QuakeSupport User Guide

Ian Lee
Penn State Ice and Climate Research Center (PSICE)
Pennsylvania State University
ianrj.lee@gmail.com

Version 1.0
January 26, 2025

Table of Contents

1. Overview 3

1.1 QuakeSupport 3

1.1.1 QuakeMigrate 3

1.1.2 GrowClust 3

1.2 How QuakeSupport works 3

1.3 Why QuakeSupport? 5

2. Running QuakeSupport 6

2.1 Installing QuakeMigrate 6

2.2 Installing GrowClust 6

2.3 Installing QuakeSupport 6

2.3.1 Running in QuakeMigrate 7

2.3.2 Running in GrowClust 8

3. Input files 9

3.1 QuakeSupport 9

3.1.1 Seismic data download credentials JSON file (`credentials.json`) 9

3.2 QuakeMigrate 9

3.2.1 Station input file (`stations.txt`) 9

3.2.2 QuakeMigrate run scripts (`lut.py`, `detect.py`, `trigger.py`, `locate.py`) 9

3.3 GrowClust 10

3.3.1 Velocity model file (`vzmodel.txt`) 10

3.3.2 Algorithm control file (`growclust.inp`) 10

4. QuakeSupport parameter tuning 10

4.1 QuakeMigrate module 10

4.1.1 `get.py` 10

4.1.2 `align.py` 12

4.1.3 `format.py` 13

4.1.4 `runs.py` 13

4.1.5 `run.py` 15

4.1.6 `reset.py` 15

4.1.7 `QMPicksPlot.py` 15

4.2 GrowClust module 16
4.2.1 QMevID.py 16
4.2.2 QMevlist.py 17
4.2.3 QMstlist.py 17
4.2.4 generate.py 18
4.3 tests module 19
4.3.1 QMcompare.py 19
4.3.2 GCcompare.py 20
4.3.3 run_test.py 20
4.4 Bonus module 20
4.4.1 view.py 21
5. Test example 21
5.1 tests module 21
5.2 Running tests 21
5.2.1 QuakeMigrate 22
5.2.2 GrowClust 24
6. References 26
7. Acknowledgments 27

1. Overview

1.1 QuakeSupport

QuakeSupport is a Python script suite that facilitates a seamless processing pipeline between the seismic analysis software **QuakeMigrate** and **GrowClust** while enhancing data processing speed, particularly for QuakeMigrate. It streamlines the download and preparation of raw seismic data for input into QuakeMigrate, processes the data through QuakeMigrate, and converts the QuakeMigrate outputs for input into GrowClust. This is achieved through a set of modular scripts, each designed to handle a specific stage of the pipeline, ensuring flexibility and ease of use. The modularity allows users to run the entire pipeline or selectively use individual scripts, such as the seismic data download script `get.py`, the waveform plotting script `view.py`, and the QuakeMigrate picks plotting script `QMPicksPlot.py`, for standalone tasks. Users focused solely on running QuakeMigrate can also utilize the dedicated QuakeMigrate module of QuakeSupport.

1.1.1 QuakeMigrate

QuakeMigrate (Winder et al., 2022) is a Python-based software that automates the detection and location of earthquakes through the use of waveform migration and stacking methods. It was first released in 2021, with the latest version as of writing being v1.1.2 (updated on Oct. 23, 2024). The QuakeMigrate GitHub repository can be accessed here.

1.1.2 GrowClust

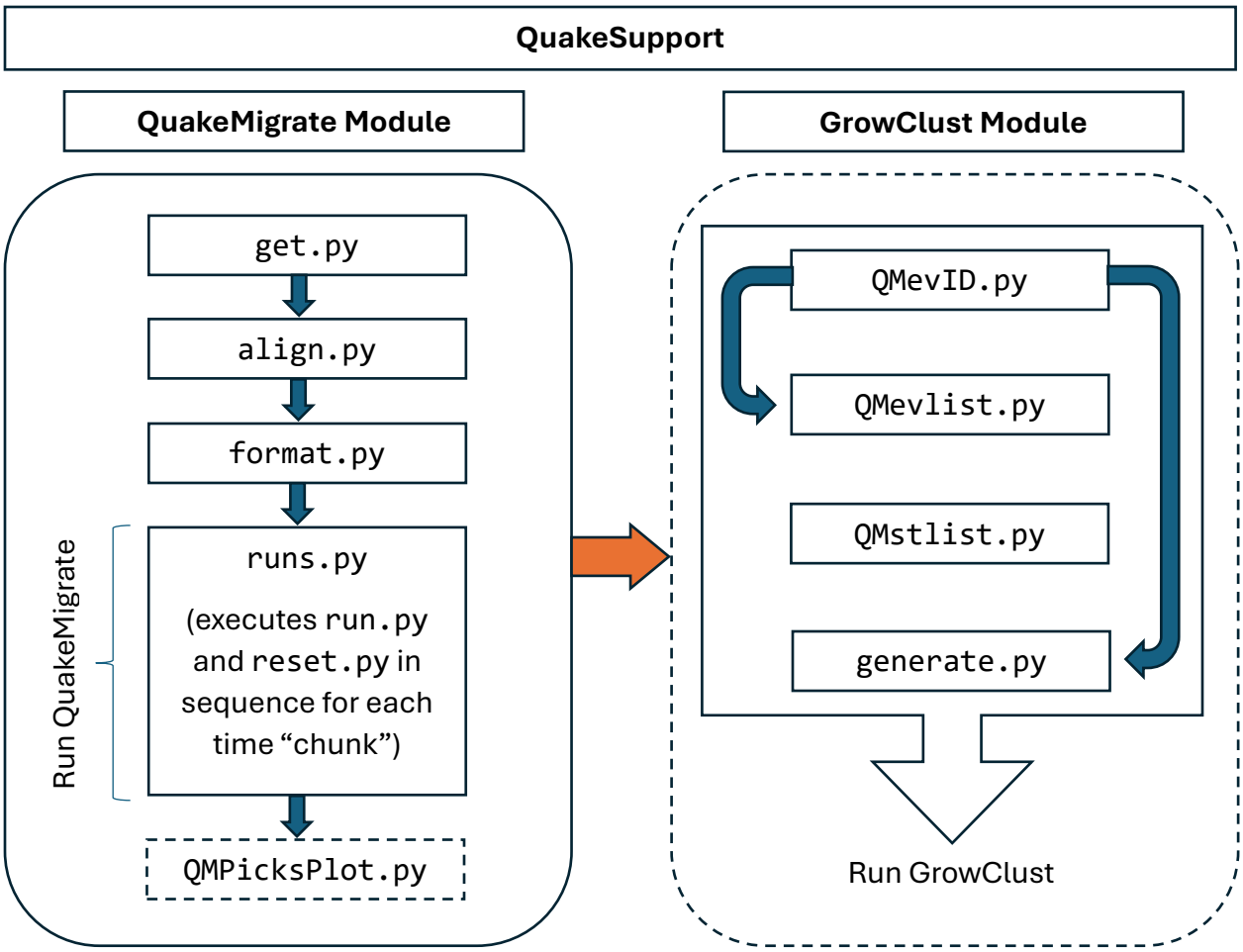
GrowClust (Trugman & Shearer, 2017) is a Fortran-based software for earthquake hypocenter relocation using differential times derived from waveform cross-correlation data. It was first released in 2017, with the latest update (no versioning) as of writing on Oct. 17, 2024. The GrowClust GitHub repository can be accessed here.

The GrowClust module of QuakeSupport has been validated for use with the Fortran version of GrowClust. A Julia version of GrowClust also exists with similar input requirements, such that the data prepared by this module is compatible by design, though this module is specifically intended for the Fortran version.

