custom (catalog file)
```

The complete parameter file named "para_input_full.txt" will be generated.

The usage of this customized complete parameter file is the same as that of the simplified parameter file:

```
NccPy_main.py -run_all para_input_full.txt
```

(Note: the parameter file name is arbitrary, and hence it is necessary to specify which one to use when running the package)

❖ To run only some of the subroutines:

First, to show the subroutine list of the package:

```
NccPy_main.py -run_single
```

To run any subroutine of interest, type:

```
NccPy_main.py -run_single (subroutine) (parameter file)
```

Note that the complete parameter file is required in this mode.

2.4. Contents of the package:

In brief, this package contains:

- ❖ Wrapper of entire package
  - ◗ `NccPy_main.py`: for simple calling and usage of the whole package

- ❖ Core calculation I: grid searches for relative locations between event pairs
  - ◗ `NccPy_GS_Ce.py`: relative centroid locations between event pairs
  - ◗ `NccPy_GS_Hy.py`: relative hypocenter locations between event pairs

- ❖ Core calculation II: inversion for final locations of all events
  - ◗ `NccPy_Inv_solo.py`: determine hypocenter and centroid locations separately
  - ◗ `NccPy_Inv_HyCe.py`: jointly determine hypocenter and centroid locations
  - ◗ `NccPy_Inv_HyCe_BS.py`: bootstrap for the joint inversion results for uncertainties

- ❖ Visualization and confirmations
  - ◗ `NccPy_plot_map.py`: plot the original/ relocated locations on maps

- ❖ Supplementary:
  - ◗ `NccPy_trvtbl.py`: pre-calculate travel-time table for given velocity structure
  - ◗ `NccPy_main_aux.py`: auxiliary subroutine file for `NccPy_main.py`
  - ◗ `NccPy_GS_Ce_aux.py`: auxiliary subroutine file for `NccPy_GS_Ce.py`
  - ◗ `NccPy_GS_Hy_aux.py`: auxiliary subroutine file for `NccPy_GS_Hy.py`
  - ◗ `NccPy_Inv_aux.py`: auxiliary subroutine file for Core calculation II

2.5. Details of parameter file

2.5.1. The simplified parameter file (default to: para_input_light.txt)

```
Catalog.txt
../Data
1
```

where

```
Name of catalog
Path to data directory (parent directory of Vel/ Acc)
1 to automatically set all parameters; or 0 to not
```

A template could be auto-generated by the initializing mode:

```
NccPy_main.py
```

2.5.2. The complete parameter file (default to: para_input_full.txt)

```
Catalog.txt
../Data
1
../Travel_table/AK135_reg

6.00     6.00     6.00     1.00     0.20     0.20     0.20     0.01
2.00     2.00     2.00     0.50     0.10     0.10     0.10     0.01

1   0   1   1   10.0  2500.0
4.0    1.0    8     200   400.0    6.0     6.0
5.0   1.0   4.0   0.10

1   0   1   1.0   10.0
0.3    0.2    8     20    150.0    6.0     6.0
10.0  1.0   4.0   0.10    10.0
1.0   1.0   2.0   3.0   1.0   0.5   1.5   1.0   0.2   2.0

1   1   0.950  3.2   8     0.3   6.00     6.00
1   0   0.950  3.2   8     0.3   6.00     6.00
0.100000  0.000010  0.900000  15.0  0.010
5000    0

10.0
4.5   3.2   2.0
red           green         blue
```

The details of all corresponding parameters are:

