QuakeMigrate

Release 1.0.2

QuakeMigrate developers

May 15, 2023

CONTENTS

`QuakeMigrate` is a Python package for automatic earthquake detection and location using waveform migration and stacking.

QuakeMigrate uses a waveform migration and stacking algorithm to search for coherent seismic phase arrivals across a network of instruments. It produces, from raw data, a catalogue of earthquakes with locations, origin times, phase arrival picks, and local magnitude estimates, as well as rigorous estimates of the associated uncertainties.

The package has been built with a modular architecture, providing the potential for extension and adaptation at numerous entry points. This includes, but is not limited to:

- the calculation or import of traveltime grids
- the choice of algorithm used to identify phase arrivals (for example by kurtosis, cross-covariance analysis between multiple components, machine learning techniques and more)
- the stacking function used to combine onset functions
- the algorithm used to perform phase picking

The source code for the project is hosted on .

CHAPTER

ONE

CITATION

If you use this package in your work, please cite the following conference presentation:

Winder, T., Bacon, C.A., Smith, J.D., Hudson, T., Greenfield, T. and White, R.S., 2020. QuakeMigrate: a Modular, Open-Source Python Package for Automatic Earthquake Detection and Location. AGUFM, 2020. pp.S38-0013.

as well as the relevant version of the source code on Zenodo.

We hope to have a publication coming out soon:

Winder, T., Bacon, C.A., Smith, J.D., Hudson, T.S., Drew, J., and White, R.S. QuakeMigrate: a Python Package for Automatic Earthquake Detection and Location Using Waveform Migration and Stacking. (to be submitted to *Seismica*).

CHAPTER

TWO

CONTACT

You can contact us directly at quakemigrate.developers@gmail.com

Any additional comments/questions can be directed to:

- **Tom Winder** - tom.winder@esc.cam.ac.uk
- **Conor Bacon** - conor.bacon@cantab.net

CHAPTER THREE

LICENSE

CHAPTER

FOUR

CONTENTS:

4.1 Installation

`QuakeMigrate` is a predominantly Python package with some routines written and optimised in C. These are built and linked to QuakeMigrate at installation - if installing from source you will need to ensure that there is a suitable compiler available (see *C compilers*).

However, most users can bypass this step by installing QuakeMigrate using `pip`.

4.1.1 Supported operating systems

QuakeMigrate was developed and tested on Ubuntu 16.04/18.04, with the intention of being "platform agnostic". As of March 2023, the package has been successfully built and run on:

- Ubuntu 16.04/18.04/20.04/22.04
- Red Hat Enterprise Linux
- Debian
- Windows 10
- macOS High Sierra 10.13, Mojave 10.14, Catalina 10.15, Big Sur 11, Monterey 12 (including M1)

4.1.2 Prerequisites

QuakeMigrate supports Python 3.8 or newer (3.8/3.9/3.10/3.11). We recommend using Anaconda as a package manager and environment management system to isolate and install the specific dependencies of QuakeMigrate.

Instructions for downloading and installing Anaconda can be found here. If drive space is limited, consider using Miniconda instead, which ships with a minimal collection of useful packages.

4.1.3 Installation via *pip*

The simplest way to get a working copy of QuakeMigrate is to install it from the Python Package Index (PyPI) using `pip` (the Python package installer).

To do this you first need to set up an enivironment. We recommend creating a minimal environment initially:

```
conda create --name quakemigrate python=3.9
conda activate quakemigrate
```

All other dependencies will be handled during the installation of QuakeMigrate. After activating your environment, type the following command into terminal:

```
pip install quakemigrate
```

This will install QuakeMigrate **and** its explicit dependencies!

Note: Installing the package this way will not provide you with the examples. These can be retrieved directly from the GitHub repository (see *Testing your installation*).

The full list of dependencies is:

- matplotlib
- numpy
- obspy >= 1.3
- pandas
- pyproj >= 2.5
- scipy

Note: We are currently not pinning the version of any dependencies. We aim to keep on top of any new syntax changes etc. as new versions of these packages are released - but please submit an issue if you come across something!

If you want to explore the example notebooks, you will also need to install *ipython* and *jupyter*. This can be done with conda (making sure your environment is still activated) as:

```
conda install ipython jupyter
```

Finally, if you wish to apply QuakeMigrate in situations where the velocity is not uniform for each phase (including for the provided `Volcanotectonic_Iceland` usage example), you will need to *install a traveltime solver*.

4.1.4 Installing a traveltime solver

In addition to the explicit dependencies, QuakeMigrate includes wrapper functions that use NonLinLoc and scikit-fmm as backends for producing 1-D traveltime lookup tables (see *The traveltime lookup table*).

Users can choose to install one or both of these software packages, which will enable them to use the corresponding wrapper function. (If you already have NonLinLoc installed, you may skip this step!)

NonLinLoc

Obtaining binaries

To download, unpack, and compile NonLinLoc, **Linux** and *most* **macOS** users can use:

Note: In order to install NonLinLoc, you will need an accessible C compiler, such as *gcc* or *clang* (see *C compilers*).

Warning: The NLLoc MakeFile specifies the compiler as `gcc`. For **macOS users** this means that the system (XCode) clang compiler will be used even if you activate the relevant environment for an alternative. To change this, edit the MakeFile to specify `clang` instead of `gcc`.

```
curl http://alomax.free.fr/nlloc/soft7.00/tar/NLL7.00_src.tgz -o NLL7.00_src.tgz
tar -xzvf NLL7.00_src.tgz
cd src
mkdir bin; export MYBIN=./bin
make -R all
```

If this is not successful, **macOS** users (at least those using systems with an Intel CPU) can instead download the binaries directly:

```
curl http://alomax.free.fr/nlloc/soft7.00/tar/NLL7.00_bin_x86_64-apple-darwin14.tgz -o␣
↪NLL7.00_bin_x86_64-apple-darwin14.tgz
tar -xvzf NLL7.00_bin_x86_64-apple-darwin14.tgz
```

Alternatively, for newer versions of NonLinLoc (and instructions for installation using CMake) see the instructions on the NonLinLoc GitHub page.

Adding to the system path

Once you have successfully obtained the binary executables, we recommend you add the newly created `bin` directory to your system path. For Unix systems, this can be done by **adding the following** to your `.bashrc` file (for Linux users), or either `.zshrc` or `.bash_profile` file (for macOS - use `echo $SHELL` to check your default login shell, and therefore the appropriate file to use). This file is typically found in your home directory, ~/):

```
export PATH=/path/to/nonlinloc/bin:$PATH
```

replacing the `/path/to/nonlinloc` with the path to where you downloaded or installed NonLinLoc. Save the changes to your `.bashrc`, `.zshrc` or `.bash_profile` file, and open a new terminal window to activate the change. This will allow your shell to access the `Vel2Grid` and `Grid2Time` programs from anywhere. To test this has worked, type:

```
which Grid2Time
```

This should return `/path/to/nonlinloc/bin/Grid2Time`, as described above.

Alternatively, if you do not wish to add the NonLinLoc executables to your system path, you can explicitly specify the `nlloc_path` variable when using NonLinLoc to generate a QuakeMigrate lookup table (see *The traveltime lookup table*).

scikit-fmm

Note: In order to install scikit-fmm, you will need an accessible C++ compiler, such as *gxx* or *clangxx* (see *C compilers*).

scikit-fmm is a 3rd-party Python package which implements the fast-marching method. It can be installed using:

```
pip install scikit-fmm
```

It can also be installed along with the rest of package if installing from source (see *Other installation methods*).

4.1.5 Other installation methods

From source

Note: In order to install from source, you will need an accessible C compiler, such as *gcc* or *clang* (see *C compilers*).

Clone the repository from our GitHub (note: you will need `git` installed on your system), or alternatively download the source code directly through the GitHub web interface. Once you have a local copy, navigate to the new `QuakeMigrate` directory.

You can build a complete environment using the `environment.yml` file which can be found in the top level of the cloned repository.

```
conda env create -f environment.yml
conda activate quakemigrate
```

Finally, you can install the package (making sure your environment is activated) by running:

```
pip install .
```

You can optionally pass a `-e` argument to install the package in 'editable' mode.

If you wish to use `scikit-fmm`, you can install it here as an optional package using:

```
pip install .[fmm]
# or for zsh users:
pip install .\[fmm]
```

You should now be able to import `quakemigrate` within a Python session:

Warning: You should try this import in any directory that is *not* the root of the git repository (i.e. `QuakeMigrate/`. Here, the local `quakemigrate` directory will override the version of QuakeMigrate installed in your environment site-packages!

```
cd  # Moving out of QuakeMigrate directory - see warning above!
python
>>> import quakemigrate
>>> quakemigrate.__version__
```

If successful, this should return '1.0.2'.

Note: If you wish to use NonLinLoc as a traveltime solver, you will need to install that as detailed *above*.

conda install

We hope to link the package with the conda forge soon, after which you will be able to use the following command to install the package:

```
conda install -c conda-forge quakemigrate
```

4.1.6 Testing your installation

In order to test your installation, you will need to have cloned the GitHub repository (see *installation from source*). This will ensure you have all of the required benchmarked data (which is not included in pip/conda installs). It is also recommended that you *install NonLinLoc*, which is required for the `Volcanotectonic_Iceland` example.

To run the tests, navigate to `QuakeMigrate/tests` and run the test scripts. First, test all packages have correctly installed and you can import QuakeMigrate:

```
python test_import.py
```

This may output some warning messages about deprecations - so long as the final output line says "OK" and not "FAILED", these aren't an issue.

Note: Check if there is a message about matplotlib backends - there ought to be a suitable backend (e.g. macOSX, Qt, or Tk), but there is a chance you might not have any. If this warning is present, see *matplotlib backends*.

Next, run the examples.

Note: This requires NonLinLoc to be installed. If you have not installed (or can not install) NonLinLoc, you may edit the `run_test_examples.py` script to only run the `Icequake_Iceland` example by commenting out the relevant section.

```
python run_test_examples.py
```

This script collates and runs the scripts for each stage in the `Icequake_Iceland` and `Volcanotectonic_Iceland` examples. This process will take a number of minutes. Once this has completed successfully, run:

```
python test_benchmarks.py
```

Note: If you edited the `run_test_examples.py` script to only run the `Icequake_Iceland` example, you will also need to edit the `test_benchmarks.py` script to reflect this, otherwise the test will report as failed!

If your installation is working as intended, this should execute with no failures.

4.1.7 C compilers

In order to install and use QuakeMigrate and/or NonLinLoc & scikit-fmm from source, you will need a C compiler.

If you already have a suitable compiler (e.g. *gcc*, *MSVC*, *clang*) at the OS level, then you can proceed with installation. If this fails, then read on for tips to overcome common issues!

Checking for a C compiler

On Linux or macOS, to check if you have a C compiler, open a terminal and type:

```
which gcc
gcc --version
```

If a compiler is present, the first command will return `/usr/bin/gcc`. However, this does not guarantee it is present! The second command will confirm this.

On **Linux** the second command should output something like:

```
gcc (Ubuntu 11.3.0-1ubuntu1~22.04) 11.3.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

As long as the version is relatively recent (version 9 or later), you should be good to go. To additionally confirm that you have a C++ compiler installed, type:

```
which g++
g++ --version
```

For which you should obtain a similar result.

On **macOS** it will be obvious if the compiler is not actually installed – you will be faced with a prompt to install the Xcode Command Line Tools. You can go ahead and install this (press `Install` and wait for the process to complete). If these are already installed, the second command should output something like:

```
Configured with: --prefix=/Library/Developer/CommandLineTools/usr --with-gxx-include-
↪dir=/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/c++/4.2.1
Apple clang version 12.0.5 (clang-1205.0.22.11)
Target: x86_64-apple-darwin20.5.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

Warning: Even if *clang* is installed, it may not have all necessary libraries included. See *OpenMP on macOS*.

Note that this indicates that the system compiler is `clang`, and that the accompanying C++ compiler is also installed. These are all supplied as part of the Xcode Command Line Tools (see e.g. here for a rundown).

If you do not have a compiler, or to be sure, we provide a simple guide for *Linux*, *macOS* and *Windows* operating systems below.

Note: In order to build the (optional) dependency scikit-fmm you will need a C++ compiler (e.g. *gxx*, *MSVC*, *clangxx*). This can also be done either at the OS level, or using conda (see guidance on the conda compiler tools page, linked below).

Linux

If you have root access, the simplest route is to install *gcc* and *gxx* at system-level. You should search for the correct way to do this for your Linux Distribution. For example, on Ubuntu you would type:

```
sudo apt-get install build-essential
```

This includes *gcc*, *g++* as well as *make*. The commands will differ on other distros (CentOS, Red Hat, etc.).

Alternatively, you can install *gcc* and *g++* through conda. Make sure you have activated your environment, then type:

```
conda install -c conda-forge gcc_linux-64 gxx_linux-64
```

You can test this was successful with the same procedure detailed *above*. Once installed, you can proceed with the QuakeMigrate *installation from source*.

macOS

By default, there is no C compiler included with macOS. If you have previously installed the Xcode Command Line Tools (via the web or the App Store), the *clang* compiler will be installed. However, this may not include all necessary libraries to install QuakeMigrate (see *OpenMP on macOS*).

Whether you already have Xcode installed or not, there are two options to install what is required: the user can either install all dependencies *through conda* - noting that they will only be available in that specific environment - or using HomeBrew. We generally recommend using conda, unless the user is already familiar with brew (in which case, see *brew*).

OpenMP on macOS

The default C compiler on macOS does not include support for OpenMP. This will result in the following error during installation from source:

```
ld: library not found for -lomp
clang: error: linker command failed with exit code 1 (use -v to see invocation)
error: command '/usr/bin/clang' failed with exit code 1
```

As above, this can either be solved with *conda* or *brew*.

conda

First create and/or activate your environment:

```
conda create -n quakemigrate python=3.9  # if not already created
conda activate quakemigrate  # replace with alternative environment name if desired
```

Then use conda to install the compiler (along with the OpenMP libraries). **Note the syntax is different if your machine is running on an Apple Silicon (M1, M2, etc.) chip**:

```
conda install -c conda-forge llvm-openmp clang_osx-64 clangxx_osx-64  # Intel chip
conda install -c conda-forge llvm-openmp clang_osx-arm64 clangxx_osx-arm64  # Apple␣
↪Silicon chip (M1, M2 etc.)
```

Note: If you did not already have Xcode Command Line Tools installed, you will be prompted to install them now. Click `Install` and wait for installation to complete.