1.2 How QuakeSupport works

QuakeSupport comprises a suite of scripts designed to enhance and streamline processing in QuakeMigrate and GrowClust, and is intended to be run sequentially as a complete workflow.

QuakeSupport is divided into two primary modules - the QuakeMigrate module and the GrowClust module:



The arrows illustrate the intended sequence for running the QuakeSupport scripts, with scripts enclosed in dashed boxes being optional. Note that users who are interested in only running QuakeMigrate may omit the GrowClust module entirely.

The QuakeMigrate module processes seismic data in time "chunks" to balance processing speed (reading seismic data in QuakeMigrate) and file clutter. The optimal duration of a time chunk depends on the size of the seismic dataset—larger datasets (with high sampling rates and a large number of stations) are best processed in shorter time chunks, resulting in more output files, as each chunk generates its own set of outputs. Conversely, smaller datasets can be processed in longer time chunks, resulting in fewer output files. As a reference, for a dataset with a 1000 Hz sampling rate and approximately 30 stations, a 2-hour time chunk was found to be appropriate. Time chunks can be processed over continuous or discontinuous time periods, supporting different chunk sizes and custom time blocks based on user requirements. QuakeSupport does not modify the QuakeMigrate source code.

The GrowClust module converts outputs generated by the QuakeMigrate module into GrowClust-compatible inputs and works exclusively with the QuakeMigrate module. Note

that generating the cross-correlation input data (`xcordata.txt`) file for GrowClust is computationally expensive, since the total number of event pairs for x events grows quadratically according to the formula $\frac{x(x-1)}{2}$. Each event pair entry includes station-wise travel-time differences and waveform cross-correlations for all seismic phases. Depending on the number of events located by QuakeMigrate, it is **recommended** to group associated events (e.g., clusters) into smaller relocation sets to relocate subsets of events rather than all events. Similarly, QuakeSupport does not modify the GrowClust source code.

1.3 Why QuakeSupport?

QuakeSupport was developed over several years during research on icequakes in Rutford Ice Stream, West Antarctica (Lee et al., 2020, 2021, 2022, 2023, 2024, 2025). The idea to create QuakeSupport arose from the challenges of using QuakeMigrate and GrowClust—both highly effective tools for generating high-granularity seismic event catalogs, but requiring considerable manual effort in data preparation and formatting. Numerous QuakeMigrate runs across various arrays, time periods, and large mSEED (MiniSEED) datasets, combined with iterative parameter tuning, made the process time-consuming and labor-intensive. Running QuakeMigrate included reading seismic data during the detect and locate stages, and with larger mSEED datasets, the reading time also became prohibitively long. To mitigate this, the time period for each QuakeMigrate run and the input mSEED dataset size had to be reduced. However, managing and partitioning QuakeMigrate runs demanded considerable manual effort and careful oversight. To streamline these operations, the QuakeMigrate module was developed to automate the entire QuakeMigrate workflow, from seismic data download to running QuakeMigrate. The GrowClust module simplifies the relocation and refinement of QuakeMigrate event locations by automating the conversion of QuakeSupport's QuakeMigrate outputs into inputs compatible with GrowClust, effectively bridging the gap between the two software packages.

By automatically partitioning QuakeMigrate runs into smaller time chunks, QuakeSupport substantially accelerates QuakeMigrate processing across **extended time periods**. For example, when processing five days of seismic data, QuakeSupport can schedule multiple 2-hour runs, thereby reducing data-reading overhead for each run and speeding up overall QuakeMigrate processing—without requiring users to manage each run individually. QuakeSupport's capability to handle extended QuakeMigrate runs is not limited to continuous time periods; it can also process disjointed time periods (e.g., days 1-2, skipping day 3, then days 4-5). Several scripts leverage multithreading and multiprocessing to enhance performance, enabling even more efficient processing of datasets. Additionally, multiple QuakeSupport runs can be executed concurrently, allowing faster completion of the overall processing workload.

QuakeSupport's greatest strength is its ease of use, achieved by reducing the learning curve associated with the workflows of QuakeMigrate and GrowClust, drawing on years of

hands-on experience working with these software packages and a commitment to sharing that knowledge. This simplicity is further exemplified in QuakeSupport's script design, with dedicated configuration sections that consolidate all user-modifiable parameters in one place, eliminating the need to navigate through the underlying code and reducing potential confusion. By managing the complexities and intricacies of data preparation, data and run partitioning, and execution, QuakeSupport allows users to focus their efforts on scientific analysis. All QuakeSupport scripts are cross-platform compatible and can be run on Linux, Windows, or Mac.

2. Running QuakeSupport

2.1 Installing QuakeMigrate

Instructions on how to install and test QuakeMigrate can be found on the QuakeMigrate GitHub repository, or more specifically on the installation page.

2.2 Installing GrowClust

Instructions on how to install and test GrowClust can be found on the GrowClust GitHub repository, or more specifically in Sections 2 and 5 of the GrowClust user guide.

2.3 Installing QuakeSupport

QuakeSupport can be cloned or downloaded from the QuakeSupport GitHub repository, and its required software packages (e.g., Python libraries and dependencies) are either already included as dependencies of QuakeMigrate or are part of Python's standard library. Therefore, no additional installations are required beyond setting up QuakeMigrate and GrowClust. Similar to QuakeMigrate, Anaconda is recommended for managing both packages and environments to isolate and install the specific dependencies required by QuakeSupport.

QuakeMigrate can be installed via `pip`, which also installs all its explicit dependencies; in that case, the environment and dependency specification files (`environment.yml` or `requirements.txt`) do not need to be run. These files define the required packages and their versions, ensuring consistent dependency management and reproducibility. **Only if** a user wants to run specific QuakeSupport scripts independently (e.g., `get.py` to download seismic data) without requiring a full QuakeMigrate installation—or if a lighter environment is preferred—should the environment and dependency specification files be run. Use `environment.yml` if `conda` is preferred for managing environments and dependencies (run `conda env create -f environment.yml`). Use `requirements.txt` if `pip` is preferred (create a virtual environment, then run `pip install -r requirements.txt` within that environment). Creating a new environment is recommended to avoid package conflicts; if using an existing environment, the `--dry-run` option (`conda env create -f`

`environment.yml --dry-run` or `pip install -r requirements.txt --dry-run`) can be used to preview package changes before applying them.

2.3.1 Running in QuakeMigrate

After cloning or downloading the QuakeSupport GitHub repository, move (or copy) all the QuakeMigrate and GrowClust module scripts and files from `QuakeSupport/quakemigrate` and `QuakeSupport/growclust` into a created or existing QuakeMigrate project directory (e.g., `QuakeMigrate/examples/ProjectName`). Running the QuakeSupport scripts within this directory will populate it with the appropriate files and directories. By default, **all** QuakeSupport scripts are intended to be run in a project directory, though they can be run from any location, provided that the I/O paths (inputs, outputs, and QuakeMigrate run scripts) are appropriately updated in each script's configurations. To execute multiple QuakeSupport runs concurrently, each run should be executed in its own separate project directory (with its own QuakeMigrate and GrowClust module scripts and files). The QuakeMigrate project directory structure with QuakeSupport contents is shown below:

```
QuakeMigrate/ (QuakeMigrate main directory)
├─ examples/
│   └─ ProjectName/ (Project directory)
│      ├─ inputs/
│      │  ├─ mSEED/
│      │  │   └─ ...
│      │  ├─ stations.txt
│      │  └─ ...
│      ├─ outputs/
│      │  └─ ...
│      ├─ QuakeMigrate module files (7 scripts and 1 credentials file)
│      ├─ GrowClust module files (4 scripts)
│      ├─ lut.py
│      ├─ detect.py
│      ├─ trigger.py
│      └─ locate.py
```

To start a QuakeSupport run in QuakeMigrate, only the files and directories highlighted in **green** are needed; the other files and directories will auto-populate the project directory once QuakeMigrate is running. The required input files are described in Section 3.