```
<Catalog file name>
<Path to data directory>
0
<Path to Source>/Travel_table/AK135_reg

ala1  alo1  adp1  adt1  dla1  dlo1  ddp1  ddt1
ala2  alo2  adp2  adt2  dla2  dlo2  ddp2  ddt2

sta_wv_out_tag(cent)  nccmap_out_tag(cent)  iconv  strdrop  vrup
twin  thead  minwv  maxwv  maxstadist  maxdist[0]  maxdist[1]
noscritsn  pre_window_sn  post_window_sn  noscritmean

sta_wv_out_tag(hypo)  nccmap_out_tag(hypo)  extdwin  db_sep
twin  thead  minwv  maxwv  maxstadist  maxdist[0]  maxdist[1]
noscritsn  pre_window_sn  post_window_sn  noscritmean  preratio
uddur  iud1  ud1bg  ud1crit  iud2  ud2seg  ud2crit  iud3  ud3seg  ud3crit

hl_exception_tag(hy/hyce)  centcc_tag  centccthres  magsep  minwv(hy)  ddifmax(hy)  maxdist[0] maxdist[1]
hl_exception_tag(ce)       centcc_tag  centccthres  magsep  minwv(ce)  ddifmax(ce)  maxdist[0] maxdist[1]
pnc_in  pnch_in  pncl_in  smax  ds
bsnum  detail_output

strdrop
color_mag[0]  color_mag[1]  color_mag[2]
color_def[0]  color_def[1]  color_def[2]
```

For full description of all parameters, refer to the description in the auto-generated para_input_full.txt file:

```
NccPy_main.py -para_custom (catalog file)
```

3. Appendix: dependencies & environment building

The package is built solely in Python environment to minimize machine dependencies. To run it, one will need the following packages:

- Python (tested with 3.8 and 3.9)
- Obspy
- Numba
- Pandas
- Numpy
- Scipy
- GMT (GMT 6 or above)

3.1. For beginners: building environment from scratch

An environment in which the necessary components are available in order to get the package working is needed to run the package. This includes various necessary packages in/ for Python (in their right combinations of versions!), as well as some for plotting the results more elegantly than that could have been achieved from within Python itself. One thing to know before we begin is that the following instructions are written by one with (inevitably) some preferences of certain packages to use, and the readers are of course welcomed to choose any approaches that suit themselves. As this could easily generate bugs, the detailed procedures of a (not only!) working example is given here, utilizing package managing functions provided by Conda.

❖ Download Conda (Miniconda is recommended, so as to have an easily-controllable environment)

❖ Create an environment (follow the instructions on the Conda Cheat Sheet (Do a search for the latest link)): go to the terminal (or equivalent)

➡ [Initial update]:

```
conda update conda
```

➡ [Create an environment]:

```
conda create –name (env. name) python=3.9
```

➡ [Enter the environment] (follow the instructions displayed on the splash screen for the correct syntax for your system):

```
conda activate (env. name)
```

➡ [Prepend channel of conda-forge] (add and priories “conda-forge” as the channel to download the following packages (this step is here to prevent bugs):

```
conda config –prepend channels conda–forge
```

➡ [Install Obspy in the environment]:

```
conda install obspy
```

➡ [Install Numba in the environment]:

```
conda install numba
```

➡ [Install Pandas in the environment]:

```
conda install pandas
```

➡ [Verification] now scroll up and look: if there are any upgrades or downgrades, there might be conflict in the dependencies for these packages just installed. To make sure no problem exists, re-run the above lines until conda deems that all necessary packages are installed.