You should now open a fresh terminal, and activate your environment. To test the installation was successful, type:

```
echo $CC
$CC --version
```

This should return something like:

```
echo $CC
x86_64-apple-darwin13.4.0-clang
$CC --version
clang version 14.0.6
Target: x86_64-apple-darwin13.4.0
Thread model: posix
InstalledDir: /Users/user/miniconda3/envs/quakemigrate/bin
```

You can proceed with the QuakeMigrate *installation from source*.

brew

If brew is not already installed (check with `which brew`), follow the instructions on the HomeBrew frontpage. This will offer to install the Xcode Command Line Tools if they are not already present (press 'ENTER' or 'Y' to accept this suggestion).

You can then proceed to install the OpenMP libraries with brew:

```
brew install libomp
```

You can safely ignore the warning about explicitly adding the relevant LDFLAGS etc. - this is already handled in the QuakeMigrate `setup.py` script.

You can proceed with the QuakeMigrate *installation from source*.

Legacy: brew gcc

Alternatively, you can use the *gcc* compiler to install QuakeMigrate (and NonLinLoc). As with *clang*, we recommend installing GCC through `Homebrew`. First, check if you already have *gcc* installed, with:

```
which gcc
```

If this doesn't return anything, continue to installing *gcc*. If this returns the path to a gcc executable (e.g. */usr/bin/gcc*), then you should check the version, with:

```
gcc --version
```

If the version string includes *Apple clang*, or is a version number lower than 9, you should proceed to install with `Homebrew`:

```
brew install gcc
brew link gcc
```

Note that the `brew link` command should add `gcc` to your path, but might not succeed if a previous `gcc` install was present. To test this, type:

```
which gcc
gcc --version
```

If the linking was successful, this should point to a new gcc executable, and the version string should contain `gcc (Homebrew GCC 9.4.0) 9.4.0` or similar. If not, you will need to manually link the new `gcc` executable. To do this, find the path to your new `gcc`` installation with:

```
brew --prefix gcc
```

Then create a symlink to this executable:

```
ln -s /usr/local/bin/gcc /path/to/brew/gcc
```

Where `/path/to/brew/gcc` is the path returned by the `brew --prefix` command.

Finally, test this has worked by repeating the check from above:

```
which gcc
gcc --version
```

This should now return the `Homebrew` `gcc` version string. If not, please get in touch and we will try to help if we can...

Windows

Compilation and linking of the C extensions has been successful using the Microsoft Visual C++ (MSVC) build tools.

We strongly recommend that you download and install these tools in order to use QuakeMigrate. You can either install Visual Studio in its entirety, or just the Build Tools - available here.

You will need to restart your computer once the installation process has completed. We recommend using the anaconda command line interface (unix shell-like) to install QuakeMigrate over command prompt.

> **Warning:** QuakeMigrate has been tested and validated on Windows, but there may yet remain some unknown issues. If you encounter an issue (and/or resolve it), please submit a GitHub issue (or send an email) to let us know!

Once installed, you can proceed with the QuakeMigrate *installation from source*.

4.1.8 Notes

PROJ

There is a known issue with PROJ version 6.2.0 which causes vertical coordinates to be incorrectly transformed when using units other than metres (the PROJ default). If you encounter this issue (you will get an `ImportError` when trying to use the `lut` subpackage), you should update `pyproj`. Using conda will install an up-to-date PROJ backend, but you may need to clear your cache of downloaded packages. This can be done using:

```
conda clean --all
```

Then reinstall `pyproj`:

```
conda uninstall pyproj
conda install pyproj
```

matplotlib backends

If you receive the warning about only the `'Agg'` backend being available, you should first verify this manually. Open a Python session, and type the following commands to attempt to open an interactive plotting window:

```
python
>>> import matplotlib.pyplot as plt
>>> plt.plot([1, 2], [1, 2])
>>> plt.show()
```

If an interactive plot window appears, then this was a false alarm, and you can proceed. Else, double-verify with:

```
>>> import matplotlib
>>> matplotlib.get_backend()
```

If this returns `'Agg'`, then you definitely need to install a backend capable of drawing interactive plots. You can do this with conda (making sure your environment is activated):

```
conda intall pyqt
```

Then re-do the steps above to verify that this was successful.

4.2 Tutorials

Here we provide a few tutorials that explore each element of the package in more detail and provide code snippets the user can use in their own research. (More coming soon - please get in touch if you would like to help out!)

4.2.1 The traveltime lookup table

This tutorial will cover the basic ideas and definitions underpinning the traveltime lookup table, as well as showing how they can be created.

In order to reduce computational costs during runtime, we pre-compute traveltime lookup tables (LUTs) for each seismic phase and each station in the network to every node in a regularised 3-D grid. This grid spans the volume of interest, herein termed the coalescence volume, within which QuakeMigrate will search for events.

Defining the underlying 3-D grid

Before we can create our traveltime lookup table, we first have to define the underlying 3-D grid which spans the volume of interest.

Coordinate projections

First, we choose a pair of coordinate reference systems to represent the input coordinate space (`cproj`) and the Cartesian grid space (`gproj`). We do this using *pyproj*, which provides the Python bindings for the PROJ library. It is important to think about which projection is best suited to your particular study region. More information can be found in their documentation.

Warning: The default units of `Proj` are *metres*! It is strongly advised that you explicitly state which units you wish to use.

In this example we use the WGS84 reference ellipsoid (used as standard by the Global Positioning System) as our input space and the Lambert Conformal Conic projection to define our Cartesian grid space:

```
from pyproj import Proj

cproj = Proj(proj="longlat", ellps="WGS84", datum"=WGS84", no_defs=True)
gproj = Proj(proj="lcc", lon_0=116.75, lat_0=6.25, lat_1=5.9, lat_2=6.6,
             datum="WGS84", ellps="WGS84", units="km", no_defs=True)
```

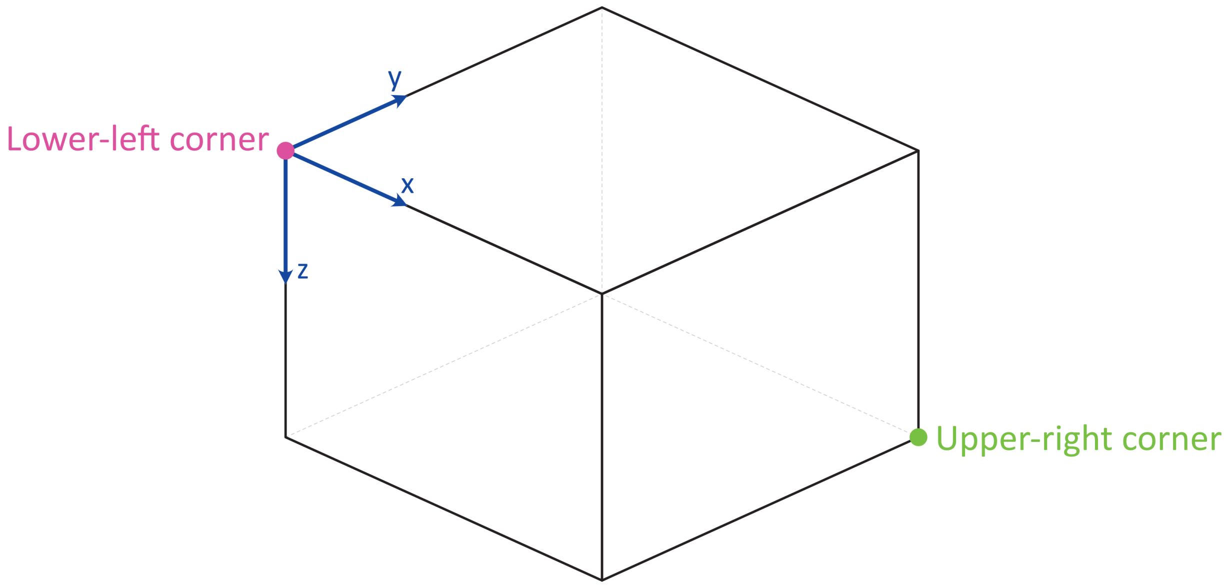
The units of the Cartesian grid space are specified as kilometres. `lon_0` and `lat_0` specify the geographic origin of the projection (which should be at roughly the centre of your grid), and `lat_1` and `lat_2` specify the two "standard parallels", which set the region in which the distortion from unit scale is minimised. We therefore recommend you choose latitudes at ~25% and 75% of the North-South extent of your grid (see *Geographical location and spatial extent*).

Note: The values used in this LCC projection are for a study region in Sabah, Borneo. Caution is advised in choosing an appropriate projection, particular if your study region is close to the poles. See the PROJ documentation for more details, and the full selection of projections available.

Note: It is possible to run QuakeMigrate with distances measured in metres if desired, as long as the user specifies this requirement when defining the grid projection and all other inputs (station elevations, grid specification, seismic phase velocities, etc) are consistently specified in metres or metres/second.

Geographical location and spatial extent

In order to geographically situate our lookup table, we choose two reference points in the input coordinate space, herein called the lower-left and upper-right corners (`ll_corner` and `ur_corner`, respectively). We work in a depth-positive frame (i.e. positive-down or left-handed coordinate system); the following schematic shows the relative positioning of the two corners:



The final piece of information required to define the grid on which we will compute traveltimes is the `node_spacing` between grid nodes along each axis (x, y and z). The LUT class will automatically find the number of nodes required to span the specified geographical region in each dimension. If the node spacing doesn't fit into the corresponding grid dimension an integer number of times, the location of the upper-right corner is shifted to accommodate an additional node.

```
ll_corner = [116.075, 5.573, -1.750]
ur_corner = [117.426, 6.925, 27.750]
node_spacing = [0.5, 0.5, 0.5]
```

Note: Any reduction in grid size can greatly reduce the computational cost of running QuakeMigrate, as runtime scales with the number of nodes - so *n^3* for an equidimensional lookup table grid of side-length *n*. The *1-D fast-marching method* for computing traveltimes requires that all stations be within the grid volume, but otherwise you are free to design the grid as you wish.

Note: The corners (`ll_corner` and `ur_corner`) are nodes - hence a grid that is 20 x 20 x 20 km, with 2 km node spacing in each dimension, will have 11 nodes in x, y, and z.

Bundling the grid specification

The grid specification needs to be bundled into a dictionary to be used as an input for the `compute_traveltimes()` function. We use here the `AttribDict` from ObsPy, which extends the standard Python *dict* data structure to also have .-style access.

```
grid_spec = AttribDict()
grid_spec.ll_corner = ll_corner
grid_spec.ur_corner = ur_corner
grid_spec.node_spacing = node_spacing
grid_spec.grid_proj = gproj
grid_spec.coord_proj = cproj
```

Computing traveltimes

Station files

In addition to the grid specification, we need to provide a list of stations for which to compute traveltime tables.

```
from quakemigrate.io import read_stations

stations = read_stations("/path/to/station_file")
```

The `read_stations()` function is a passthrough for `pandas.read_csv()`, so we can handle any delimiting characters (e.g. by specifying `read_stations("station_file", delimiter=",")`). There are four required (case-sensitive) column headers - “Name”, “Longitude”, “Latitude”, “Elevation”.

Note: Station elevations are in the positive-up/right-handed coordinate frame. An elevation of 2 would correspond to 2 (km) above sea level.

The `compute_traveltimes()` function used in the following sections returns a lookup table (a fully-populated instance of the LUT class) which can be used for `detect()`, `trigger()`, and `locate()`.

We have bundled a few methods of computing traveltimes into QuakeMigrate:

Homogeneous velocity model

Simply calculates the straight line traveltimes between stations and points in the grid. It is possible to use stations that are outside the specified span of the grid if desired. For example, if you are searching for basal icequakes you may limit the LUT grid to span a relatively small range of depths around the ice-bed interface.

```
from quakemigrate.lut import compute_traveltimes

compute_traveltimes(grid_spec, stations, method="homogeneous",
                    phases=["P", "S"], vp=5., vs=3., log=True,
                    save_file=/path/to/save_file)
```

1-D velocity models

Similarly to *station files*, 1-D velocity models are read in from an (arbitrarily delimited) textfile using `quakemigrate.io.read_vmodel()` (see below for examples). There is only 1 required (case-sensitive) column header - "Depth", which contains the depths at the top of each layer in the velocity model. Each additional column should contain the seismic velocity for each layer corresponding to a particular seismic phase, with a (case-sensitive) header, e.g. *Vp* (Note: Uppercase *V*, lowercase phase code).

Note: The units for velocities should correspond to the units used in specifying the grid projection. km -> kms^{-1}; m -> ms^{-1}.

Note: Depths are in the positive-down/left-handed coordinate frame. A depth of 5 would correspond to 5 (km) below sea level.

1-D fast-marching method

The fast-marching method calculates traveltimes by implicitly tracking the evolution of the wavefront. We use the *scikit-fmm* package as our backend to provide this functionality. It is possible to use this package to compute traveltimes from 1-D, 2-D, or 3-D velocity models, however currently we provide a utility function that computes traveltime tables from 1-D velocity models. The format of this velocity model file is specified below. See the scikit-fmm documentation and Rawlinson & Sambridge (2005) for more details.

Note: Using this method, traveltime calculation can only be performed between grid nodes: the station location is therefore taken as the closest grid node. For large node spacings this may cause a modest error in the calculated traveltimes.

Note: All stations must be situated within the grid on which traveltimes are to be computed.

```
from quakemigrate.lut import compute_traveltimes
from quakemigrate.io import read_vmodel

vmod = read_vmodel("/path/to/vmodel_file")
compute_traveltimes(grid_spec, stations, method="1dfmm", phases=["P", "S"],
                    vmod=vmod, log=True, save_file=/path/to/save_file)
```

The format of the required input velocity model file is specified *above*.