2.3.2 Running in GrowClust

Running both the QuakeMigrate and GrowClust module scripts in the QuakeMigrate project directory, in the order described in Section 1.2, will create several files. The GrowClust module creates four files: `evID.txt`, `evlist.txt`, `stlist.txt`, and `xcordata.txt`. Move (or copy) **`evlist.txt`**, **`stlist.txt`**, and **`xcordata.txt`** into the `IN` subdirectory of a created or existing GrowClust project directory (e.g., `GrowClust/ProjectName/IN`). These files, along with the velocity model file (`vzmodel.txt`, which users **must** provide) are the required input files for GrowClust. GrowClust can then be run using the algorithm control file (`growclust.inp`, which users **must** also provide) in the project directory. The GrowClust project directory structure with QuakeSupport contents is shown below:

```
GrowClust/ (GrowClust main directory)
├── ProjectName/ (Project directory)
│   ├── IN/
│   │   ├── evlist.txt
│   │   ├── stlist.txt
│   │   ├── xcordata.txt
│   │   └── vzmodel.txt
│   ├── OUT/
│   │   └── ...
│   ├── TT/
│   │   └── ...
│   └── growclust.inp (Algorithm control file)
```

To start a GrowClust run, only the files and directories highlighted in **green** are needed; the other files will auto-populate the project directory once GrowClust is running. As mentioned earlier, the GrowClust module creates the GrowClust input files `evlist.txt`, `stlist.txt`, and `xcordata.txt`. Users **must** provide the `vzmodel.txt` and `growclust.inp` files separately, as these are not part of the QuakeSupport workflow. These two files are described in Section 3. Additionally, users **must** manually create empty `OUT` and `TT` subdirectories in the project directory before running GrowClust.

3. Input files

3.1 QuakeSupport

3.1.1 Seismic data download credentials JSON file (`credentials.json`)

Depending on the seismic dataset being downloaded and/or the data center being accessed, some data may be restricted and require download credentials, consisting of a username and password. To keep credentials secure, they can be stored in a JSON file (`credentials.json`), which is accessed by `get.py` during the data download process. For unrestricted datasets, the username and password fields can be left empty – an empty template for `credentials.json` is provided in the QuakeMigrate module.

3.2 QuakeMigrate

3.2.1 Station input file (`stations.txt`)

A list of stations to be used in the QuakeMigrate run. Accepted formatting for the station input file is detailed in QuakeMigrate's documentation, either online or on page 23 of the PDF version (latest: Oct. 21, 2023). A sample station input file can be found here.

3.2.2 QuakeMigrate run scripts (`lut.py`, `detect.py`, `trigger.py`, `locate.py`)

These four scripts are executed sequentially (in the order listed) in a QuakeMigrate run. QuakeSupport automates the execution sequence and dynamically updates the run start and end times, as well as run names, in the relevant scripts to partition the runs. Note that **QuakeMigrate run script parameters still need to be tuned by users for their specific use cases to achieve meaningful results**, except for `starttime`, `endtime`, and `run_name` in the `detect.py`, `trigger.py`, and `locate.py` scripts, which are handled by QuakeSupport. For guidance on tuning QuakeMigrate parameters, refer to the QuakeMigrate documentation (online or PDF version, latest: Oct. 21, 2023), examples, and the dissertations of two of its developers, Conor Bacon (2021) and Tom Winder (2021).

* By default, `lut.py` uses a homogeneous velocity model to generate the travel-time lookup table (LUT) and does not require a `vmodel.txt` input file (distinct from GrowClust's `vzmodel.txt`, which may have similar values but a slightly different format). For 1-D velocity models, refer to the documentation (online or on page 24 of the PDF version, latest: Oct. 21, 2023) and an example for guidance on creating the `vmodel.txt` file.

** In `detect.py` and `locate.py`, increase `scan.threads` based on the number of available CPU threads in the user's system to improve processing speed.

3.3 GrowClust

3.3.1 Velocity model file (`vzmodel.txt`)

GrowClust relies on the velocity model specified in this file to compute the predicted differential travel-times between event pairs at common stations. For guidance on formatting `vzmodel.txt`, refer to GrowClust's example and pages 13-14 of its user guide.

3.3.2 Algorithm control file (`growclust.inp`)

This file is read by GrowClust at the start of execution and defines the required input files, program options, and algorithm parameters used during the GrowClust run. For guidance on configuring and formatting `growclust.inp`, refer to the GrowClust example and pages 4-6 of its user guide.

4. QuakeSupport parameter tuning

A description of its functionality and an explanation of each configuration parameter are provided for each QuakeSupport script. Users only need to modify parameters within the demarcated "**Configurations**" section.

4.1 QuakeMigrate module

4.1.1 `get.py`

Script to download seismic data and instrument response inventory. It supports data download from FDSN-compliant data centers and leverages multithreading to enable concurrent downloads, reducing overall download times. The script can download either regular, continuous time chunks or variable, disjointed time chunks. Upon execution, it automatically creates the necessary `inputs` directory and subdirectories for storing downloaded data and logs if they do not already exist. If a download fails (due to reasons such as invalid arguments or temporary data unavailability) and raises an exception, it is retried up to 5 times; this does not affect other concurrent, ongoing downloads.

`get.py` can also be used, along with `align.py` if needed, as a **standalone, fast seismic data downloader**. The default formats are mSEED for seismic data and StationXML for instrument response inventory—both the primary formats supported by QuakeMigrate. If other formats are required for non-QuakeMigrate purposes (e.g., SAC or SEGY for seismic data; RESP, SC3ML, or DATALESS for instrument response inventory), update the `format` arguments accordingly. When using formats other than mSEED, any reference to mSEED (in any capitalization) throughout `get.py` and `align.py` refers to the configured format.

`get.py`, along with the two other QuakeMigrate pre-processing scripts `align.py` and `format.py`, does not explicitly sort traces in a downloaded stream; traces follow the

sequence defined in the `station_input` configuration for stations, and channels are automatically sorted in alphabetical order based on the inherent sorting of the `Stream` object returned by `client.get_waveforms`. If `get.py` and `align.py` are used for non-QuakeMigrate purposes, keep this in mind.

Configuration parameters

- `credentials_file`: Path to the download credentials file.
- `r_mseed_path`: Path to the directory for storing raw (downloaded) mSEED files (created if it does not exist).
- `log_path`: Path to the directory for storing log files (created if it does not exist).
- `response_file`: Path to write the instrument response inventory file.
- `seismic_format`: Format of the downloaded seismic data.
- `response_format`: Format of the downloaded instrument response inventory file.
- `network`: Name of the network.
- `datacenter`: Name of the data center.
- `station_input`: Station codes. Accepts a string for a single station, a wildcard string for multiple stations, or a list of station codes for even more granular control.
- `location_input`: Location code.
- `channel_input`: Instrument channels.
- `time_buffer`: Additional time, in seconds, added to the start and end of each raw (downloaded) time chunk to prevent data gaps during a QuakeMigrate run. A default value of 300 seconds is recommended.
- `time_type`: Type of data time intervals (or time chunks). Choice between regular time chunks (1) or custom time chunks (2).
- `starttime`: Start time of the initial time chunk in the processing window. Also used to write the instrument response inventory file. If using custom times, set to the first start time entry in `times`.
- `endtime`: End time of the initial time chunk in the processing window. Defined as `starttime` + `chunk_size`. Also used to write the instrument response inventory file. If using custom times, set to the first end time entry in `times`.
- `time_chunks`: Number of regular time chunks. Ignore if using custom time chunks.
- `chunk_size`: Size of regular time chunk, in seconds. Ignore if using custom time chunks.
- `times`: List of custom start and end time pairs. Ignore if using regular time chunks.
- `workers`: The number of concurrent workers used for multithreading. Adjust based on the data center's concurrent download limit (e.g., `IRISPH5` is limited to 6).
- `max_retries`: The number of retries specified if a download fails. Capped at 5.
- `retry_backoff`: The wait time, in seconds, between each download retry. Capped at 180 seconds (3 minutes).