```
conda update (whichever)
```

➡ [Install GMT (for map plots)] follow the instructions on its webpage or using conda; may also be necessary to add it to the path of current shell.

4. Appendix: details of each subroutine

4.1. `NccPy_trvtbl.py`

❖ In this subroutine:

Calculates the travel-time table and save to destined directory. By doing so, we will only need to run the actual travel-time calculation once, and simply look it up in the actual implementation.

The package currently ships with a pre-calculated travel-time table using the AK135 velocity structure in distance ranges of ≤ 10° and ≤ 100 km (named *AK135_reg*). For applications outside of the range or using different velocity structures, please build a new travel-time table with the following parameters changed. When running, do not forget to use the custom complete parameter file, and modify the travel-time table to use.

❖ Inputs:

- Parameters to set:
  - *vtbdistini*: [distance] start; closest (degrees)
  - *vtbdistend*: [distance] end; furtherest (degrees)
  - *vtbdiststep*: [distance] step (degrees)
  - *vtbdepini*: [depth] start; shallowest (km)
  - *vtbdepend*: [depth] end; deepest (km)
  - *vtbdepstep*: [depth] step (km) (Please set as 10 km, 1 km, 0.1 km, etc, for it is used in rounding of catalog before grid searches and inversions)
  - *dirname*: Output directory name (default to "Travel_table")
  - *fnametag*: Output file name tag (e.g. 'AK135_reg' for the regional version with the AK135 velocity structure)

❖ Outputs:

- In directory ***{dirname}***:
  - ***{dirname}/{fnametag}.P.txt***: travel-time table for P first arrival
  - ***{dirname}/{fnametag}.S.txt***: travel-time table for S first arrival
  - ***{dirname}/{fnametag}.para.txt***: parameters for this travel-time table, to be read in when applied

4.2. `NccPy_GS_Ce.py` (accompanied by `NccPy_GS_Ce_aux.py`)

❖ In this subroutine:

This subroutine performs grid searches to look for the best relative centroid locations between all possible event pair combinations. This is performed by grid searching for the relative location that correspond to the highest summation of cross correlation coefficient across all stations and components within the network (or to maximize the "network correlation coefficient values", hence "NCC values").

To make the calculation time feasible, several packages, such as numba and pandas, are implemented to greatly (up to several orders) decrease the calculation time.

What it does is generally the following:

- Waveform screening: length of data, signal-to-noise ratio, etc
- Perform convolution of event waveforms to adjust for difference in source time function (STF) duration. For example, in a case where theoretical duration of STF for event 1 is 5 s and that for event 2 is 2 s: convolve waveforms of event 1 with triangle function with duration of 2 s; and waveforms of event 2 with that of 5 s, to adjust for the differences (assumes STFs are simple triangle functions; theoretical durations calculated with assumptions of circular, unilateral ruptures using given stress drop and rupture velocity).
- Calculates cross-correlation tables, then perform the grid searches for max NCC values. Grid searches are performed in 2-stages to save computing resources: a larger and coarser as the first, followed by a smaller and finer one.

❖ Outputs:

- ***nccgs.cent.tbl***: the concise relative location file, to be passed onto inversions
  - ➡ In the format of:

event_1, event_2, r_lat, r_lon, r_dep, r_dt, NCC/std, precision, (-), max NCC/# data, # data

- In directory ***Logs/GS_cent****:*
  - ***nccgs.cent.out***: additional results to accompany the above
  - ***nccgs.cent.log***: detailed log file
  - ***stainfo.~***: information of station for the number of waveform data (default: outputted)
  - ***wvindex.~***: information of waveform alignment prior to/ after the grid search to maximize NCC values (default: outputted)
  - ***nccmap.~***: the complete NCC value records (default: not outputted)

4.3. `NccPy_GS_Hy.py` (accompanied by `NccPy_GS_Hy_aux.py`)

❖ In this subroutine:

This subroutine is modified from `NccPy_GS_Ce.py` (the centroid version) to be able to determine the relative hypocenter locations with cross-correlation- and grid-searching-based approaches.