1-D NonLinLoc Grid2Time Eikonal solver

Uses the Grid2Time Eikonal solver from NonLinLoc under the hood to generate a 2D traveltime grid spanning the distance between a station and the point in the lookup table grid furthest away from its location. This slice is then "swept" through the necessary range of azimuths to populate the 3-D traveltime grid using a bilinear interpolation scheme. This method has the benefit of being able to include stations outside of the volume of interest, allowing the user to specify the minimum grid dimensions required to image the target region of seismicity.

Note: Requires the user to install the NonLinLoc software package (available from http://alomax.free.fr/nlloc/) – see the *Installation instructions* for guidance.

```
from quakemigrate.lut import compute_traveltimes
from quakemigrate.io import read_vmodel

vmod = read_vmodel("/path/to/vmodel_file")
compute_traveltimes(grid_spec, stations, method="1dnlloc",
                    phases=["P", "S"], vmod=vmod, block_model=False,
                    log=True, save_file=/path/to/save_file)
```

The format of the required input velocity model file is specified *above*.

Other formats

It is also straightforward to import traveltime lookup tables generated by other means. We have provided a parser for lookup tables stored in the NonLinLoc format (`read_nlloc()`). This code can be adapted to read any other traveltime lookup table, so long as it is stored as an array: create an instance of the `LUT` class with the correct projections and grid dimensions, then add the (C-ordered) traveltime arrays to the `LUT.traveltimes` dictionary using:

```
lut.traveltimes.setdefault(STATION, {}).update(
    {PHASE.upper(): traveltime_table})
```

where `STATION` and `PHASE` are station name and seismic phase strings, respectively (e.g. *ST01* and *P*).

Saving your LUT

If you provided a `save_file` argument to the `compute_traveltimes()` function, the LUT will already be saved. We use the `pickle` library (a Python standard library) to serialise the LUT, which essentially freezes the state of the LUT. If you did not provide a `save_file` argument, or have added 3rd-party traveltime lookup tables to the LUT, you will need to save it using:

```
lut.save("/path/to/output/lut")
```

In any case, the lookup table object is returned by the `compute_traveltimes()` function allowing you to explore the object further if you wish.

Reading in a saved LUT

When running the main stages of QuakeMigrate (`detect()`, `trigger()`, and `locate()`) it is necessary to read in the saved LUT, which can be done as:

```
from quakemigrate.io import read_lut
lut = read_lut(lut_file="/path/to/lut_file")
```

Decimating a LUT

You may wish to experiment with different node spacings, to find the optimal balance between computational requirements (runtime and memory usage), resolution, and detection sensitivity. The `LUT` object has decimation functionality built-in, e.g.:

```
lut = lut.decimate([2, 2, 2])
```

will decimate (increase the node spacing) by a factor of 2 in each of the *x*, *y* and *z* dimensions.

Note: The `lut.decimate()` function is (by default) **not** carried out in-place, so you need to explicitly set the variable *lut* equal to the returned copy. Alternatively, use `inplace=True`.

Note: Where the decimation factor *d* is not a multiple of *n-1*, where *n* is the number of grid nodes along the given axis, one or more grid nodes will be removed from the upper-right-corner direction of the LUT, which will accordingly slightly reduce the grid extent.

4.3 Source code

Explore the documentation and source code for the QuakeMigrate package.

4.3.1 quakemigrate.core

The *`quakemigrate.core`* module provides Python bindings for the library of compiled C routines that form the core of QuakeMigrate:

- Migrate onsets - This routine performs the continuous migration through time and space of the onset functions. It has been parallelised with openMP.
- Find maximum coalescence - This routine finds the continuous maximum coalescence amplitude in the 4-D coalesence volume.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

Functions

Bindings for the C library functions, migrate and find_max_coa.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

`quakemigrate.core.lib.`**`find_max_coa`**`(`*map4d*, *threads*`)`

Finds time series of the maximum coalescence/normalised coalescence in the 3-D volume, and the corresponding grid indices.

Parameters

- **`map4d`** (*numpy.ndarray* of *numpy.double*) – 4-D coalescence map, shape(nx, ny, nz, nsamples).
- **`threads`** (*`int`*) – Number of threads with which to perform the scan.

Returns

- **max_coa** (*numpy.ndarray* of *numpy.double*) – Time series of the maximum coalescence value in the 3-D volume.
- **max_norm_coa** (*numpy.ndarray* of *numpy.double*) – Times series of the maximum normalised coalescence value in the 3-D volume.
- **max_coa_idx** (*numpy.ndarray* of int) – Time series of the flattened grid indices corresponding to the maximum coalescence value in the 3-D volume.

`quakemigrate.core.lib.`**`migrate`**`(`*onsets*, *traveltimes*, *first_idx*, *last_idx*, *available*, *threads*`)`

Computes 4-D coalescence map by migrating seismic phase onset functions.

Parameters

- **`onsets`** (*numpy.ndarry* of float) – Onset functions for each seismic phase, shape(nonsets, nsamples).
- **`traveltimes`** (*numpy.ndarry* of int) – Grids of seismic phase traveltimes, converted to an integer multiple of the sampling rate, shape(nx, ny, nz, nonsets).
- **`first_idx`** (*`int`*) – Index of first sample in array from which to scan.
- **`last_idx`** (*`int`*) – Index of last sample in array up to which to scan.
- **`available`** (*`int`*) – Number of available onset functions.
- **`threads`** (*`int`*) – Number of threads with which to perform the scan.

Returns
map4d – 4-D coalescence map, shape(nx, ny, nz, nsamples).

Return type
numpy.ndarray of *numpy.double*

Raises

- **`ValueError`** – If mismatch between number of onset functions and traveltime lookup tables. Expect both to be equal to the no. stations * no. phases.
- **`ValueError`** – If the number of samples in the onset functions is less than the number of samples array is smaller than map4d[0, 0, 0, :].

4.3.2 quakemigrate.export

The *quakemigrate.export* module provides some utility functions to export the outputs of QuakeMigrate to other catalogue formats/software inputs:

- Input files for NonLinLoc
- ObsPy Catalog object
- Snuffler pick/event files for manual phase picking
- MFAST for shear-wave splitting analysis

> **Warning:** Export modules are an ongoing work in progress. The functionality of the core module *to_obspy* has been validated, but there may still be bugs elsewhere. If you are interested in using these, or wish to add additional functionality, please contact the QuakeMigrate developers at: quakemigrate.developers@gmail.com .

quakemigrate.export.to_mfast

This module provides parsers to generate SAC waveform files from an ObsPy Catalog, with headers correctly populated for MFAST.

quakemigrate.export.to_mfast.**sac_mfast**(*event*, *stations*, *output_path*, *units*, *filename=None*)

Function to create the SAC file.

Parameters

- **event** (*ObsPy.Event* object) – Contains information about the origin time and a list of associated picks.
- **stations** (*pandas.DataFrame* object) – DataFrame containing station information.
- **output_path** (*str*) – Location to save the SAC file.
- **units** (*{"km", "m"}*) – Grid projection coordinates for QM LUT (determines units of depths and uncertainties in the .event files).
- **filename** (*str, optional*) – Name of SAC file - defaults to "eventid/eventid.station.{comp}".

quakemigrate.export.to_nlloc

This module provides parsers to export an ObsPy Catalog to the NonLinLoc input file format. We prefer this to the one offered by ObsPy as it includes the additional weighting term.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

`quakemigrate.export.to_nlloc.`**`nlloc_obs`**`(`*event*, *filename*, *autopick=True*`)`

Write a NonLinLoc Phase file from an obspy Event object.

Parameters

- **`event`** (*obspy.Event* object) – Contains information on a single event.
- **`filename`** (*`str`*) – Name of NonLinLoc phase file.
- **`autopick`** (*`bool, optional`*) – Whether to read the autopicks or the modelled arrival times. Default: True (use autopicks).

quakemigrate.export.to_obspy

This module provides parsers to export the output of a QuakeMigrate run to an ObsPy Catalog.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

`quakemigrate.export.to_obspy.`**`read_quakemigrate`**`(`*run_dir*, *units*, *run_subname=''*, *local_mag_ph='S'*`)`

Reads the .event and .picks outputs, and .amps outputs if available, from a QuakeMigrate run into an obspy Catalog object.

NOTE: if a station_corrections dict was used to calculate the network-averaged local magnitude, this information will not be included in the ObsPy event object. There might therefore be a discrepancy between the mean of the StationMagnitudes and the event magnitude.

Parameters

- **`run_dir`** (*`str`*) – Path to QuakeMigrate run directory.
- **`units`** (*`{"km", "m"}`*) – Grid projection coordinates for QM LUT (determines units of depths and uncertainties in the .event files).
- **`run_subname`** (*`str, optional`*) – Run_subname string (if applicable).
- **`local_mag_ph`** (*`{"S", "P"}, optional`*) – Amplitude measurement used to calculate local magnitudes. (Default "S").

Returns
cat – Catalog containing events in the specified QuakeMigrate run directory.

Return type
obspy.Catalog object

quakemigrate.export.to_snuffler

This module provides parsers to generate input files for Snuffler, a manual phase picking interface from the Pyrocko package.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

`quakemigrate.export.to_snuffler.`**`snuffler_markers`**`(`*event*, *output_path*, *filename=None*`)`

Function to create marker files compatible with snuffler

Parameters

- **`event`** (*ObsPy.Event* object) – Contains information about the origin time and a list of associated picks.
- **`output_path`** (`str`) – Location to save the marker file.
- **`filename`** (`str, optional`) – Name of marker file - defaults to 'eventid/eventid.markers'.

`quakemigrate.export.to_snuffler.`**`snuffler_stations`**`(`*stations*, *output_path*, *filename*, *network_code=None*`)`

Function to create station files compatible with snuffler.

Parameters

- **`stations`** (*pandas.DataFrame* object) – DataFrame containing station information.
- **`output_path`** (`str`) – Location to save snuffler station file.
- **`filename`** (`str`) – Name of output station file.
- **`network_code`** (`str`) – Unique identifier for the seismic network.

4.3.3 quakemigrate.io

The *`quakemigrate.io`* module handles the various input/output operations performed by QuakeMigrate. This includes:

- Reading waveform data - The submodule data.py can handle any waveform data archive with a regular directory structure. It also provides functions for checking data quality and removing/simulating instrument reponse.
- Reading station files, velocity model files, instrument response inventories and QuakeMigrate lookup tables.
- The *`Run`* class encapsulates all i/o path information and logger configuration for a given QuakeMigrate run.
- The *`Event`* class encapsulates waveforms, coalescence information, picks and location information for a given event, and provides functionality to write ".event" files.
- Reading and writing results, including station availablity data and continuous coalescence output from detect; triggered event files from trigger, amplitude and local magnitude measurements and cut waveforms for located events.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

quakemigrate.io.amplitudes

Module to handle input/output of .amps files.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

`quakemigrate.io.amplitudes.write_amplitudes`(*run*, *amplitudes*, *event*)

Write amplitude results to a new .amps file. This includes amplitude measurements, and the magnitude estimates derived from them (with station correction terms applied, if provided).

Parameters

- **run** (*Run* object) – Light class encapsulating i/o path information for a given run.
- **amplitudes** (*pandas.DataFrame* object) – P- and S-wave amplitude measurements for each component of each station in the station file, and individual local magnitude estimates derived from them. Columns = [“epi_dist”, “z_dist”, “P_amp”, “P_freq”, “P_time”,

  ”P_avg_amp”, “P_filter_gain”, “S_amp”, “S_freq”, “S_time”, “S_avg_amp”, “S_filter_gain”, “Noise_amp”, “is_picked”, “ML”, “ML_Err”]

  Index = Trace ID (see *obspy.Trace* object property ‘id’)
- **event** (*Event* object) – Light class encapsulating waveforms, coalescence information, picks and location information for a given event.

quakemigrate.io.availability

Module to handle input/output of StationAvailability.csv files.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

`quakemigrate.io.availability.read_availability`(*run*, *starttime*, *endtime*)

Read in station availability data to a *pandas.DataFrame* from csv files split by Julian day.

Parameters

- **run** (*Run* object) – Light class encapsulating i/o path information for a given run.
- **starttime** (*obspy.UTCDateTime* object) – Timestamp from which to read the station availability.
- **endtime** (*obspy.UTCDateTime* object) – Timestamp up to which to read the station availability.

Returns
availability – Details the availability of each station for each timestep of detect.

Return type
pandas.DataFrame object

`quakemigrate.io.availability.`**`write_availability`**`(`*run*, *availability*`)`

Write out csv files (split by Julian day) containing station availability data.

Parameters

- **`run`** (*Run* object) – Light class encapsulating i/o path information for a given run.
- **`availability`** (*pandas.DataFrame* object) – Details the availability of each station for each timestep of detect.

quakemigrate.io.core

Module to handle input/output for QuakeMigrate.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

`class quakemigrate.io.core.`**`Run`**`(`*path*, *name*, *subname=''*, *stage=None*, *loglevel='info'*`)`

Bases: `object`

Light class to encapsulate i/o path information for a given run.

Parameters

- **`stage`** (*`str`*) – Specifies run stage of QuakeMigrate ("detect", "trigger", or "locate").
- **`path`** (*`str`*) – Points to the top level directory containing all input files, under which the specific run directory will be created.
- **`name`** (*`str`*) – Name of the current QuakeMigrate run.
- **`subname`** (*`str, optional`*) – Optional name of a sub-run - useful when testing different trigger parameters, for example.

`path`

Points to the top level directory containing all input files, under which the specific run directory will be created.

Type
pathlib.Path object

`name`

Name of the current QuakeMigrate run.

Type
str

`run_path`

Points to the run directory into which files will be written.

Type
pathlib.Path object

`subname`

Optional name of a sub-run - useful when testing different trigger parameters, for example.

Type
str

stage

Track which stage of QuakeMigrate is being run.

Type
{"detect", "trigger", "locate"}, optional

loglevel

Set the logging level. (Default "info")

Type
{"info", "debug"}, optional

logger(*log*)

Spins up a logger configured to output to stdout or stdout + log file.

logger(*log*)

Configures the logging feature.

Parameters
log (*bool*) – Toggle for logging. If True, will output to stdout and generate a log file.

property name

Get the run name as a formatted string.

quakemigrate.io.core.**read_lut**(*lut_file*)

Read the contents of a pickle file and restore state of the lookup table object.