4.1.2 align.py

Script to align offset stream trace times. If the seismic data sampling period (e.g., 1 ms sampling period for a 1000 Hz sampling rate) exceeds the timestamp precision (e.g., microseconds for `IRISPH5` timestamps), some stream traces may have time offsets, relative to the intended timestamps, that are smaller than the sampling period. These offsets result from floating-point rounding during time calculations or data processing. While these offsets do not affect data quality, alignment of traces is recommended to ensure exact time synchronization when needed. QuakeMigrate performs in-memory stream trace alignment during a run as needed, but the alignment does not persist afterward and logs a message for each trace realigned. This script bypasses these issues by permanently aligning stream traces and preventing log clutter.

Configuration parameters

- `r_mseed_path`: Path to the raw (downloaded) mSEED directory.
- `log_path`: Path to the directory for storing log files (created if it does not exist).
- `a_mseed_path`: Path to the directory for storing aligned mSEED files (created if it does not exist).
- `seismic_format`: Format of the aligned seismic data. Match `seismic_format` in `get.py` if running at least the QuakeMigrate module. Conversion between formats can result in metadata or precision loss.
- `mseed_pattern`: Wildcard pattern to match target raw mSEED files. Alternatively, to avoid matching unwanted files, set `r_mseed_path` to a directory containing only the target files.
- `fs`: Trace sampling frequency, in Hz.
- `time_buffer`: Additional time, in seconds, added to the start and end of each raw (downloaded) time chunk to prevent data gaps during a QuakeMigrate run. Match the value of `time_buffer` in `get.py`.
- `time_type`: Type of data time intervals (or time chunks). Choice between regular time chunks (1) or custom time chunks (2). Match `time_type` in `get.py`.
- `starttime`: Start time of the initial time chunk in the processing window. Match `starttime` in `get.py`. Ignore if using custom time chunks.
- `time_chunks`: Number of regular time chunks. Match the value of `time_chunks` in `get.py`. Ignore if using custom time chunks.
- `chunk_size`: Size of regular time chunk, in seconds. Match the value of `chunk_size` in `get.py`. Ignore if using custom time chunks.
- `starttimes`: List of custom start times. Match the start times in `times` in `get.py`. Ignore if using regular time chunks.
- `verbose_logging`: Enables detailed logging, including time shift and stream information, when set to `True`.

4.1.3 format.py

Script to prepare mSEED files for QuakeMigrate input. It zero-centers stream trace data and splits each stream into individual station-channel traces, formatting trace file names as `yearJulianday_starttime_station_channel.mseed`. Each formatted file is written to its corresponding subdirectory within QuakeMigrate's `mSEED` subdirectory, following the structure: year → Julian day and time chunk. Streams spanning two Julian days have their trace files duplicated in both Julian day subdirectories, ensuring QuakeMigrate can access them during runs.

Configuration parameters

- `a_mseed_path`: Path to the aligned mSEED directory.
- `log_path`: Path to the directory for storing log files (created if it does not exist).
- `mseed_path`: Path to the QuakeMigrate `inputs/mSEED` subdirectory (created if it does not exist).
- `mseed_pattern`: Wildcard pattern to match target aligned mSEED files. Alternatively, to avoid matching unwanted files, set `a_mseed_path` to a directory containing only the target files.
- `channels`: Seismogram channels. Order of channels listed does not matter, since each station-channel trace file is created individually.
- `verbose_logging`: Enables detailed logging, including trace information, when set to `True`.

4.1.4 runs.py

Script to perform multiple, sequential QuakeMigrate runs in the same year. It partitions the specified time period into separate runs of regular or custom time chunks by iterating through the process of updating the QuakeMigrate run scripts' (specifically `detect.py`, `trigger.py`, and `locate.py`) run name and times, unlocking the relevant input mSEED subdirectory, and executing each run.

During a run, QuakeMigrate reads mSEED files matching the `archive_format` wildcard (default: `YEAR/JD/*_STATION_*`). To access the relevant Julian day and time chunk subdirectory (or subdirectories for runs spanning two Julian days), `runs.py` "unlocks" the subdirectory by renaming it from its full identifier—containing the Julian day and chunk start/end times—to just the Julian day. This renaming allows QuakeMigrate to match the Julian day subdirectory (`JD`) using the wildcard pattern, enabling access to the relevant subdirectory. After the run is complete, the subdirectory is "relocked" by renaming it back to its full identifier. This lock-unlock process repeats for all runs. If an error is encountered during a run, a fail-safe mechanism relocks the Julian day subdirectory and resets the QuakeMigrate run scripts for the affected run. However, if `runs.py` is forcefully terminated, any Julian day subdirectory that was in use by it will remain unlocked and must be

manually relocked before re-running. QuakeMigrate reads entire mSEED files during the detect and locate stages, which can slow processing if the subdirectory includes large files or files where only a small portion of the data is used for the run. QuakeSupport mitigates this issue by facilitating QuakeMigrate's reading of only smaller, relevant files.

Multiple concurrent runs can be achieved by running multiple instances of `runs.py`. For example, days 1-10 can be split into separate runs, such as 1-3, 4-6, and 7-10. Care must be taken to avoid overlap when locking and unlocking Julian day subdirectories, as this can cause conflicts and disrupt runs. An instance of `runs.py` should not cross over years.

Configuration parameters

- `mseed_path`: Path to the QuakeMigrate `inputs/mSEED` subdirectory.
- `qm_scripts`: Detect, trigger, and locate QuakeMigrate run script names (in order).
- `run_script`: Corresponding `run.py` script name.
- `reset_script`: Corresponding `reset.py` script name.
- `qm_run_name`: QuakeMigrate run name. Match `run_name` in the detect, trigger, and locate QuakeMigrate run scripts.
- `year`: Run year. An instance of `runs.py` should not cross over years.
- `time_buffer`: Additional time, in seconds, added to the start and end of each raw (downloaded) time chunk to prevent data gaps during a QuakeMigrate run. Match the value of `time_buffer` in `get.py`.
- `det_buffer`: Additional time, in seconds, applied to the start and end of the time period for the detect stage of a QuakeMigrate run, accounting for QuakeMigrate's pre/post time padding (120 seconds). Double that time (240 seconds) is suggested.
- `time_type`: Type of data time intervals (or time chunks). Choice between regular time chunks (1) or custom time chunks (2). Match `time_type` in `get.py`.
- `selected_starttime`: Start time of the first time chunk to be processed by `runs.py`. It must be an integer multiple of `chunk_size` from the initial time chunk start time. For example, if the initial time chunk start time is `2023-01-06T00:00:00.000000Z` and `chunk_size` is `7200` seconds (2 hours), valid start times are `2023-01-06T02:00:00.000000Z`, `2023-01-06T04:00:00.000000Z`, and so on, as long as the run remains within the available data time period. This enables flexibility in selecting time ranges, such as for multiple concurrent runs. Ignore if using custom time chunks.
- `time_chunks`: Number of regular time chunks to be processed by `runs.py`, equal to the desired number of QuakeMigrate runs. For example, `time_chunks=3` schedules three consecutive runs, each with a duration of `chunk_size`, starting at `selected_starttime`. Ignore if using custom time chunks.
- `chunk_size`: Size of regular time chunk, in seconds. Match the value of `chunk_size` in `get.py`. Ignore if using custom time chunks.