This seemingly very tricky "locating the best (onset) segment to cross correlate" is managed by including an additional layer of grid search that helps looking for the best window to cross correlate, which is performed after the relative timing of the two waveforms are already fixed under appropriate time shift according to the relative locations and timings from the two locations (reference event & point in grid) to the station. To accelerate, this grid search is done prior to the main grid search for location and timing as well (i.e. perform this extra layer of grid search in advance, and only record the timings and the max CC values, rather than recording the extra layer of grid search and perform the same "search" during the maingrid search for relative location and timing). The definition of CC is replaced by scaled CC to additionally measure similarity in amplitude in addition to shape in ordinary CCs (following Ide (2019)), under the assumptions that it is much more difficult to find a random pair with both shape and amplitude being the same, while easier if only shape similarities are taken into account.

In addition to and along with the above, there are several other changes made from the centroid version:

- Calculation of the NCC/ standard deviation: in centroid, the standard deviation is measured using the same grid search; here it is measured with an additional ("mirror") grid search using waveforms before event waveform arrival (without extra layer of search, as they are all supposed to be "background", with or without additional layer).
- Uses acceleration waveforms instead of velocity waveforms for clearer onsets.
- As the interest is now on the onsets, the corrections for rupture duration differences are naturally omitted.
- Instead, we require additional screenings of polarity, which is picked using an automatic picking algorithm that picks when 3 criterions are met: departure from mean in standard deviations, gradient of kurtosis, and the ratio of standard deviation between the short segment immediately before and after the picking. This automatic picking algorithm is not expected to work perfectly, as we implement it solely for waveform screening purposes.
- In addition, as the cross correlation is performed on the onset, the cross correlation is re-defined by the scaled cross-correlation in that the normalization is different during the calculations. As mentioned above, by doing so, one could

measure the similarity in both shape and magnitude, as compared to only the shape in traditional cross correlations. As the absolute values and not just the shape is valued, the waveform pairs with pre-signal time signal being very different (as a proxy for the whole waveform) are removed as additional threshold for S/N screening. (This point is taken from Ide (2019))

- Use only P-waves to retain clearer starts.

❖ Outputs:

- ***nccgs.hypo.tbl***: the concise relative location file, to be passed onto inversions
  - ➡ In the format of:

event_1, event_2, r_lat, r_lon, r_dep, r_dt, NCC/std (mirror), precision, (-), max NCC/# data, # data, NCC/std (original)

- In directory ***Logs/GS_hypo****:*
  - ***nccgs.hypo.out***: additional results to accompany the above
  - ***nccgs.hypo.log***: detailed log file
  - ***stainfo.~***: information of station for the number of waveform data (default: outputted)
  - ***wvindex.~***: information of waveform alignment prior to/ after the grid search to maximize NCC values & (default: outputted)
  - ***nccmap.~***: the complete NCC value records (default: not outputted)

4.4. `NccPy_Inv_solo.py` (accompanied by `NccPy_Inv_aux.py`)

❖ In this subroutine:

This subroutine uses the original catalog and the relative locations between event pairs given by the grid searches and determines the final locations for all events through inversions, with original catalog information as prior information via Akaike Bayesian Information Criterion (ABIC). This subroutine, `NccPy_Inv_solo.py`, determines the final hypocenter locations and final centroid locations independently.

The relative location pairs from the grid searches are screened before applied. The first is the precision threshold, whereby events with relative locations from event A to B being reasonably similar with that of from B to A. An exception for this precision threshold is available if the sum of NCC / standard deviation between from A to B and from B to A are very different (retain the one with much larger sum of NCC / standard deviation).

The second is especially for hypocenters: for repeating earthquakes, the longer segments of the waveforms, in addition to only the onsets, could often be very similar between events. As this violates our assumption when adopting the scaled cross correlation that the only segments between the two event waveforms with almost the same shape and amplitude are the onsets, we remove all relative hypocenter locations that belong to larger repeating earthquakes to avoid such issues (i.e., the "relative hypocenter locations" may actually stand for relative locations of later part of ruptures, such as the centroid). Smaller repeating earthquakes are allowed, as smaller events possess systematic smaller hypocenter and centroid separations. Judgements of repeating earthquakes are made using sum of NCC values/ number of stations obtained from the relative centroid location grid searches.