Parameters
lut_file (*str*) – Path to pickle file to load.

Returns
lut – Lookup table populated with grid specification and traveltimes.

Return type
LUT object

quakemigrate.io.core.**read_response_inv**(*response_file*, *sac_pz_format=False*)

Reads response information from file, returning it as a *obspy.Inventory* object.

Parameters

- **response_file** (*str*) – Path to response file. Please see the *obspy.read_inventory()* documentation for a full list of supported file formats. This includes a dataless.seed volume, a concatenated series of RESP files or a stationXML file.
- **sac_pz_format** (*bool, optional*) – Toggle to indicate that response information is being provided in SAC Pole-Zero files. NOTE: not yet supported.

Returns
response_inv – ObsPy response inventory.

Return type
obspy.Inventory object

Raises

- **NotImplementedError** – If the user selects *sac_pz_format=True*.
- **TypeError** – If the user provides a response file that is not readable by ObsPy.

`quakemigrate.io.core.read_stations(station_file, **kwargs)`

Reads station information from file.

Parameters

- **station_file** (*str*) – Path to station file. File format (header line is REQUIRED, case sensitive, any order):

  Latitude, Longitude, Elevation (units matching LUT grid projection; either metres or kilometres; positive upwards), Name

- **kwargs** (*dict*) – Passthrough for *pandas.read_csv* kwargs.

Returns

stn_data – Columns: “Latitude”, “Longitude”, “Elevation”, “Name”

Return type

pandas.DataFrame object

Raises

StationFileHeaderException – Raised if the input file is missing required entries in the header.

`quakemigrate.io.core.read_vmodel(vmodel_file, **kwargs)`

Reads velocity model information from file.

Parameters

- **vmodel_file** (*str*) – Path to velocity model file. File format: (header line is REQUIRED, case sensitive, any order):

  ”Depth” of each layer in the model (units matching the LUT grid projection; positive-down) “V<phase>” velocity for each layer in the model, for each phase the user wishes to calculate traveltimes for (units matching the LUT grid projection). There are no required phases, and no maximum number of separate phases. E.g. “Vp”, “Vs”, “Vsh”.

- **kwargs** (*dict*) – Passthrough for *pandas.read_csv* kwargs.

Returns

vmodel_data –

Columns:

”Depth” of each layer in model (positive down) “V<phase>” velocity for each layer in model (e.g. “Vp”)

Return type

pandas.DataFrame object

Raises

VelocityModelFileHeaderException – Raised if the input file is missing required entries in the header.

`quakemigrate.io.core.stations(station_file, **kwargs)`

Alias for read_stations.

quakemigrate.io.cut_waveforms

Module to handle input/output of cut waveforms.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

`quakemigrate.io.cut_waveforms.`**`get_waveforms`**(*st*, *event*, *waveform_type*, *units*)

Get real or simulated waveforms for a Stream.

Parameters

- **st** (*obspy.Stream* object) – Stream for which to get real or simulated waveforms.
- **event** (*Event* object) – Light class encapsulating waveforms, coalescence information, picks and location information for a given event.
- **waveform_type** (*{"real", "wa"}*) – Whether to get real or Wood-Anderson simulated waveforms.
- **units** (*{"displacement", "velocity"}*) – Units to return waveforms in.

Returns
st_out – Stream of real or Wood-Anderson simulated waveforms in the requested units.

Return type
obspy.Stream object

`quakemigrate.io.cut_waveforms.`**`write_cut_waveforms`**(*run*, *event*, *file_format*, *pre_cut=0.0*, *post_cut=0.0*, *waveform_type='raw'*, *units='displacement'*)

Output cut waveform data as a waveform file – defaults to miniSEED format.

Parameters

- **run** (*Run* object) – Light class encapsulating i/o path information for a given run.
- **event** (*Event* object) – Light class encapsulating waveforms, coalescence information, picks and location information for a given event.
- **file_format** (*str, optional*) – File format to write waveform data to. Options are all file formats supported by ObsPy, including: “MSEED” (default), “SAC”, “SEGY”, “GSE2”
- **pre_cut** (*float or None, optional*) – Specify how long before the event origin time to cut the waveform data from.
- **post_cut** (*float or None, optional*) – Specify how long after the event origin time to cut the waveform data to.
- **waveform_type** (*{"raw", "real", "wa"}, optional*) – Whether to output raw, real or Wood-Anderson simulated waveforms. Default: “raw”
- **units** (*{"displacement", "velocity"}, optional*) – Whether to output displacement waveforms or velocity waveforms for real/ Wood-Anderson corrected traces. Default: displacement

Raises
AttributeError – If real or wa waveforms are requested and no response inventory has been provided.

quakemigrate.io.cut_waveforms.**write_waveforms**(*st*, *fpath*, *fstem*, *file_format*)

Output waveform data as a waveform file – defaults to miniSEED format.

Parameters

- **st** (*obspy.Stream* object) – Waveforms to be written to file.
- **fpath** (*pathlib.Path* object) – Path to output directory.
- **fstem** (*str*) – File name (without suffix).
- **file_format** (*str*) – File format to write waveform data to. Options are all file formats supported by ObsPy, including: "MSEED" (default), "SAC", "SEGY", "GSE2"

quakemigrate.io.data

Module for processing waveform files stored in a data archive.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

class quakemigrate.io.data.**Archive**(*archive_path*, *stations*, *archive_format=None*, ***kwargs*)

Bases: `object`

The Archive class handles the reading of archived waveform data.

It is capable of handling any regular archive structure. Requests to read waveform data are served up as a `WaveformData` object.

If provided, a response inventory for the archive will be stored with the waveform data for response removal, if needed (e.g. for local magnitude calculation, or to output real cut waveforms).

By default, data with mismatched sampling rates will only be decimated. If necessary, and if the user specifies *resample* = *True* and an upfactor to upsample by *upfactor* = *int* for the waveform archive, data can also be upsampled and then, if necessary, subsequently decimated to achieve the desired sampling rate.

For example, for raw input data sampled at a mix of 40, 50 and 100 Hz, to achieve a unified sampling rate of 50 Hz, the user would have to specify an upfactor of 5; 40 Hz x 5 = 200 Hz, which can then be decimated to 50 Hz - see `resample()`.

Parameters

- **archive_path** (*str*) – Location of seismic data archive: e.g.: "./DATA_ARCHIVE".
- **stations** (*pandas.DataFrame* object) – Station information. Columns ["Latitude", "Longitude", "Elevation", "Name"]. See `read_stations()`
- **archive_format** (*str, optional*) – Sets directory structure and file naming format for different archive formats. See `path_structure()`
- **kwargs** (******dict) – See Archive Attributes for details.

archive_path

Location of seismic data archive: e.g.: ./DATA_ARCHIVE.

Type
pathlib.Path object

stations

Series object containing station names.

Type
pandas.Series object

format

Directory structure and file naming format of data archive.

Type
str

read_all_stations

If True, read all stations in archive for that time period. Else, only read specified stations.

Type
bool, optional

resample

If true, perform resampling of data which cannot be decimated directly to the desired sampling rate. See `resample()`

Type
bool, optional

response_inv

ObsPy response inventory for this waveform archive, containing response information for each channel of each station of each network.

Type
obspy.Inventory object, optional

pre_filt

Pre-filter to apply during the instrument response removal. E.g. (0.03, 0.05, 30., 35.) - all in Hz. (Default None)

Type
tuple of floats

water_level

Water level to use in instrument response removal. (Default 60.)

Type
float

remove_full_response

Whether to remove the full response (including the effect of digital FIR filters) or just the instrument transform function (as defined by the PolesZeros Response Stage). Significantly slower. (Default False)

Type
bool

upfactor

Factor by which to upsample the data to enable it to be decimated to the desired sampling rate, e.g. 40Hz -> 50Hz requires upfactor = 5. See `resample()`

Type
int, optional

interpolate

If data is timestamped "off-sample" (i.e. a non-integer number of samples after midnight), whether to interpolate the data to apply the necessary correction. Default behaviour is to just alter the metadata, resulting in a sub-sample timing offset. See `shift_to_sample()`.

Type
bool, optional

path_structure(*archive_format*, *channels='*'*)

Set the directory structure and file naming format of the data archive.

read_waveform_data(*starttime*, *endtime*)

Read in waveform data between two times.

path_structure(*archive_format='YEAR/JD/STATION'*, *channels='*'*)

Define the directory structure and file naming format of the data archive.

Parameters

- **archive_format** (`str, optional`) – Directory structure and file naming format of the data archive. This may be the name of a generic archive format (e.g. SeisComp3), or one of a selection of additional formats built into QuakeMigrate.
- **channels** (`str, optional`) – Channel codes to include. E.g. channels="[B,H]H*". (Default "*")

Raises
`ArchivePathStructureError` – If the *archive_format* specified by the user is not a valid option.

read_waveform_data(*starttime*, *endtime*, *pre_pad=0.0*, *post_pad=0.0*)

Read in waveform data from the archive between two times.

Supports all formats currently supported by ObsPy, including: "MSEED", "SAC", "SEGY", "GSE2".

Optionally, read data with some pre- and post-pad, and for all stations in the archive - this will be stored in *data.raw_waveforms*, while *data.waveforms* will contain only data for selected stations between *starttime* and *endtime*.

Parameters

- **starttime** (*obspy.UTCDateTime* object) – Timestamp from which to read waveform data.
- **endtime** (*obspy.UTCDateTime* object) – Timestamp up to which to read waveform data.
- **pre_pad** (`float, optional`) – Additional pre pad of data to read. Defaults to 0.
- **post_pad** (`float, optional`) – Additional post pad of data to read. Defaults to 0.

Returns
data – Object containing the waveform data read from the archive that satisfies the query.

Return type
`WaveformData` object

Raises

- `ArchiveEmptyException` – If no data files are found in the archive for this day(s).
- `DataAvailabilityException` – If no data is found in the archive for the specified stations within the specified time window.

class quakemigrate.io.data.**WaveformData**(*starttime*, *endtime*, *stations=None*, *response_inv=None*, *water_level=60.0*, *pre_filt=None*, *remove_full_response=False*, *read_all_stations=False*, *resample=False*, *upfactor=None*, *pre_pad=0.0*, *post_pad=0.0*)

Bases: `object`

The WaveformData class encapsulates the waveform data returned by an Archive query.

It also provides a number of utility functions. These include removing instrument response and checking data availability against a flexible set of data quality criteria.

Parameters

- **starttime** (*obspy.UTCDateTime* object) – Timestamp of first sample of waveform data requested from the archive.
- **endtime** (*obspy.UTCDateTime* object) – Timestamp of last sample of waveform data requested from the archive.
- **stations** (*pandas.Series* object, optional) – Series object containing station names.
- **read_all_stations** (*bool, optional*) – If True, *raw_waveforms* contain all stations in archive for that time period. Else, only selected stations will be included.
- **resample** (*bool, optional*) – If true, allow resampling of data which cannot be decimated directly to the desired sampling rate. See `resample()` Default: False
- **upfactor** (*int, optional*) – Factor by which to upsample the data to enable it to be decimated to the desired sampling rate, e.g. 40Hz -> 50Hz requires upfactor = 5. See `resample()`
- **response_inv** (*obspy.Inventory* object, optional) – ObsPy response inventory for this waveform data, containing response information for each channel of each station of each network.
- **pre_filt** (*tuple of floats*) – Pre-filter to apply during the instrument response removal. E.g. (0.03, 0.05, 30., 35.) - all in Hz. (Default None)
- **water_level** (*float*) – Water level to use in instrument response removal. (Default 60.)
- **remove_full_response** (*bool*) – Whether to remove the full response (including the effect of digital FIR filters) or just the instrument transform function (as defined by the PolesZeros Response Stage). Significantly slower. (Default False)
- **pre_pad** (*float, optional*) – Additional pre pad of data included in *raw_waveforms*.
- **post_pad** (*float, optional*) – Additional post pad of data included in *raw_waveforms*.

starttime

Timestamp of first sample of waveform data requested from the archive.

Type

obspy.UTCDateTime object

endtime

Timestamp of last sample of waveform data requested from the archive.

Type

obspy.UTCDateTime object

stations

Series object containing station names.

Type
pandas.Series object

`read_all_stations`

If True, *raw_waveforms* contain all stations in archive for that time period. Else, only selected stations will be included.

Type
bool

`raw_waveforms`

Raw seismic data read in from the archive. This may be for all stations in the archive, or only those specified by the user. See *read_all_stations*. It may also cover the time period between *starttime* and *endtime*, or feature an additional pre- and post-pad. See *pre_pad* and *post_pad*.

Type
obspy.Stream object

`waveforms`

Seismic data read in from the archive for the specified list of stations, between *starttime* and *endtime*.

Type
obspy.Stream object

`pre_pad`

Additional pre pad of data included in *raw_waveforms*.

Type
float

`post_pad`

Additional post pad of data included in *raw_waveforms*.

Type
float

`check_availability`(*stream*, ***data_quality_params*)

Check data availability against a set of data quality criteria.

`get_wa_waveform`(*trace*, ***response_removal_params*)

Calculate the Wood-Anderson corrected waveform for a *obspy.Trace* object.

Raises
`NotImplementedError` – If the user attempts to use the get_real_waveform() method.

`check_availability`(*st*, *all_channels=False*, *n_channels=None*, *allow_gaps=False*, *full_timespan=True*, *check_sampling_rate=False*, *sampling_rate=None*, *check_start_end_times=False*)

Check waveform availability against data quality criteria.

There are a number of hard-coded checks: for whether any data is present; for whether the data is a flatline (all samples have the same value); and for whether the data contains overlaps. There are a selection of additional optional checks which can be specified according to the onset function / user preference.

Parameters

- **st** (*obspy.Stream* object) – Stream containing the waveform data to check against the availability criteria.
- **all_channels** (`bool, optional`) – Whether all supplied channels (distinguished by SEED id) need to meet the availability criteria to mark the data as 'available'.