- `times`: List of custom start and end time pairs. Ignore if using regular time chunks.

4.1.5 `run.py`

Script to automate QuakeMigrate runs. It executes QuakeMigrate's core processing workflow, consisting of the LUT, detect, trigger, and locate stages, in order. The `runs.py` script calls `run.py` for each QuakeMigrate run iteration.

Configuration parameters

- `qm_scripts`: LUT, detect, trigger, and locate QuakeMigrate run script names (in order).

4.1.6 `reset.py`

Script to reset QuakeMigrate scripts with template run name and times. It overwrites specific lines in the QuakeMigrate run scripts by searching for predefined Regex patterns (allowing optional whitespace) associated with the run name and time variables. Note that the Regex patterns defined here match from the start of each line. This script is executed by `runs.py` once at the start and again at the end of every QuakeMigrate run iteration.

Configuration parameters

- `reset_scripts`: Detect, trigger, and locate QuakeMigrate run script names (in order); these are the scripts that need to be updated for each QuakeMigrate run.
- `qm_run_name`: QuakeMigrate run name. Match `qm_run_name` in `runs.py`.

4.1.7 `QMPicksPlot.py`

Script to plot QuakeMigrate modeled and observed picks. It uses the `picks` and `raw_cut_waveforms` outputs from QuakeMigrate runs to generate PDF plots for each event, with a separate plot for each channel displaying all stations. This complements QuakeMigrate's phase pick plots, which display observed picks for individual station-channel pairs for each event, and summary plots, which display modeled picks for each event. Note that QuakeMigrate labels their `raw_cut_waveforms` mSEED files with the `.m` extension. Be mindful of this to avoid confusion with MATLAB files, as both use the `.m` extension. QuakeMigrate's `raw_cut_waveforms` output `.m` files contain unsorted traces; this script sorts them by station before plotting, but keep this in mind for other uses.

`QMPicksPlot.py` can also be used as a standalone pick plotting script with any QuakeMigrate run(s), including those that were not executed through QuakeSupport.

Configuration parameters

- `runs_path`: Path to the QuakeMigrate `outputs/runs` subdirectory. The default `outputs/runs` path will generate plots for all events from all QuakeMigrate runs. To

target specific events or even time periods, set `runs_path` to specific QuakeMigrate run subdirectories, modify the wildcard patterns (`picks_pattern`, `wf_pattern`), or isolate the relevant `.picks` and `.m` files in a separate subdirectory and point to it.

- `plot_path`: Path to save the PDF pick plots.
- `picks_pattern`: Wildcard pattern to match target pick (`.picks`) files.
- `wf_pattern`: Wildcard pattern to match target `raw_cut_waveforms` (`.m`) files.
- `channels`: Seismogram channels to plot.
- `vertical_comp`: Vertical component channels. Leave as an empty list if not applicable.
- `horizontal_comp`: Horizontal component channels. Leave as an empty list if not applicable.
- `pick_zoom`: Set to `True` to zoom in around observed pick time.
- `pick_window`: Time window (± seconds) around observed pick time if `pick_zoom` is `True`.
- `ylims_flag`: Set to `True` to use custom y-axis limits, `False` for automatic scaling.
- `ymin`: Minimum y-axis limit if using custom y-axis limits.
- `ymax`: Maximum y-axis limit if using custom y-axis limits.

4.2 GrowClust module

4.2.1 `QMevID.py`

Script to write QuakeMigrate event ID file. The script's output file, `evID.txt`, is used by `QMevlist.py` and `generate.py` to create their respective GrowClust input files. Each event in `evID.txt` has two columns: the first is a sequential QuakeSupport-assigned event ID, mapped to the second column, the QuakeMigrate-assigned event ID. The sequential IDs, zero-padded to 7 digits and supported up to 9,999,999 events, are necessary because QuakeMigrate-assigned event IDs exceed Fortran's integer limit and cause errors in GrowClust. When `QMevlist.py` and `generate.py` use `evID.txt`, they read and process the sequential IDs as the event IDs. For GrowClust post-processing analysis, use `evID.txt` to map the sequential IDs back to the original QuakeMigrate-assigned event IDs if needed. Sample `evID.txt`.

Before relocating QuakeMigrate events with GrowClust, it is **recommended** to group associated events (e.g., clusters) into smaller relocation sets. Since these events are more likely to share similar waveforms, GrowClust's cross-correlation-based relocation becomes more accurate and processes faster—because the number of event pairs grows quadratically, $\frac{x(x-1)}{2}$, with the number of events. To that end, the default `QMevID.py` script writes all events from all QuakeMigrate runs, but it is primarily intended as a **template** for users to adapt when generating their own `evID.txt` file, which is also simple to manually create without needing to use this script.

Configuration parameters

- `runs_path`: Path to the QuakeMigrate `outputs/runs` subdirectory. The default `outputs/runs` path writes all events from all QuakeMigrate runs. To target specific events after applying a post-QuakeMigrate event filter, set `runs_path` to specific QuakeMigrate run subdirectories, modify the wildcard pattern (`events_pattern`), or isolate the relevant `.event` files in a separate subdirectory and point to it. Alternatively, create an `evID.txt` file manually, including the sequential IDs.
- `evID_file`: Path to write the QuakeMigrate event ID file (`evID.txt`).
- `events_pattern`: Wildcard pattern to match target event (`.event`) files.

4.2.2 QMevlist.py

Script to write GrowClust evlist input file from QuakeMigrate event files. The format written follows the phase format (refer to Section 3.1.1 in the GrowClust user guide). The location error fields—eh (horizontal), ez (vertical), and rms (RMS)—differ from the equivalent fields GrowClust uses internally and are ignored during processing, so they are assigned `"0.000"` as a placeholder.

Configuration parameters

- `evID_file`: Path to the QuakeMigrate event ID file (`evID.txt`).
- `runs_path`: Path to the QuakeMigrate `outputs/runs` subdirectory. Recommended to set `runs_path` to the default `outputs/runs` path to allow access to all events from all QuakeMigrate runs, as this script can traverse all runs and extract the target events.
- `evlist_file`: Path to write the GrowClust evlist input file (`evlist.txt`).
- `latdp`: Latitude field decimal precision, consistent with GrowClust defaults (5).
- `londp`: Longitude field decimal precision, consistent with GrowClust defaults (5).
- `depdp`: Depth field decimal precision, consistent with GrowClust defaults (3).
- `magdp`: Magnitude field decimal precision, consistent with GrowClust defaults (2).
- `dep_unit`: Depth unit (m or km) defined for QuakeMigrate events. Match the `units` variable defined in QuakeMigrate's `lut.py` script, as it determines the depth unit of the QuakeMigrate outputs. If `dep_unit = "m"`, this script converts event depths to kilometers, the depth unit required by GrowClust for `evlist.txt`.
- `verbose`: Provides detailed output, including event information, when set to `True`.

4.2.3 QMstlist.py

Script to write GrowClust stlist input file from QuakeMigrate station file. The format written follows the station name format, with elevation field (refer to Section 3.2.2 in the GrowClust user guide). GrowClust requires the station names to be between 3 and 5 characters in length, excluding blanks.