❖ Outputs:

- ***reloc.solo.(hypo/cent).full***: master output file: time (original catalog), original catalog location, relocated hypocenter or centroid locations
- ***reloc.solo.(hypo/cent).catalog***: [for clean output] time (original catalog), relocated hypocenter or centroid locations
- ***reloc.solo.(hypo/cent).equsage***: Number of relative locations used to determine final location for each event
- In folder ***Logs/Inv_(hypo/cent)***: additional detailed log files

4.5. `NccPy_Inv_HyCe.py` (accompanied by `NccPy_Inv_aux.py`)

❖ In this subroutine:

This subroutine performs the same calculations as in `NccPy_Inv_solo.py`, but instead of determining hypocenter and centroid locations separately, it determines them jointly. This is achieved by using smaller events as anchors: assuming that smaller events have systematic smaller hypocenter—centroid separations, we "merge" the hypocenters and centroids of small events into "hypocentroids", thus reducing the unknown locations to be solved for from 2 to 1. With this, we link hypocenters and centroids of all events with smaller events, and thus obtaining relative locations among hypocenters and centroids of all events that are fully overlap-able and relative to each other.

❖ Outputs:

- ***reloc.joint.full***: master output file: time (original catalog), original catalog location, relocated hypocenter locations, and relocated centroid locations
- ***reloc.joint.(hypo/cent).catalog***: [for clean output] time (original catalog), relocated hypocenter or centroid locations
- ***reloc.joint.equsage***: Number of relative locations used to determine final location for each event (column 2: hypocenter; column 3: centroid)
- In folder ***Logs/Inv_HyCe***: additional detailed log files

4.6. `NccPy_Inv_HyCe_BS.py` (accompanied by `NccPy_Inv_aux.py`)

❖ In this subroutine:

This subroutine performs bootstrap to obtain uncertainty information of the jointly-inverted results. Specifically, the same calculations as in `NccPy_Inv_HyCe.py` is performed for a given number of times, each time a subset (same number but resampled with duplications) of the relative locations are used in the inversion. To roughly keep the weighting between relative hypocenter and centroid data, each time the same amount of relative hypocenter location data as that implemented in `NccPy_Inv_HyCe.py` is used; the same for that of centroid. The confidence intervals, standard deviations of each component is given, as well as the confidence ellipses in 3 different cross-sections, are outputted.

❖ Outputs:

- ***bs.joint.(hypo/cent).std***: standard deviation of lat(km), lon(km), dep(km), time(s)
- ***bs.joint.(hypo/cent).errorellipse***: Error ellipse (major-, minor-axis, rotation from top (N/ + dep) of error ellipse) for the 3 cross-sections (lon-lat, lon-dep, lat-dep)
- ***bs.joint.(hypo/cent).ci***: confidence intervals (90%, 95%, 99%) in lat, lon, dep, time
- In directory ***Logs/Inv_BS_HyCe***: additional detailed log files

4.7. NccPy_plot_map.py

❖ In this subroutine:

This subroutine plots a set of quick-and-dirty maps of the results. A total of 4 figures will be auto-generated, including 1) the catalog locations, 2) the joint hypocenter and centroid locations, 3) independently-relocated hypocenter locations, and 4) the independently-relocated centroid locations. Events are plotted with circles sized using magnitude and theoretical area of slip with circular rupture and a constant stress drop assumed. Events whose final locations that are determined using too few effective relative locations are plotted in grey; the rest are plotted in accordance with their magnitudes (magnitude thresholds and colors are specified in the complete parameter file).

❖ Outputs:

- In directory ***Figure/Preliminary***:
  - ***Figure/Preliminary/Ori.pdf***: original catalog location (circles sized using event magnitudes)
  - ***Figure/Preliminary/Reloc_joint.pdf***: final locations for joint hypocenter and centroid locations. Dashed circles are centroid, sized using event magnitudes, and closed thicker and smaller circles are hypocenters.
  - ***Figure/Preliminary/Reloc_solo_hypo.pdf***: separately-determined hypocenters
  - ***Figure/Preliminary/Reloc_solo_cent.pdf***: separately-determined centroids