- `n_channels` (`int, optional`) – If *all_channels=True*, this argument is required (in order to specify the number of channels expected to be present).
- `allow_gaps` (`bool, optional`) – Whether to allow gaps.
- `full_timespan` (`bool, optional`) – Whether to ensure the data covers the entire timespan requested; note that this implicitly requires that there be no gaps. Checks the number of samples in the trace, not the start and end times; for that see *check_start_end_times*.
- `check_sampling_rate` (`bool, optional`) – Check that all channels are at the desired sampling rate.
- `sampling_rate` (`float, optional`) – If *check_sampling_rate=True*, this argument is required to specify the sampling rate that the data should be at.
- `check_start_end_times` (`bool, optional`) – A stricter alternative to *full_timespan*; checks that the first and last sample of the trace have exactly the requested timestamps.

Returns

- **available** (*int*) – 0 if data doesn't meet the availability requirements; 1 if it does.
- **availability** (*dict*) – Dict of {tr_id : available} for each unique SEED ID in the input stream (available is again 0 or 1).

Raises

`TypeError` – If the user specifies *all_channels=True* but does not specify *n_channels*.

`get_real_waveform`(*tr*, *velocity=True*)

Calculate the real waveform for a Trace by removing the instrument response.

Parameters

- `tr` (*obspy.Trace* object) – Trace containing the waveform for which to remove the instrument response.
- `velocity` (`bool, optional`) – Output velocity waveform (as opposed to displacement). Default: True.

Returns

tr – Trace with instrument response removed.

Return type

obspy.Trace object

Raises

- `AttributeError` – If no response inventory has been supplied.
- *`ResponseNotFoundError`* – If the response information for a trace can't be found in the supplied response inventory.
- *`ResponseRemovalError`* – If the deconvolution of the instrument response is unsuccessful.

`get_wa_waveform`(*tr*, *velocity=False*)

Calculate simulated Wood Anderson displacement waveform for a Trace.

Parameters

- `tr` (*obspy.Trace* object) – Trace containing the waveform to be corrected to a Wood-Anderson response.

- **velocity** (*bool, optional*) – Output velocity waveform, instead of displacement. Default: False. NOTE: all attenuation functions provided within the QM local_mags module are calculated for displacement seismograms.

Returns
tr – Trace corrected to Wood-Anderson response.

Return type
obspy.Trace object

quakemigrate.io.event

Module containing the Event class, which stores information related to an individual event.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

`class quakemigrate.io.event.Event`(*marginal_window*, *triggered_event=None*)

Bases: `object`

Light class to encapsulate information about an event, including waveform data, coalescence information, origin time, locations, picks, magnitudes.

Parameters

- **marginal_window** (*float*) – Estimate of the uncertainty in the event origin time; time window over which the 4-D coalescence image is marginalised around the peak coalescence time (event origin time) to produce the 3-D coalescence map.
- **triggered_event** (*pandas.Series* object, optional) – Contains information on the candidate event identified by `trigger()`

`coa_data`

Event coalescence data computed during locate.

DT
[*numpy.ndarray* of *obspy.UTCDateTime* objects, shape(nsamples)] Timestamps for the coalescence data.

COA
[*numpy.ndarray* of floats, shape(nsamples)] Max coalescence value in the grid at each timestep.

COA_NORM
[*numpy.ndarray* of floats, shape(nsamples)] Normalised max coalescence value in the grid at each timestep.

X
[*numpy.ndarray* of floats, shape(nsamples)] X coordinate of maximum coalescence value in the grid at each timestep, in input (geographic) projection coordinates.

Y
[*numpy.ndarray* of floats, shape(nsamples)] Y coordinate of maximum coalescence value in the grid at each timestep, in input (geographic) projection coordinates.

Z
[*numpy.ndarray* of floats, shape(nsamples)] Z coordinate of maximum coalescence value in the grid at each timestep, in input (geographic) projection coordinates.

Type
pandas.DataFrame object

`data`

Light class encapsulating waveform data returned from an archive query.

Type
WaveformData object

`hypocentre`

[X, Y, Z]; Geographical coordinates of the event hypocentre (default is interpolated peak of a spline function fitted to the marginalised 3-D coalescence map).

Type
numpy.ndarray of floats

`locations`

Information on the various locations and reported uncertainties.

spline
[dict] The location of the peak coalescence value in the marginalised 3-D coalescence map, interpolated using a 3-D spline. If no spline fit was able to be made, it is just the gridded peak location.

gaussian
[dict] The location and uncertainty as determined by fitting a 3-D Gaussian to the marginalised 3-D coalescence map in a small region around the (gridded) peak coalescence location.

covariance
[dict] The location and uncertainty as determined by calculating the covariance of the coalescence values in X, Y, and Z above some percentile of the max coalescence value in the marginalised 3-D coalescence map.

Type
dict

`map4d`

4-D coalescence map generated in *locate()*.

Type
numpy.ndarray, shape(nx, ny, nz, nsamp), optional

`max_coalescence`

Dictionary containing the raw and normalised maximum coalescence values in the 3-D grid at the timestamp corresponding to the instantaneous (non-marginalised) maximum coalescence value in the 4-D grid (i.e. the event origin time).

Type
dict

`onset_data`

Light class encapsulating data generated during onset calculation.

Type
OnsetData object

`otime`

Timestamp of the instantaneous peak in the 4-D coalescence function generated in *locate()* - best estimate of the event origin time.

Type
obspy.UTCDateTime object

`trigger_info`

Useful information about the triggered event to be fed forward.

TRIG_COA
[float] The peak value of the coalescence stream used to trigger the event.

DEC_COA
[float] The coalescence value of the "raw" maximum coalsecence stream at the *trigger_time*.

DEC_COA_NORM
[float] The coalescence value of the normalised maximum coalsecence stream at the *trigger_time*.

Type
dict

`trigger_time`

The time of the peak in the continuous coalescence stream (output by detect) corresponding to the triggered event.

Type
obspy.UTCDateTime object

`uid`

A unique identifier for the event based on the event trigger time.

Type
str

`add_compute_output`(*times*, *max_coa*, *max_coa_n*, *coord*, *map4d*, *onset_data*)

Add values returned by `_compute()` to the event.

`add_covariance_location`(*xyz*, *xyz_unc*)

Add the covariance location and uncertainty to the event.

`add_gaussian_location`(*xyz*, *xyz_unc*)

Add the gaussian location and uncertainty to the event.

`add_spline_location`(*xyz*)

Add the spline-interpolated location to the event.

`add_picks`(*pick_df*)

Add phase picks to the event.

`add_local_magnitude`(*mag*, *mag_err*, *mag_r2*)

Add local magnitude to the event.

`add_waveform_data`(*data*)

Add waveform data read from the archive to the event (as a `WaveformData` object).

`in_marginal_window`(*marginal_window*)

Simple test to see if event is within the marginal window around the event origin time (time of max instantaneous coalescence value).

`mw_times`(*marginal_window*, *sampling_rate*)

Generates timestamps for data in the window around the event trigger scanned by `_compute()`; *trigger_time* +/- 2*`marginal_window`.

trim2window(*marginal_window*)

Trim the coalescence data and *map4d* to the marginal window about the event origin time.

write(*run*)

Output the event to a .event file.

get_hypocentre(*method*)

Get the event hypocentre estimate calculated by a specific method; {"gaussian", "covariance", "spline"}.

add_compute_output(*times*, *max_coa*, *max_coa_n*, *coord*, *map4d*, *onset_data*)

Append outputs of compute to the Event object. This includes time series of the maximum coalescence values in the 3-D grid at each timestep, and their locations, the full 4-D coalescence map, and the onset data generated for migration.

Parameters

- **times** (*numpy.ndarray* of *obspy.UTCDateTime* objects, shape(nsamples)) – Timestamps for the coalescence data.
- **max_coa** (*numpy.ndarray* of floats, shape(nsamples)) – Max coalescence value in the grid at each timestep.
- **max_coa_n** (*numpy.ndarray* of floats, shape(nsamples)) – Normalised max coalescence value in the grid at each timestep.
- **coord** (*numpy.ndarray* of floats, shape(nsamples, 3)) – [x, y, z] Location of maximum coalescence in the grid at each timestep, in input (geographic) projection coordinates
- **map4d** (*numpy.ndarry*, shape(nx, ny, nz, nsamp)) – 4-D coalescence map.
- **onset_data** (*OnsetData* object) – Light class encapsulating data generated during onset calculation.

add_covariance_location(*xyz*, *xyz_unc*)

Add the location determined by calculating the 3-D covariance of the marginalised coalescence map filtered above a percentile threshold.

Parameters

- **xyz** (*numpy.ndarray* of floats, shape(3)) – Geographical coordinates (lon/lat/depth) of covariance location.
- **xyz_unc** (`numpy.ndarray' of floats, shape(3)) – One sigma uncertainties on the covariance location (units determined by the LUT projection units).

add_gaussian_location(*xyz*, *xyz_unc*)

Add the location determined by fitting a 3-D Gaussian to a small window around the Gaussian smoothed maximum coalescence location.

Parameters

- **xyz** (*numpy.ndarray* of floats, shape(3)) – Geographical coordinates (lon/lat/depth) of Gaussian location.
- **xyz_unc** (`numpy.ndarray' of floats, shape(3)) – One sigma uncertainties on the Gaussian location (units determined by the LUT projection units).

add_local_magnitude(*mag*, *mag_err*, *mag_r2*)

Add outputs from local magnitude calculation to the Event object.

Parameters

- **mag** (*float*) – Network-averaged local magnitude estimate for the event.

- **mag_err** (*float*) – (Weighted) standard deviation of the magnitude estimates from amplitude measurements on individual stations/channels.
- **mag_r2** (*float*) – r-squared statistic describing the fit of the amplitude vs. distance curve predicted by the calculated mean_mag and chosen attenuation model to the measured amplitude observations. This is intended to be used to help discriminate between 'real' events, for which the predicted amplitude vs. distance curve should provide a good fit to the observations, from artefacts, which in general will not.

add_picks(*pick_df*, ***kwargs*)

Add phase picks, and a selection of picker outputs and parameters.

Parameters

- **pick_df** (*pandas.DataFrame* object) – DataFrame that contains the measured picks with columns: ["Name", "Phase", "ModelledTime", "PickTime", "PickError", "SNR"] Each row contains the phase pick from one station/phase.
- ****kwargs** – For *GaussianPicker*:

  gaussfits
  [dict of dicts] Keys "station"["phase"], each containing:

  "popt" : popt "xdata" : x_data "xdata_dt" : x_data_dt "PickValue" : max_onset "PickThreshold" : threshold

  pick_windows
  [dict] {station : phase{window}}

  window: [min_time, modelled_arrival, max_time] - all ints, referring to indices of the onset function.

add_spline_location(*xyz*)

Add the location determined by fitting a 3-D spline to a small window around the maximum coalescence location and interpolating.

Parameters
xyz (*numpy.ndarray* of floats, shape(3)) – Geographical coordinates (lon/lat/depth) of best-fitting location.

add_waveform_data(*data*)

Add waveform data in the form of a *WaveformData* object.

Parameters
data (*WaveformData* object) – Contains cut waveforms - *raw_waveforms* may be for all stations in the archive, and include an additional pre- and post-pad; *waveforms* contains data only for the stations and time period required for migration.

get_hypocentre(*method='spline'*)

Get an estimate of the event hypocentre location.

Parameters
method (*{"spline", "gaussian", "covariance"}, optional*) – Which location result to return. (Default "spline").

Returns
ev_loc – [x_coordinate, y_coordinate, z_coordinate] of event hypocentre, in the global (geographic) coordinate system.

Return type
numpy.ndarray of floats

get_loc_uncertainty(*method='gaussian'*)

Get an estimate of the hypocentre location uncertainty.

Parameters
`method` (*{"gaussian", "covariance"}, optional*) – Which location result to return. (Default "gaussian").

Returns
ev_loc_unc – [x_uncertainty, y_uncertainty, z_uncertainty] of event hypocentre; units are determined by the LUT projection units.

Return type
numpy.ndarray of floats

property hypocentre

Get an estimate of the event hypocentre location.

Parameters
`method` (*{"spline", "gaussian", "covariance"}, optional*) – Which location result to return. (Default "spline").

Returns
ev_loc – [x_coordinate, y_coordinate, z_coordinate] of event hypocentre, in the global (geographic) coordinate system.

Return type
numpy.ndarray of floats

in_marginal_window()

Test if triggered event time is within marginal window around the maximum coalescence time (origin time).

Returns
cond – Result of test.

Return type
bool

property loc_uncertainty

Get an estimate of the hypocentre location uncertainty.

Parameters
`method` (*{"gaussian", "covariance"}, optional*) – Which location result to return. (Default "gaussian").

Returns
ev_loc_unc – [x_uncertainty, y_uncertainty, z_uncertainty] of event hypocentre; units are determined by the LUT projection units.

Return type
numpy.ndarray of floats

property local_magnitude

Get the local magnitude, if it exists.

property max_coalescence

Get information related to the maximum coalescence.

mw_times(*sampling_rate*)

Utility function to generate timestamps for the time period around the trigger time for which the 4-D coalescence function is calculated in `_compute()`.

Returns
times – Timestamps for time range *trigger_time* +/- 2 * *marginal_window*.

Return type
numpy.ndarray of *obspy.UTCDateTime*, shape(nsamples)

trim2window()

Trim the coalescence data to be within the marginal window.

write(*run*, *lut*)

Write event to a .event file.

Parameters

- **run** (*Run* object) – Light class encapsulating i/o path information for a given run.
- **lut** (*LUT* object) – Contains the traveltime lookup tables for seismic phases, computed for some pre-defined velocity model.

quakemigrate.io.scanmseed

Module to handle input/output of .scanmseed files.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

class quakemigrate.io.scanmseed.**ScanmSEED**(*run*, *continuous_write*, *sampling_rate*)

Bases: `object`

Light class to encapsulate the data output by the detect stage of QuakeMigrate. This data is stored in an *obspy.Stream* object with the channels: [“COA”, “COA_N”, “X”, “Y”, “Z”].