Configuration parameters

- `station_file`: Path to the QuakeMigrate station input file (`stations.txt`).
- `stlist_file`: Path to write the GrowClust stlist input file (`stlist.txt`).
- `elev_unit`: Elevation unit (m or km) defined for QuakeMigrate stations. If QuakeMigrate is tuned correctly, `elev_unit` should match `dep_unit` in `QMevlist.py`. If `elev_unit = "km"`, this script converts station elevations to meters, the elevation unit required by GrowClust for `stlist.txt` (note: this is the opposite of the required event depth unit [kilometers] for `evlist.txt`).
- `latdp`: Latitude field decimal precision, consistent with GrowClust defaults (4).
- `londp`: Longitude field decimal precision, consistent with GrowClust defaults (4).
- `elevdp`: Elevation field decimal precision, consistent with GrowClust defaults (0) and seismology standards, which record station elevations in meters as an integer.

4.2.4 `generate.py`

Script to generate GrowClust xcordata input file from QuakeMigrate outputs. The format written follows the text file (dt.cc) format (refer to Section 3.3.1 in the GrowClust user guide), which is compatible with both GrowClust and HypoDD, another event relocation program. The script leverages multiprocessing to perform event pairwise cross-correlation computations in parallel, enhancing computational efficiency. Computations are executed sequentially: event 1 is cross-correlated with events 2, 3, 4, and so on up to the last event; then event 2 with events 3 through the last event, and so forth. With multiprocessing, computations for later-ordered event pairs may complete before those for earlier-ordered pairs, but their outputs in `xcordata.txt` are written in sequential order.

`generate.py` is designed to work with QuakeSupport's QuakeMigrate module-generated QuakeMigrate outputs. It assumes the following about these outputs: (1) events are derived from the same stations, (2) run directories consist of regular time chunks (e.g., 2-hour runs) and do not use custom time chunks, and (3) run directories follow the `runname_starttime_endtime` naming convention, incorporating the `run_name` variable in `runs.py` used to name each run. QuakeMigrate's `raw_cut_waveforms` output `.m` files contain unsorted traces; this script sorts them first by phase and then by station before writing the cross-correlation results, but keep this in mind for other uses. The origin time correction (OTC) field is not used by GrowClust, so it is assigned `"0.000"` as a placeholder.

Configuration parameters

- `evID_file`: Path to the QuakeMigrate event ID file (`evID.txt`).
- `runs_path`: Path to the QuakeMigrate `outputs/runs` subdirectory. Recommended to set `runs_path` to the default `outputs/runs` path to allow access to all events from all QuakeMigrate runs, as this script can traverse all runs and extract the target events, including their `raw_cut_waveforms`, `picks`, and `event` files.

- `xcordata_file`: Path to write the GrowClust xcordata input file (`xcordata.txt`).
- `unique_phases`: QuakeMigrate seismic phases. Available phases are "P" and "S".
- `channels`: Seismogram channels. Align with order of phases in `unique_phases`, so for example if "P" and "S" phases are listed, include the P channels first before the S channels.
- `starttime`: Start time of the initial time chunk in the processing window. Match `starttime` in `get.py`.
- `chunk_size`: Size of regular time chunk, in seconds. Match the value of `chunk_size` in `get.py`.
- `tdif_fmt`: Differential time sign convention for event pairs (12: t1 - t2, 21: t2 - t1). Match `tdif_fmt` in `growclust.inp`.
- `fs`: Trace sampling frequency, in Hz, of `raw_cut_waveforms .m` data. Match the value of `fs` in `align.py`, as the original sampling frequency is retained even if data was downsampled during the QuakeMigrate run.
- `xcor_P_window`: P-phase cross-correlation window length before and after the pick time. Units are in samples (`1/fs`), based on data sampling frequency (`fs`).
- `xcor_S_window`: S-phase cross-correlation window length before and after the pick time. Units are in samples (`1/fs`), based on data sampling frequency (`fs`).
- `ttddp`: Travel-time differential field decimal precision, consistent with GrowClust defaults (5).
- `xcordp`: Cross-correlation value field decimal precision, consistent with GrowClust defaults (4).
- `prog_threshold`: Set progress threshold for pairwise event computations, such that an alert is output for every *nth* event that has been pairwise computed with all subsequent (later-ordered) events.
- `num_workers`: Define number of workers for multiprocessing, or use `cpu_count()` to default to the number of available CPU cores.
- `user_defined_workers`: Specify a different number of workers from the default, if needed.

4.3 `tests` module

4.3.1 `QMcompare.py`

Script to validate QuakeSupport's QuakeMigrate test run against benchmark results. If differences between the test and benchmarked results are reported, tools such as ObsPy for waveform data, XML comparison utilities for metadata, and text comparison tools for text files can be used to visualize and analyze the discrepancies. A numerical tolerance of 1e-5 is applied for numerical comparisons to account for floating-point variations across operating systems and QuakeMigrate versions.

Configuration parameters

- `verbose`: Provides detailed output, including the comparison status of individual file pairs to assist with troubleshooting, when set to `True`.

4.3.2 `GCcompare.py`

Script to validate QuakeSupport's GrowClust test run against benchmark results. It parses and compares the test and benchmarked GrowClust input and output files (in free format) to ensure the corresponding files are identical. A numerical tolerance of 1e-5 is applied for numerical comparisons to account for floating-point variations across operating systems and between different versions of QuakeMigrate and GrowClust.

Configuration parameters

- `evlist_file`: Path to the evlist input file (`evlist.txt`). Point to its location or move (or copy) the file to the QuakeMigrate `tests` project directory with `GCcompare.py` to use the default path (`"./"`).
- `stlist_file`: Path to the stlist input file (`stlist.txt`). Point to its location or move (or copy) the file to the QuakeMigrate `tests` project directory with `GCcompare.py` to use the default path (`"./"`).
- `xcordata_file`: Path to the xcordata input file (`xcordata.txt`). Point to its location or move (or copy) the file to the QuakeMigrate `tests` project directory with `GCcompare.py` to use the default path (`"./"`).
- `fout_cat_file`: Path to the relocated event catalog output file (`out.growclust_cat`). Point to its location or move (or copy) the file to the QuakeMigrate `tests` project directory with `GCcompare.py` to use the default path (`"./"`). The focus is on the relocated events (`out.growclust_cat`), as GrowClust's waveform similarity clustering (`out.growclust_clust`) is outside the scope.
- `verbose`: Provides detailed output, including specific line differences to assist with troubleshooting, when set to `True`.

4.3.3 `run_test.py`

Script to run QuakeSupport test run scripts. Run in the QuakeMigrate `tests` project directory to execute the complete workflow, including the QuakeMigrate and GrowClust modules (up to `generate.py`, creating the required inputs to then run GrowClust).

No configuration parameters

4.4 Bonus module

This module is supplementary to the QuakeSupport workflow, offering additional tools for users.

4.4.1 `view.py`

Script to read and plot seismic waveforms. It provides a simple tool for inspecting waveforms at any point during the QuakeSupport workflow, aiding quality control, troubleshooting, and data analysis. This script uses ObsPy's `plot()` method and displays full waveforms, with filters for stations and channels. Interactive features such as panning and zooming are enabled. For a more advanced waveform inspection tool with more granular control and greater functionality, consider using Snuffler.

`view.py` can also be used as a standalone script to read and plot seismic waveforms in formats supported by ObsPy (e.g., mSEED, SAC, SEGY). Specify the appropriate seismic file and apply station and channel filters as needed.

Configuration parameters

- `seismic_file`: Path to the seismic file. Supports formats compatible with ObsPy.
- `station`: String or wildcard string to filter for stream stations.
- `channel`: String or wildcard string to filter for stream channels.