Parameters

- **run** (*Run* object) – Light class encapsulating i/o path information for a given run.
- **continuous_write** (*bool*) – Option to continuously write the .scanmseed file output by `detect()` at the end of every time step. Default behaviour is to write in day chunks where possible.
- **sampling_rate** (*int*) – Desired sampling rate of input data; sampling rate at which to compute the coalescence function. Default: 50 Hz.

stream

Output of `detect()` stored in *obspy.Stream* object. The values have been multiplied by a factor to make use of more efficient compression. Channels: [“COA”, “COA_N”, “X”, “Y”, “Z”]

Type
obspy.Stream object

written

Tracker for whether the data appended has been written recently.

Type
bool

append(*times*, *max_coa*, *max_coa_n*, *coord*, *map4d=None*)

Append the output of `_compute()` to the coalescence stream.

empty(*starttime*, *timestep*, *i*, *msg*)

Create an set of empty arrays for a given timestep and append to the coalescence stream.

write(*write_start=None*, *write_end=None*)

Write the coalescence stream to a .scanmseed file.

append(*starttime*, *max_coa*, *max_coa_n*, *coord*, *ucf*)

Append latest timestep of `detect()` output to *obspy.Stream* object.

Multiply channels [“COA”, “COA_N”, “X”, “Y”, “Z”] by factors of [“1e5”, “1e5”, “1e6”, “1e6”, “1e3”] respectively, round and convert to int32 as this dramatically reduces memory usage, and allows the coastream data to be saved in mSEED format with STEIM2 compression. The multiplication factor is removed when the data is read back in.

Parameters

- **starttime** (*obspy.UTCDateTime* object) – Timestamp of first sample of coalescence data.
- **max_coa** (*numpy.ndarray* of floats, shape(nsamples)) – Coalescence value through time.
- **max_coa_n** (*numpy.ndarray* of floats, shape(nsamples)) – Normalised coalescence value through time.
- **coord** (*numpy.ndarray* of floats, shape(nsamples)) – Location of maximum coalescence through time in input projection space.
- **ucf** (*float*) – A conversion factor based on the lookup table grid projection. Used to ensure the same level of precision (millimetre) is retained during compression, irrespective of the units of the grid projection.

empty(*starttime*, *timestep*, *i*, *msg*, *ucf*)

Create an empty set of arrays to write to .scanmseed; used where there is no data available to run `_compute()`.

Parameters

- **starttime** (*obspy.UTCDateTime* object) – Timestamp of first sample in the given timestep.
- **timestep** (*float*) – Length (in seconds) of timestep used in detect().
- **i** (*int*) – The ith timestep of the continuous compute.
- **msg** (*str*) – Message to output to log giving details as to why this timestep is empty.
- **ucf** (*float*) – A conversion factor based on the lookup table grid projection. Used to ensure the same level of precision (millimetre) is retained during compression, irrespective of the units of the grid projection.

write(*write_start=None*, *write_end=None*)

Write a new .scanmseed file from an *obspy.Stream* object containing the data output from detect(). Note: values have been multiplied by a power of ten, rounded and converted to an int32 array so the data can be saved as mSEED with STEIM2 compression. This multiplication factor is removed when the data is read back in with read_scanmseed().

Parameters

- **write_start** (*obspy.UTCDateTime* object, optional) – Timestamp from which to write the coalescence stream to file.

- **write_end** (*obspy.UTCDateTime* object, optional) – Timestamp up to which to write the coalescence stream to file.

`quakemigrate.io.scanmseed.`**`read_scanmseed`**`(`*run*, *starttime*, *endtime*, *pad*, *ucf*`)`

Read .scanmseed files between two time stamps. Files are labelled by year and Julian day.

Parameters

- **run** (*Run* object) – Light class encapsulating i/o path information for a given run.
- **starttime** (*obspy.UTCDateTime* object) – Timestamp from which to read the coalescence stream.
- **endtime** (*obspy.UTCDateTime* object) – Timestamp up to which to read the coalescence stream.
- **pad** (`float`) – Read in "pad" seconds of additional data on either end.
- **ucf** (`float`) – A conversion factor based on the lookup table grid projection. Used to ensure the same level of precision (millimetre) is retained during compression, irrespective of the units of the grid projection.

Returns

- **data** (*pandas.DataFrame* object) – Data output by detect() – decimated scan. Columns: ["DT", "COA", "COA_N", "X", "Y", "Z"] - X/Y/Z as lon/lat/units where units is the user-selected units of the lookup table grid projection (either metres or kilometres).
- **stats** (*obspy.trace.Stats* object) – Container for additional header information for coalescence trace. Contains keys: network, station, channel, starttime, endtime,

  sampling_rate, delta, npts, calib, _format, mseed

quakemigrate.io.triggered_events

Module to handle input/output of TriggeredEvents.csv files.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

`quakemigrate.io.triggered_events.`**`read_triggered_events`**`(`*run*, ***kwargs*`)`

Read triggered events from .csv file.

Parameters

- **run** (*Run* object) – Light class encapsulating i/o path information for a given run.
- **starttime** (*obspy.UTCDateTime* object, optional) – Timestamp from which to include events in the locate scan.
- **endtime** (*obspy.UTCDateTime* object, optional) – Timestamp up to which to include events in the locate scan.
- **trigger_file** (`str, optional`) – File containing triggered events to be located.

Returns
events – Triggered events information. Columns: ["EventID", "CoaTime", "TRIG_COA", "COA_X", "COA_Y", "COA_Z", "COA", "COA_NORM"].

Return type
pandas.DataFrame object

`quakemigrate.io.triggered_events.`**`write_triggered_events`**`(`*run*, *events*, *starttime*`)`

Write triggered events to a .csv file.

Parameters

- **run** (*Run* object) – Light class encapsulating i/o path information for a given run.
- **events** (*pandas.DataFrame* object) – Triggered events information. Columns: [“EventID”, “CoaTime”, “TRIG_COA”, “COA_X”, “COA_Y”, “COA_Z”, “COA”, “COA_NORM”].
- **starttime** (*obspy.UTCDateTime* object) – Timestamp from which events have been triggered.

4.3.4 quakemigrate.lut

The `quakemigrate.lut` module handles the definition and generation of the traveltime lookup tables used in QuakeMigrate.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

`quakemigrate.lut.`**`update_lut`**`(`*old_lut_file*, *save_file*`)`

Utility function to convert old-style LUTs to new-style LUTs.

Parameters

- **old_lut_file** (`str`) – Path to lookup table file to update.
- **save_file** (`str, optional`) – Output path for updated lookup table.

quakemigrate.lut.create_lut

Module to produce traveltime lookup tables defined on a Cartesian grid.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

`quakemigrate.lut.create_lut.`**`compute_traveltimes`**`(`*grid_spec*, *stations*, *method*, *phases=['P', 'S']*, *fraction_tt=0.1*, *save_file=None*, *log=False*, ***kwargs*`)`

Top-level method for computing traveltime lookup tables.

This function takes a grid specification and is capable of computing traveltimes for an arbitrary number of phases using a variety of techniques.

Parameters

- **grid_spec** (`dict`) – Dictionary containing all of the defining parameters for the underlying 3-D grid on which the traveltimes are to be calculated. For expected keys, see `Grid3D`.
- **stations** (*pandas.DataFrame*) – DataFrame containing station information (lat/lon/elev).
- **method** (`str`) – Method to be used when computing the traveltime lookup tables.

"homogeneous" - straight line velocities.

"1dfmm" - 1-D fast-marching method using scikit-fmm.

"1dnlloc" - a 2-D traveltime grid is calculated from the 1-D velocity model using the Grid2Time eikonal solver in NonLinLoc, then swept over the 3-D grid using a bilinear interpolation scheme.

- **phases** (*list of str, optional*) – List of seismic phases for which to calculate traveltimes.
- **fraction_tt** (*float, optional*) – An estimate of the uncertainty in the velocity model as a function of a fraction of the traveltime. (Default 0.1 == 10%)
- **save_file** (*str, optional*) – Path to location to save pickled lookup table.
- **log** (*bool, optional*) – Toggle for logging - default is to only print information to stdout. If True, will also create a log file.
- **kwargs** (*dict*) – Dictionary of all keyword arguments passed to compute when called. For lists of valid arguments, please refer to the relevant method.

Returns
lut – Lookup table populated with traveltimes.

Return type
LUT object

Raises

- **ValueError** – If the specified *method* is not a valid option.
- **TypeError** – If the velocity model, or constant phase velocity, is not specified.
- **NotImplementedError** – If the *3dfmm* method is specified.

quakemigrate.lut.create_lut.**read_nlloc**(*path*, *stations*, *phases=['P', 'S']*, *fraction_tt=0.1*, *save_file=None*, *log=False*)

Read in a traveltime lookup table that is saved in the NonLinLoc format.

Parameters

- **path** (*str*) – Path to directory containing .buf and .hdr files.
- **stations** (*pandas.DataFrame*) – DataFrame containing station information (lat/lon/elev).
- **phases** (*list of str, optional*) – List of seismic phases for which to read in traveltimes.
- **fraction_tt** (*float, optional*) – An estimate of the uncertainty in the velocity model as a function of a fraction of the traveltime. (Default 0.1 == 10%)
- **save_file** (*str, optional*) – Path to location to save pickled lookup table.
- **log** (*bool, optional*) – Toggle for logging - default is to only print information to stdout. If True, will also create a log file.

Returns
lut – Lookup table populated with traveltimes from the NonLinLoc lookup table files.

Return type
LUT object

Raises
NotImplementedError – If the specified projection type is not supported.

quakemigrate.lut.lut

Module to produce traveltime lookup tables defined on a Cartesian grid.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

class quakemigrate.lut.lut.**Grid3D**(*ll_corner*, *ur_corner*, *node_spacing*, *grid_proj*, *coord_proj*)

Bases: `object`

A grid object represents a collection of points in a 3-D Cartesian space that can be used to produce regularised traveltime lookup tables that sample the continuous traveltime space for each station in a seismic network.

This class also provides the series of transformations required to move between the input projection, the grid projection and the grid index coordinate spaces.

The size and shape specifications of the grid are defined by providing the (input projection) coordinates for the lower-left and upper-right corners, a node spacing and the projections (defined using pyproj) of the input and grid spaces.

coord_proj

Input coordinate space projection.

Type
pyproj.Proj object

grid_corners

Positions of the corners of the grid in the grid coordinate space.

Type
array-like, shape (8, 3)

grid_proj

Grid space projection.

Type
pyproj.Proj object

grid_xyz

Positions of the grid nodes in the grid coordinate space. The shape of each element of the list is defined by the number of nodes in each dimension.

Type
array-like, shape (3, nx, ny, nz)

ll_corner

Location of the lower-left corner of the grid in the grid projection. Should also contain the minimum depth in the grid.

Type
array-like, [float, float, float]

node_count

Number of nodes in each dimension of the grid. This is calculated by finding the number of nodes with a given node spacing that fit between the lower-left and upper-right corners. This value is rounded up if the number of nodes returned is non-integer, to ensure the requested area is included in the grid.

Type
array-like, [int, int, int]

`node_spacing`
Distance between nodes in each dimension of the grid.

Type
array-like, [float, float, float]

`precision`
An appropriate number of decimal places for distances as a function of the node spacing and coordinate projection.

Type
list of float

`unit_conversion_factor`
A conversion factor based on the grid projection, used to convert between units of metres and kilometres.

Type
float

`unit_name`
Shorthand string for the units of the grid projection.

Type
str

`ur_corner`
Location of the upper-right corner of the grid in the grid projection. Should also contain the maximum depth in the grid.

Type
array-like, [float, float, float]

`coord2grid`(*value*, *inverse=False*, *clip=False*)
Provides a transformation between the input projection and grid coordinate spaces.

`decimate`(*df*, *inplace=False*)
Downsamples the traveltime lookup tables by some decimation factor.

`index2coord`(*value*, *inverse=False*, *unravel=False*, *clip=False*)
Provides a transformation between grid indices (can be a flattened index or an [i, j, k] position) and the input projection coordinate space.

`index2grid`(*value*, *inverse=False*, *unravel=False*)
Provides a transformation between grid indices (can be a flattened index or an [i, j, k] position) and the grid coordinate space.

`property cell_count`
Handler for deprecated attribute name 'cell_count'

`property cell_size`
Handler for deprecated attribute name 'cell_size'

`coord2grid`(*value*, *inverse=False*)
Convert between input coordinate space and grid coordinate space.

Parameters

- **value** (*array-like*) – Array (of arrays) containing the coordinate locations to be transformed. Each sub-array should describe a single point in the 3-D input space.
- **inverse** (*bool, optional*) – Reverses the direction of the transform. Default input coordinates -> grid coordinates

Returns
out – Returns an array of arrays of the transformed values.

Return type
array-like

decimate(*df*, *inplace=False*)

Resample the traveltime lookup tables by decimation by some factor.

Parameters

- **df** (*array-like [int, int, int]*) – Decimation factor in each dimension.
- **inplace** (*bool, optional*) – Perform the operation on the lookup table object or a copy.

Returns
grid – Returns a Grid3D object with decimated traveltime lookup tables.

Return type
Grid3D object (optional)

get_grid_extent(*cells=False*)

Get the minimum/maximum extent of each dimension of the grid.

The default returns the grid extent as the convex hull of the grid nodes. It is useful, for visualisation purposes, to also be able to determine the grid extent as the convex hull of a grid of cells centred on the grid nodes.

Parameters
cells (*bool, optional*) – Specifies the grid mode (nodes / cells) for which to calculate the extent.

Returns
extent – Pair of arrays representing two corners for the grid.

Return type
array-like

property grid_corners

Get the xyz positions of the nodes on the corners of the grid.

property grid_extent

Get the minimum/maximum extent of each dimension of the grid.

The default returns the grid extent as the convex hull of the grid nodes. It is useful, for visualisation purposes, to also be able to determine the grid extent as the convex hull of a grid of cells centred on the grid nodes.

Parameters
cells (*bool, optional*) – Specifies the grid mode (nodes / cells) for which to calculate the extent.

Returns
extent – Pair of arrays representing two corners for the grid.

Return type
array-like

property grid_xyz

Get the xyz positions of all of the nodes in the grid.

index2coord(*value*, *inverse=False*, *unravel=False*)

Convert between grid indices and input coordinate space.

This is a utility function that wraps the other two defined transforms.

Parameters

- **value** (*array-like*) – Array (of arrays) containing the grid indices (grid coordinates) to be transformed. Can be an array of flattened indices.
- **inverse** (*bool, optional*) – Reverses the direction of the transform. Default indices -> input projection coordinates.
- **unravel** (*bool, optional*) – Convert a flat index or array of flat indices into a tuple of coordinate arrays.

Returns
out – Returns an array of arrays of the transformed values.

Return type
array-like

index2grid(*value*, *inverse=False*, *unravel=False*)

Convert between grid indices and grid coordinate space.

Parameters

- **value** (*array-like*) – Array (of arrays) containing the grid indices (grid coordinates) to be transformed. Can be an array of flattened indices.
- **inverse** (*bool, optionale*) – Reverses the direction of the transform. Default indices -> grid coordinates.
- **unravel** (*bool, optional*) – Convert a flat index or array of flat indices into a tuple of coordinate arrays.

Returns
out – Returns an array of arrays of the transformed values.

Return type
array-like

property node_count

Get and set the number of nodes in each dimension of the grid.

property node_spacing

Get and set the spacing of nodes in each dimension of the grid.

property precision

Get appropriate number of decimal places as a function of the node spacing and coordinate projection.

property unit_conversion_factor

Expose unit_conversion_factor of the grid projection.

property unit_name

Expose unit_name of the grid_projection and return shorthand.

class `quakemigrate.lut.lut.LUT`(*fraction_tt=0.1*, *lut_file=None*, ***grid_spec*)

Bases: *Grid3D*

A lookup table (LUT) object is a simple data structure that is used to store a series of regularised tables that, for each seismic station in a network, store the traveltimes to every point in the 3-D volume. These lookup tables are pre-computed to efficiently carry out the migration.

This class provides utility functions that can be used to serve up or query these pre-computed lookup tables.

This object is-a *Grid3D*.

fraction_tt

An estimate of the uncertainty in the velocity model as a function of a fraction of the traveltime. (Default 0.1 == 10%)

Type

float

max_traveltime

The maximum traveltime between any station and a point in the grid.

Type

float

phases

Seismic phases for which there are traveltime lookup tables available.

Type

list of str

stations_xyz

Positions of the stations in the grid coordinate space.

Type

array-like, shape (n, 3)

traveltimes

A dictionary containing the traveltime lookup tables. The structure of this dictionary is:

traveltimes

- **"<Station1-ID>"**
  - "<PHASE>"
  - "<PHASE>"
- **"<Station2-ID"**
  - "<PHASE>"
  - "<PHASE>"

etc

Type

dict

velocity_model

Contains the input velocity model specification.

Type

pandas.DataFrame object

serve_traveltimes(*sampling_rate*)

Serve up the traveltime lookup tables.

traveltime_to(*phase*, *ijk*)

Query traveltimes to a grid location (in terms of indices) for a particular phase.

save(*filename*)

Dumps the current state of the lookup table object to a pickle file.

load(*filename*)

Restore the state of the saved LUT object from a pickle file.

plot(*fig*, *gs*, *slices=None*, *hypocentre=None*, *station_clr='k'*)

Plot cross-sections of the LUT with station locations. Optionally plot slices through a coalescence image.

load(*filename*)

Read the contents of a pickle file and restore state of the lookup table object.

Parameters

filename (*str*) – Path to pickle file to load.

property max_extent

Get the minimum/maximum geographical extent of the stations/grid.

property max_traveltime

Get the maximum traveltime from any station across the grid.

plot(*fig*, *gs*, *slices=None*, *hypocentre=None*, *station_clr='k'*, *station_list=None*)

Plot the lookup table for a particular station.

Parameters

- **fig** (*matplotlib.Figure* object) – Canvas on which LUT is plotted.
- **gs** (*tuple(int, int)*) – Grid specification for the plot.
- **slices** (*array of arrays, optional*) – Slices through a coalescence image to plot.
- **hypocentre** (*array of floats*) – Event hypocentre - will add cross-hair to plot.
- **station_clr** (*str, optional*) – Plot the stations with a particular colour.
- **station_list** (*list-like of str, optional*) – List of stations from the LUT to plot - useful if only a subset have been selected to be used in e.g. locate.

save(*filename*)

Dump the current state of the lookup table object to a pickle file.

Parameters

filename (*str*) – Path to location to save pickled lookup table.

serve_traveltimes(*sampling_rate*, *availability=None*)

Serve up the traveltime lookup tables.

The traveltimes are multiplied by the scan sampling rate and converted to integers.

Parameters

- **sampling_rate** (*int*) – Samples per second used in the scan run.
- **availability** (*dict, optional*) – Dict of stations and phases for which to serve traveltime lookup tables: keys "station_phase".

Returns
traveltimes – Stacked traveltime lookup tables for all seismic phases, stacked along the station axis, with shape(nx, ny, nz, nstations)

Return type
numpy.ndarray of *numpy.int*

`property station_extent`
Get the minimum/maximum extent of the seismic network.

`property stations_xyz`
Get station locations in the grid space [X, Y, Z].

`traveltime_to`(*phase*, *ijk*, *station=None*)
Serve up the traveltimes to a grid location for a particular phase.

Parameters

- **phase** (*str*) – The seismic phase to lookup.
- **ijk** (*array-like*) – Grid indices for which to serve traveltime.
- **station** (*str or list-like (of str), optional*) – Station or stations for which to serve traveltimes. Can be str (for a single station) or list / *pandas.Series* object for multiple.

Returns
traveltimes – Array of interpolated traveltimes to the requested grid position.

Return type
array-like

4.3.5 quakemigrate.plot

The *quakemigrate.plot* module provides methods for the generation of figures in QuakeMigrate, including:

- Event summaries
- Phase pick summaries
- Triggered event summaries
- Amplitude / local magnitude summaries

quakemigrate.plot.event

Module containing methods to generate event summaries and videos.

`quakemigrate.plot.event.`**`event_summary`**(*run*, *event*, *marginalised_coa_map*, *lut*, *xy_files=None*)

Plots an event summary illustrating the locate results: slices through the marginalised coalescence map with the best location estimate (peak of interpolated spline fitted to 3-D coalescence map) and uncertainty ellipse from gaussian fit to gaussian-smoothed 3-D coalescence map. Plus a waveform gather of the pre-processed waveform data used to calculate the onset functions (sorted by distance from the event), and a plot of the maximum value of the 4-D coalescence function through time.

Parameters

- **run** (*Run* object) – Light class encapsulating i/o path information for a given run.
- **event** (*Event* object) – Light class encapsulating waveforms, coalescence information, picks and location information for a given event.
- **marginalised_coa_map** (*numpy.ndarray* of *numpy.double*) – Marginalised 3-D coalescence map, shape(nx, ny, nz).
- **lut** (*LUT* object) – Contains the traveltime lookup tables for seismic phases, computed for some pre-defined velocity model.
- **xy_files** (*str, optional*) – Path to comma-separated value file (.csv) containing a series of coordinate files to plot. Columns: ["File", "Color", "Linewidth", "Linestyle"], where "File" is the absolute path to the file containing the coordinates to be plotted. E.g: "/home/user/volcano_outlines.csv,black,0.5,-". Each .csv coordinate file should contain coordinates only, with columns: ["Longitude", "Latitude"]. E.g.: "-17.5,64.8". Lines pre-pended with # will be treated as a comment - this can be used to include references. See the Volcanotectonic_Iceland example XY_files for a template.

Note: Do not include a header line in either file.

quakemigrate.plot.phase_picks

Module to produce a summary plot for the phase picking.

`quakemigrate.plot.phase_picks.`**`pick_summary`**(*event*, *station*, *waveforms*, *picks*, *onsets*, *ttimes*, *windows*)

Plot a figure showing the pre-processed traces for each data component and the onset functions calculated from them for each phase. The search window to make a phase pick is displayed, along with the dynamic pick threshold, the phase pick time and its uncertainty (if made) and the Gaussian fit to the onset function.

Parameters

- **event** (*Event* object) – Light class encapsulating waveforms, coalescence information, picks and location information for a given event.

- **station** (*str*) – Station code.
- **waveforms** (*obspy.Stream* object) – Filtered seismic data used to calculate the onset functions.
- **picks** (*pandas.DataFrame* object) – Phase pick times with columns ["Name", "Phase", "ModelledTime", "PickTime", "PickError", "SNR"] Each row contains the phase pick from one station/phase.
- **onsets** (dict of {str: *numpy.ndarray*}) – Keys are phases. Onset functions for each seismic phase.
- **ttimes** (*list of float*) – Modelled traveltimes from the event hypocentre to the station for each phase to be plotted.
- **windows** (*dict of list, [int, int, int]*) – Keys are phase. Indices specifying the window within which the pick was made [start, modelled_arrival, end].

Returns
fig – Figure showing phase picking information.

Return type
matplotlib.Figure object

quakemigrate.plot.trigger

Module to plot the triggered events on a decimated grid.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

quakemigrate.plot.trigger.**trigger_summary**(*events*, *starttime*, *endtime*, *run*, *marginal_window*, *min_event_interval*, *detection_threshold*, *normalise_coalescence*, *lut*, *data*, *region*, *discarded_events*, *interactive*, *xy_files=None*, *plot_all_stns=True*)

Plots the data from a .scanmseed file with annotations illustrating the trigger results: event triggers and marginal windows on the coalescence traces, and map and cross section view of the gridded triggered earthquake locations.

Parameters

- **events** (*pandas.DataFrame*) – Triggered events information, columns: ["EventID", "CoaTime", "TRIG_COA", "COA_X", "COA_Y", "COA_Z", "MinTime", "MaxTime", "COA", "COA_NORM"].
- **starttime** (*obspy.UTCDateTime*) – Start time of trigger run.
- **endtime** (*obspy.UTCDateTime*) – End time of trigger run.
- **run** (*Run* object) – Light class encapsulating i/o path information for a given run.
- **marginal_window** (*float*) – Time window over which to marginalise the 4D coalescence function.
- **min_event_interval** (*float*) – Minimum time interval between triggered events.
- **detection_threshold** (*array-like*) – Coalescence value above which to trigger events.
- **normalise_coalescence** (*bool*) – If True, use coalescence normalised by the average coalescence value in the 3-D grid at each timestep.

- **lut** (*LUT* object) – Contains the traveltime lookup tables for the selected seismic phases, computed for some pre-defined velocity model.
- **data** (*pandas.DataFrame*) – Data output by `detect()` – continuous scan, columns: [“COA”, “COA_N”, “X”, “Y”, “Z”]
- **region** (*list*) – Geographical region within which to trigger earthquakes; events located outside this region will be discarded.
- **discarded_events** (*pandas.DataFrame*) – Discarded triggered events information, columns: [“EventID”, “CoaTime”, “TRIG_COA”, “COA_X”, “COA_Y”, “COA_Z”, “MinTime”, “MaxTime”, “COA”, “COA_NORM”].
- **interactive** (*bool*) – Toggles whether to produce an interactive plot.
- **xy_files** (*str, optional*) – Path to comma-separated value file (.csv) containing a series of coordinate files to plot. Columns: [“File”, “Color”, “Linewidth”, “Linestyle”], where “File” is the absolute path to the file containing the coordinates to be plotted. E.g: “/home/user/volcano_outlines.csv,black,0.5,-“. Each .csv coordinate file should contain coordinates only, with columns: [“Longitude”, “Latitude”]. E.g.: “-17.5,64.8”. Lines pre-pended with # will be treated as a comment - this can be used to include references. See the Volcanotectonic_Iceland example XY_files for a template.

  Note: Do not include a header line in either file.

- **plot_all_stns** (*bool, optional*) – If true, plot all stations used for detect. Otherwise, only plot stations which for which some data was available during the trigger time window. NOTE: if no station availability data is found, all stations in the LUT will be plotted. (Default, True)

4.3.6 quakemigrate.signal

The `quakemigrate.signal` module handles the core of the QuakeMigrate methods. This includes:

- Generation of onset functions from raw data.
- Picking of waveforms from onset functions.
- Migration of onsets for detect() and locate().
- Measurement of phase amplitudes and calculation of local earthquake magnitudes.

Subpackages

quakemigrate.signal.onsets

The `quakemigrate.onsets` module handles the generation of Onset functions. The default method uses the ratio between the short-term and long-term averages of the signal amplitude.

Feel free to contribute more Onset function options!

copyright
: 2020–2023, QuakeMigrate developers.

license
: GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

quakemigrate.signal.onsets.base

A simple abstract base class with method stubs to enable users to extend QuakeMigrate with custom onset functions that remain compatible with the core of the package.

Also contains a light class to encapsulate the data generated by the onset function, to be used for migration or phase picking.

copyright
: 2020–2023, QuakeMigrate developers.

license
: GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

class `quakemigrate.signal.onsets.base.Onset(**kwargs)`

Bases: `ABC`

QuakeMigrate default onset function class.

`sampling_rate`

Desired sampling rate for input data; sampling rate at which the onset functions will be computed.

Type
: int

`pre_pad`

Option to override the default pre-pad duration of data to read before computing 4-D coalescence in detect() and locate().

Type
: float, optional

`post_pad`

Option to override the default post-pad duration of data to read before computing 4-D coalescence in detect() and locate().

Type
: float

`calculate_onsets()`

Generate onset functions that represent seismic phase arrivals

`pad(timespan)`

Create appropriate padding to include the taper.

abstract calculate_onsets()

Method stub for calculation of onset functions.

gaussian_halfwidth(*phase*)

Method stub for Gaussian half-width estimate.

pad(*timespan*)

Determine the number of samples needed to pre- and post-pad the timespan.

Parameters

timespan (`float`) – The time window to pad.

Returns

- **pre_pad** (*float*) – Option to override the default pre-pad duration of data to read before computing 4-D coalescence in detect() and locate().
- **post_pad** (*float*) – Option to override the default post-pad duration of data to read before computing 4-D coalescence in detect() and locate().

abstract property post_pad

Get property stub for pre_pad.

abstract property pre_pad

Get property stub for pre_pad.

class quakemigrate.signal.onsets.base.**OnsetData**(*onsets*, *phases*, *channel_maps*, *filtered_waveforms*, *availability*, *starttime*, *endtime*, *sampling_rate*)

Bases: `object`

The OnsetData class encapsulates the onset functions calculated by transforming seismic data using the chosen onset detection algorithm (characteristic function).