5. Test example

5.1 `tests` module

The `tests` module in the QuakeSupport GitHub repository enables users to test and verify the core functionality of the QuakeSupport software in both QuakeMigrate and GrowClust. This module contains 19 files and 3 folders, which includes all the required benchmarked results for comparing outputs from the test run. Included as well are all relevant QuakeMigrate and GrowClust module scripts required for a QuakeSupport run, along with the `credentials.json` download credentials file, the four core QuakeMigrate run scripts (`lut.py`, `detect.py`, `trigger.py`, `locate.py`), the `run_test.py` script to execute the test run, and the `QMcompare.py` and `GCcompare.py` scripts to validate the test run.

No parameter tuning is required for the `tests` module scripts, as they have been pre-configured for the test run. The `tests` module folder simply needs to be placed in the appropriate subdirectory and run. The test run covers a short time period (just under a minute), and is partitioned into three regular, continuous QuakeMigrate runs primarily to demonstrate QuakeSupport's ability to partition runs. The test run data is derived from the 2018-2019 Rutford Ice Stream 5B network (Anandakrishnan, 2018), specifically from 13 stations in the western inland array.

5.2 Running `tests`

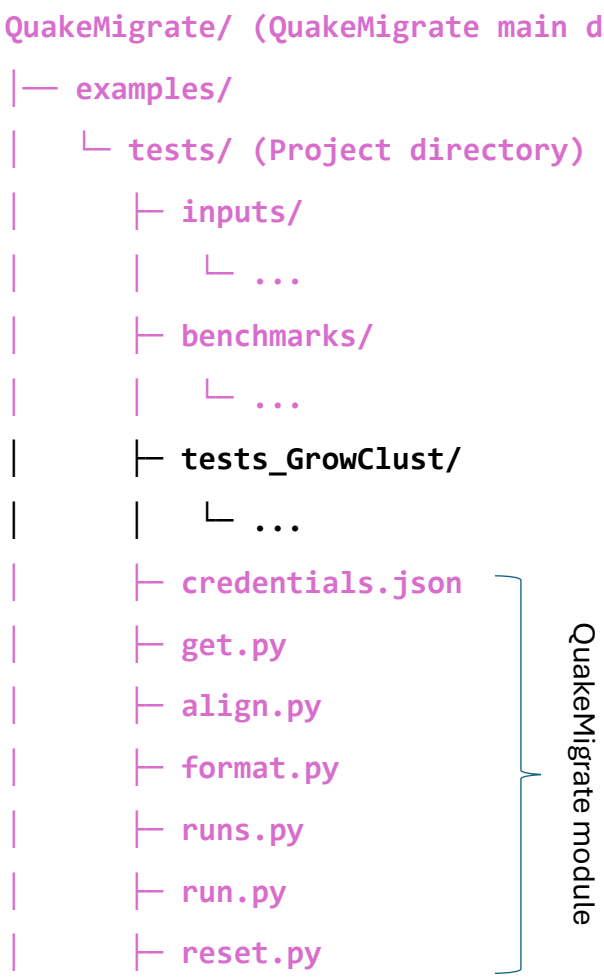
The test run involves executing QuakeMigrate and GrowClust within their respective `tests` project directories, which have been prepared and provided by the `tests` module.

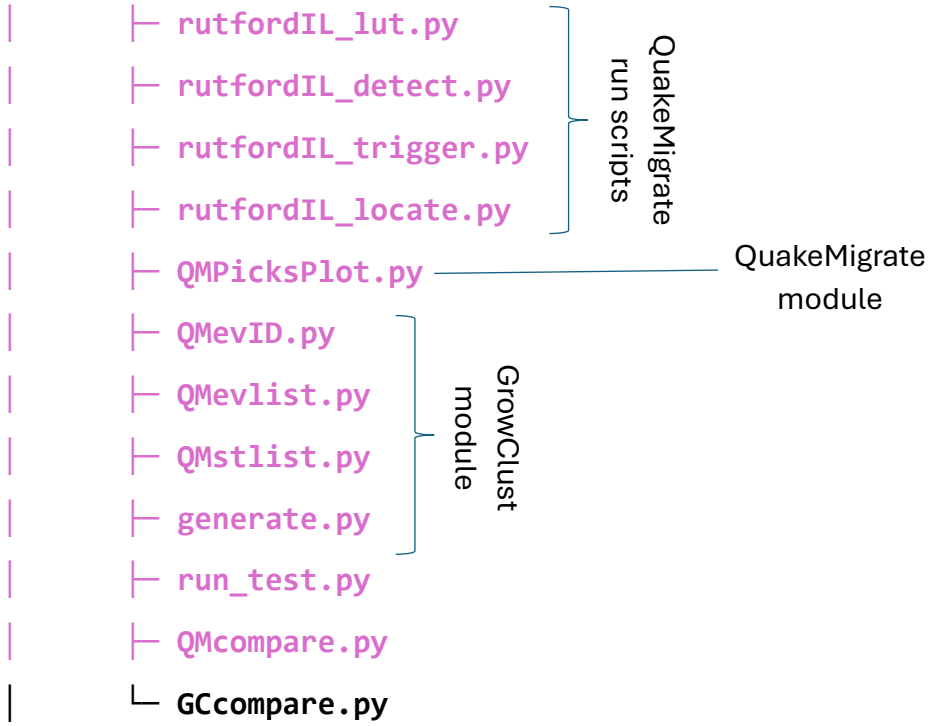
5.2.1 QuakeMigrate

The following steps outline the QuakeMigrate section of the test run:

1. **Install QuakeMigrate** (refer to Section 2.1). For the simplest route to a working copy, installation via `pip` is recommended. The steps are: i) Install Anaconda, ii) Create a minimal environment, iii) Install QuakeMigrate within that environment using `pip` (which will also install its explicit dependencies), iv) Clone or download the QuakeMigrate GitHub repository, and v) Test the installation to verify functionality.

2. **Set up the `tests` QuakeMigrate project directory**. After cloning or downloading the QuakeSupport GitHub repository, move (or copy) the `tests` module folder to the QuakeMigrate `examples` subdirectory (`QuakeMigrate/examples`). The `tests` module folder functions as a QuakeMigrate project directory and contains all the required files for a QuakeSupport run. The `tests` QuakeMigrate project directory structure should look like this:





Only the files and directories highlighted in **purple** are involved in the QuakeMigrate section of the test run.

3. **Execute the test run**. In the `tests` project directory, initiate the QuakeMigrate test run by running `python run_test.py`. This script collates and executes the scripts in the QuakeMigrate and GrowClust modules. The test run will take several minutes to complete. During the run, the terminal output displays updates from the current script being executed and the status of data processing.

   Notes on the test run: 1) In `rutfordIL_detect.py` and `rutfordIL_locate.py`, `scan.threads` is set to 4 for compatibility with lower-spec systems, but it can be increased based on the number of available CPU threads in the user's system to improve processing speed. 2) During the QuakeMigrate trigger stage, two warnings will appear: "`Warning! No .scanmseed data found for pre-pad!`" and "`Warning! No .scanmseed data found for post-pad!`" These can be ignored, as the test run time period is under a minute and does not require the 120-second time buffer used for QuakeMigrate's default pre/post time padding. 3) `QMPicksPlot.py` generates sample QuakeMigrate pick plots from this test run.

4. **Validate the test run**. Once `run_test.py` has executed successfully, run `python QMcompare.py` in the `tests` project directory. This script compares seven items: the downloaded instrument response inventory, the raw (downloaded), aligned, and QuakeMigrate-formatted mSEED data, and the QuakeMigrate `raw_cut_waveforms`,

`events`, and `picks` outputs against their respective benchmarked results. For each item, the script compares every file pair, outputting an "**OK**" message if all pairs are identical. If all seven items match their benchmarked results, the script outputs "**All comparisons passed**," confirming the validation of the QuakeMigrate test run.

5.2.2 GrowClust

The following steps outline the GrowClust section of the test run:

1. **Install GrowClust** (refer to Section 2.2). The steps are: i) Clone or download the GrowClust GitHub repository, ii) Compile GrowClust (refer to Section 2.1 of the GrowClust user guide), and iii) Test the installation by running the tutorial example (refer to Section 5 of the GrowClust user guide). Step iii) is optional and can be run post-test run, as the tutorial example guides users through an example GrowClust run and focuses on parameter tuning rather than testing functionality.

   As stated in Section 1.1.2, the GrowClust module of QuakeSupport has been validated for use with the Fortran version of GrowClust. While a Julia version of GrowClust also exists with similar input requirements, this module is specifically intended for the Fortran version.

   Notes on Fortran compilation across operating systems: 1) **Windows** lacks native support for Fortran compilation, so installing a program such as Windows Subsystem for Linux (WSL) provides a seamless Linux environment to compile and run Fortran code. 2) **Mac** requires the installation of Xcode Command Line Tools and GCC (which includes `gfortran`), recommended via Homebrew. Update Xcode Command Line Tools and system libraries using `softwareupdate --all --install --force` to ensure compatibility, after which `make` can be run to compile and run Fortran code. 3) **Linux** provides the necessary tools (`gfortran`, `make`) via package managers, allowing easy compilation with `make`.

2. **Set up the `tests_GrowClust` GrowClust project directory.** After setting up the `tests` QuakeMigrate project directory (step 2 of the QuakeMigrate section of the test run), **move** the `tests_GrowClust` folder from the `tests` project directory to the GrowClust repository main directory (`GrowClust`). The `tests_GrowClust` folder functions as a GrowClust project directory. After `run_test.py` has executed successfully during the QuakeMigrate section of the test run, **copy** the created GrowClust input files **`evlist.txt`**, **`stlist.txt`**, and **`xcordata.txt`** in the `tests` project directory to the `IN` subdirectory of the `tests_GrowClust` project directory (`GrowClust/tests_GrowClust/IN`). The `tests_GrowClust` GrowClust project directory structure should look like this:

```
GrowClust/ (GrowClust main directory)
├── tests_GrowClust/ (Project directory)
│   ├── IN/
│   │   ├── evlist.txt
│   │   ├── stlist.txt
│   │   ├── xcordata.txt
│   │   └── vzmodel.txt
│   ├── OUT/ (.gitkeep file, if present, can be ignored)
│   ├── TT/ (.gitkeep file, if present, can be ignored)
│   └── tests.inp
```

The files highlighted in **orange** are GrowClust input files generated during the QuakeMigrate section of the test run.

3. **Execute the test run**. In the `tests_GrowClust` project directory, initiate the GrowClust test run by running `../SRC/growclust tests.inp`. The test run should complete in under a minute due to the small event count (5). During the run, the terminal will display GrowClust's standard output (refer to Section 4.0 of the GrowClust user guide).

   Note that for this test run, due to its short time period, only 5 events are located by QuakeMigrate, and all are relocated together in GrowClust. For larger event counts, it is **recommended** to group associated events (e.g., clusters) into smaller relocation sets to enhance processing speed, improve relocation accuracy, and reflect physically consistent spatial relationships.

4. **Validate the test run**. Once the GrowClust test run has executed successfully, **copy** the generated relocated event catalog output file (`out.growclust_cat`) in the OUT subdirectory of the `tests_GrowClust` project directory (`GrowClust/tests_GrowClust/OUT`) to the QuakeMigrate `tests` project directory. Run `python GCcompare.py` in the `tests` project directory. This script compares four items: the GrowClust input files for the event list, station list, and cross-correlation data, and the GrowClust relocated event catalog output file against their respective benchmarked files, outputting an "**OK**" message if they are identical. If all four items match their benchmarked results, the script outputs "**All comparisons passed**," confirming the validation of the GrowClust test run.

6. References

Anandakrishnan, S. (2018). Rutford Ice Stream Cooperative Research Program with British Antarctic Survey [Data set]. *International Federation of Digital Seismograph Networks*. https://doi.org/10.7914/SN/5B_2018

Bacon, C. (2021). *Seismic anisotropy and microseismicity: from crustal formation to subduction termination* [Apollo - University of Cambridge Repository]. https://doi.org/10.17863/CAM.82196

Lee, I., Anandakrishnan, S., Kufner, S. K., & Alley, R. B. (2020, December). Locating Basal Microseismicity in Rutford Ice Stream, West Antarctica using QuakeMigrate, for Statistical Pattern Recognition. In *AGU Fall Meeting Abstracts* (Vol. 2020, pp. NS003-0008). https://doi.org/10.1002/essoar.10504932.1

Lee, I., Anandakrishnan, S., Alley, R., Kufner, S. K., Smith, A., & Brisbourne, A. (2021, December). Repeating Icequakes at the Grounding Line of Rutford Ice Stream, West Antarctica. In *AGU Fall Meeting Abstracts* (Vol. 2021, pp. S55B-0139). https://doi.org/10.1002/essoar.10509837.1

Lee, I., Anandakrishnan, S., Alley, R. B., Brisbourne, A., & Smith, A. (2022, December). Detailed Bed Information at the Grounding Line of Rutford Ice Stream in West Antarctica Gleaned from Comprehensive Microseismic Event Relations, and Other Sources. In *AGU Fall Meeting Abstracts* (Vol. 2022, pp. S15D-0220). https://doi.org/10.1002/essoar.10512965.1

Lee, I., Anandakrishnan, S., Alley, R., Brisbourne, A., & Smith, A. (2023, May). Event Relations and Sources of Icequakes at the Grounding Line of Rutford Ice Stream, West Antarctica. In *EGU General Assembly Conference Abstracts* (pp. EGU-1677). https://doi.org/10.5194/egusphere-egu23-1677

Lee, I. R., Anandakrishnan, S., Alley, R. B., Brisbourne, A., & Smith, A. (2024). Characterization of Sticky Spots in Rutford Ice Stream, West Antarctica with High-Granularity Microseismicity. *Authorea Preprints*. https://doi.org/10.22541/au.170629228.87287610/v1

Lee, I., Anandakrishnan, S., Alley, R. B., Brisbourne, A., & Smith, A. (2025, in review). Uncovering Basal Sliding Mechanisms at the Grounding Line of Rutford Ice Stream, West Antarctica: Big Data Analysis of Sticky Spots. Presented at AGU24 and submitted to *Authorea Preprints*.

Trugman, D. T., & Shearer, P. M. (2017). GrowClust: A hierarchical clustering algorithm for relative earthquake relocation, with application to the Spanish Springs and Sheldon, Nevada, earthquake sequences. *Seismological Research Letters*, *88*(2A), 379-391. https://doi.org/10.1785/0220160188

Winder, T. (2021). *Tectonic earthquake swarms in the Northern Volcanic Zone, Iceland* [Apollo - University of Cambridge Repository]. https://doi.org/10.17863/CAM.82505

Winder, T., Bacon, C., Smith, J., Hudson, T., Greenfield, T., & White, R. (2022). QuakeMigrate: a modular, open-source python package for automatic earthquake detection and location. *Authorea Preprints*. https://doi.org/10.1002/essoar.10505850.1

7. Acknowledgments

We thank Amanda Willet for testing the QuakeSupport prototypes and providing valuable feedback that helped identify user pain points and improve functionality. We also extend our gratitude to the developers of QuakeMigrate (particularly Conor Bacon and Tom Winder) and GrowClust (particularly Daniel Trugman) for developing these remarkable and invaluable tools upon which QuakeSupport builds, as well as for their readiness to address questions related to their software. Lastly, we thank Pennsylvania State University and the British Antarctic Survey for making the test run dataset available, which was collected with support from the U.S. National Science Foundation (NSF) award 1643961.