This includes a dictionary describing which onset functions are available for each station and phase, and the intermediary filtered or otherwise pre-processed waveform data used to calculate the onset function.

Parameters

- **onsets** (`dict of dicts`) – Keys "station", each of which contains keys for each phase, e.g. "P" and "S". {"station": {"P": *p_onset*, "S": *s_onset*}}. Onset functions are calculated by transforming the raw seismic data using some characteristic function designed to highlight phase arrivals.
- **phases** (`list of str`) – Phases for which onsets have been calculated. (e.g. ["P", "S"])
- **channel_maps** (`dict of str`) – Data component maps - keys are phases. (e.g. {"P": "Z"})
- **filtered_waveforms** (*obspy.Stream* object) – Filtered and/or resampled and otherwise processed seismic data generated during onset function generation. Only contains waveforms that have passed the quality control criteria, at a unified sampling rate - see *sampling_rate*.
- **availability** (`dict`) – Dictionary with keys "station_phase", containing 1's or 0's corresponding to whether an onset function is available for that station and phase - determined by data availability and quality checks.
- **starttime** (*obspy.UTCDateTime* object) – Start time of onset functions.
- **endtime** (*obspy.UTCDateTime* object) – End time of onset functions.
- **sampling_rate** (`int`) – Sampling rate of filtered waveforms and onset functions.

quakemigrate.signal.onsets.stalta

The default onset function class - performs some pre-processing on raw seismic data and calculates STA/LTA onset (characteristic) function.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

`class quakemigrate.signal.onsets.stalta.CentredSTALTAOnset(**kwargs)`

Bases: `STALTAOnset`

QuakeMigrate default onset function class - uses a centred STA/LTA onset.

NOTE: THIS CLASS HAS BEEN DEPRECATED AND WILL BE REMOVED IN A FUTURE UPDATE

`class quakemigrate.signal.onsets.stalta.ClassicSTALTAOnset(**kwargs)`

Bases: `STALTAOnset`

QuakeMigrate default onset function class - uses a classic STA/LTA onset.

NOTE: THIS CLASS HAS BEEN DEPRECATED AND WILL BE REMOVED IN A FUTURE UPDATE

`class quakemigrate.signal.onsets.stalta.STALTAOnset(**kwargs)`

Bases: `Onset`

QuakeMigrate default onset function class - uses the Short-Term Average to Long-Term Average ratio of the signal energy amplitude.

Raw seismic data will be pre-processed, including re-sampling if necessary to reach the specified uniform sampling raate, checked against a user- specified set of data quality criteria, then used to calculate onset functions for each phase (using seismic channels as specified in *channel_maps*) by computing the STA/LTA of s^2.

phases
Which phases to calculate onset functions for. This will determine which phases are used for migration/picking. The selected phases must be present in the travel-time look-up table to be used for these purposes.

Type
list of str

bandpass_filters
Butterworth bandpass filter specification - keys are phases. [lowpass (Hz), highpass (Hz), corners*] *NOTE: two-pass filter effectively doubles the number of corners.

Type
dict of [float, float, int]

channel_maps
Data component maps - keys are phases. These are passed into the `ObsPy.stream.select()` method.

Type
dict of str

channel_counts
Number of channels to be used to calculate the onset function for each phase. Keys are phases.

Type
dict of int

`sta_lta_windows`

Short-term average (STA) and Long-term average (LTA) window lengths - keys are phases. [STA, LTA] (both in seconds)

Type
dict of [float, float]

`all_channels`

If True, only calculate an onset function when all requested channels meet the availability criteria. Otherwise, if at least one channel is available (e.g. just the N component for the S phase) the onset function will be calculated from that/those.

Type
bool

`allow_gaps`

If True, allow gappy data to be used to calculate the onset function. Gappy data will be detrended, tapered and filtered, then gaps padded with zeros. This should help mitigate the expected spikes as data goes on- and off-line, but will not eliminate it. Onset functions for periods with no data will be filled with ~ zeros (smallest possible float, to avoid divide by zero errors). NOTE: This feature is experimental and still under development.

Type
bool

`full_timespan`

If False, allow data which doesn't cover the full timespan requested to be used for onset function calculation. This is a subtly different test to *allow_gaps*; data must be continuous within the timespan, but may not span the whole period. Data will be treated as described in *allow_gaps*. NOTE: This feature is experimental and still under development.

Type
bool

`position`

Compute centred STA/LTA (STA window is preceded by LTA window; value is assigned to end of LTA window / start of STA window) or classic STA/LTA (STA window is within LTA window; value is assigned to end of STA & LTA windows). Default: "classic".

Centred gives less phase-shifted (late) onset function, and is closer to a Gaussian approximation, but is far more sensitive to data with sharp offsets due to instrument failures. We recommend using classic for detect() and centred for locate() if your data quality allows it. This is the default behaviour; override by setting this variable.

Type
str, optional

`sampling_rate`

Desired sampling rate for input data, in Hz; sampling rate at which the onset functions will be computed.

Type
int

`calculate_onsets()`

Generate onset functions that represent seismic phase arrivals.

`gaussian_halfwidth()`

Phase-appropriate Gaussian half-width estimate based on the short-term average window length.

calculate_onsets(*data*, *log=True*, *timespan=None*)

Calculate onset functions for the requested stations and phases.

Returns a stacked array of onset functions for the requested phases, and an `OnsetData` object containing all outputs from the onset function calculation: a dict of the onset functions, a Stream containing the pre-processed input waveforms, and a dict of availability info describing which of the requested onset functions could be calculated (depending on data availability and data quality checks).

Parameters

- **data** (`WaveformData` object) – Light class encapsulating data returned by an archive query.
- **log** (*bool*) – Calculate log(onset) if True, otherwise calculate the raw onset.
- **timespan** (*float or None, optional*) – If the timespan for which the onsets are being generated is provided, this will be used to calculate the tapered window of data at the start and end of the onset function which should be disregarded. This is necessary to accurately set the pick threshold in GaussianPicker, for example.

Returns

- **onsets** (*numpy.ndarray* of float) – Stacked onset functions served up for migration, shape(nonsets, nsamples).
- **onset_data** (`OnsetData` object) – Light class encapsulating data generated during onset calculation.

gaussian_halfwidth(*phase*)

Return the phase-appropriate Gaussian half-width estimate based on the short-term average window length.

Parameters
phase (*{'P', 'S'}*) – Seismic phase for which to serve the estimate.

property onset_centred

Handle deprecated onset_centred kwarg / attribute

property p_bp_filter

Handle deprecated p_bp_filter kwarg / attribute

property p_onset_win

Handle deprecated p_onset_win kwarg / attribute

property post_pad

Post-pad is determined as a function of the max traveltime in the grid and the onset windows

property pre_pad

Pre-pad is determined as a function of the onset windows

property s_bp_filter

Handle deprecated s_bp_filter kwarg / attribute

property s_onset_win

Handle deprecated s_onset_win kwarg / attribute

quakemigrate.signal.onsets.stalta.**pre_process**(*stream*, *sampling_rate*, *resample*, *upfactor*, *filter_*, *starttime*, *endtime*)

Resample raw seismic data, detrend and apply cosine taper and zero phase-shift Butterworth band-pass filter; all carried out using the built-in obspy functions.

By default, data with mismatched sampling rates will only be decimated. If necessary, and if the user has specified *resample = True* and an *upfactor* to upsample by *upfactor = int* for the waveform archive, data can also be upsampled and then, if necessary, subsequently decimated to achieve the desired sampling rate.

For example, for raw input data sampled at a mix of 40, 50 and 100 Hz, to achieve a unified sampling rate of 50 Hz, the user would have to specify an *upfactor* of 5; 40 Hz x 5 = 200 Hz, which can then be decimated to 50 Hz.

NOTE: data will be detrended and a cosine taper applied before decimation, in order to avoid edge effects when applying the lowpass filter. See `resample()`

Parameters

- **stream** (*obspy.Stream* object) – Waveform data to be pre-processed.
- **sampling_rate** (*int*) – Desired sampling rate for data to be used to calculate onset. This will be achieved by resampling the raw waveform data. By default, only decimation will be applied, but data can also be upsampled if specified by the user when creating the `Archive` object.
- **resample** (*bool, optional*) – If true, perform resampling of data which cannot be decimated directly to the desired sampling rate. See `resample()`
- **upfactor** (*int, optional*) – Factor by which to upsample the data to enable it to be decimated to the desired sampling rate, e.g. 40Hz -> 50Hz requires upfactor = 5. See `resample()`
- **filter** (*list*) – Filter specifications, as [lowcut (Hz), highcut (Hz), order]. NOTE - two-pass filter effectively doubles the number of corners (order).

Returns

filtered_waveforms – Pre-processed seismic data.

Return type

obspy.Stream object

Raises

`NyquistException` – If the high-cut filter specified for the bandpass filter is higher than the Nyquist frequency of the *sampling_rate*.

`quakemigrate.signal.onsets.stalta.`**`sta_lta_centred`**(*signal*, *nsta*, *nlta*)

Calculates the ratio of the average of a^2 in a short-term (signal) window to a preceding long-term (noise) window. STA/LTA value is assigned to the end of the LTA /one sample before the start of the STA.

Parameters

- **signal** (*array-like*) – Signal array
- **nsta** (*int*) – Number of samples in short-term window
- **nlta** (*int*) – Number of samples in long-term window

Returns

sta / lta – Ratio of a^2 in a short term average window to a preceding long term average window. STA/LTA value is assigned to end of LTA window / one sample before the start of STA window – “centred”.

Return type

array-like

quakemigrate.signal.pickers

The `quakemigrate.pickers` module handles the picking of seismic phases. The default method makes the phase picks by fitting a 1-D Gaussian to the Onset function.

Feel free to contribute more phase picking methods!

copyright
: 2020–2023, QuakeMigrate developers.

license
: GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

quakemigrate.signal.pickers.base

A simple abstract base class with method stubs enabling simple modification of QuakeMigrate to use custom phase picking methods that remain compatible with the core of the package.

copyright
: 2020–2023, QuakeMigrate developers.

license
: GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

`class quakemigrate.signal.pickers.base.`**`PhasePicker`**`(`*`**kwargs`*`)`

Bases: `ABC`

Abstract base class providing a simple way of modifying the default picking function in QuakeMigrate.

`plot_picks`

Toggle plotting of phase picks.

Type
: bool

`pick_phases``()`

Abstract method stub providing interface with QuakeMigrate scan.

`write``(`*event_uid*, *phase_picks*, *output*`)`

Outputs phase picks to file.

`plot``()`

Method stub for phase pick plotting.

`abstract` **`pick_phases`**`()`

Method stub for phase picking.

`plot``()`

Method stub for phase pick plotting.

`write``(`*run*, *event_uid*, *phase_picks*`)`

Write phase picks to a new .picks file.

Parameters

- **event_uid** (*str*) – Unique identifier for the event.
- **phase_picks** (*pandas DataFrame object*) –

Phase pick times with columns: ["Name", "Phase",
"ModelledTime", "PickTime", "PickError", "SNR"]

Each row contains the phase pick from one station/phase.

- **output** (*QuakeMigrate input/output control object*) – Contains useful methods controlling output for the scan.

quakemigrate.signal.pickers.gaussian

The default seismic phase picking class - fits a 1-D Gaussian to the calculated onset functions.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

class quakemigrate.signal.pickers.gaussian.**GaussianPicker**(*onset=None*, ***kwargs*)

Bases: *PhasePicker*

This class details the default method of making phase picks shipped with QuakeMigrate, namely fitting a 1-D Gaussian function to the onset function for each station and phase.

phase_picks

"GAU_P"
[array-like] Numpy array stack of Gaussian pick info (each as a dict) for P phase

"GAU_S"
[array-like] Numpy array stack of Gaussian pick info (each as a dict) for S phase

Type
dict

threshold_method
Which method to use to calculate the pick threshold; a percentile of the data outside the pick windows (e.g. 0.99 = 99th percentile) or a multiple of the Median Absolute Deviation of the signal outside the pick windows. Default uses the MAD method.

Type
{"MAD", "percentile"}

percentile_pick_threshold
Picks will only be made if the onset function exceeds this percentile of the noise level (amplitude of onset function outside pick windows). (Default: 1.0)

Type
float, optional

mad_pick_threshold
Picks will only be made if the onset function exceeds its median value plus this multiple of the MAD (calculated from the onset data outside the pick windows). (Default: 8)

Type
float, optional

plot_picks
Toggle plotting of phase picks.

Type
bool

pick_phases(*event*, *lut*, *run*)
Picks phase arrival times for located events by fitting a 1-D Gaussian function to the P and/or S onset functions

DEFAULT_GAUSSIAN_FIT = {'PickValue': -1, 'popt': 0, 'xdata': 0, 'xdata_dt': 0}

property fraction_tt
Handler for deprecated attribute 'fraction_tt'

pick_phases(*event*, *lut*, *run*)
Picks phase arrival times for located events.

Parameters

- **event** (*Event* object) – Light class encapsulating waveforms, coalescence information and location information for a given event.
- **lut** (*LUT* object) – Contains the traveltime lookup tables for seismic phases, computed for some pre-defined velocity model.
- **run** (*Run* object) – Light class encapsulating i/o path information for a given run.

Returns

- **event** (*Event* object) – Event object provided to pick_phases(), but now with phase picks!
- **picks** (*pandas.DataFrame*) – DataFrame that contains the measured picks with columns: ["Name", "Phase", "ModelledTime", "PickTime", "PickError", "SNR"] Each row contains the phase pick from one station/phase.

property pick_threshold
Handler for deprecated attribute 'pick_threshold'

plot(*event*, *station*, *onset_data*, *picks_df*, *traveltimes*, *run*)
Plot figure showing the filtered traces for each data component and the onset functions calculated from them (P and/or S) for each station. The search window to make a phase pick is displayed, along with the dynamic pick threshold, the phase pick time and its uncertainty (if made) and the Gaussian fit to the onset function.

Parameters

- **event** (*Event* object) – Light class to encapsulate information about an event, including origin time, location and waveform data.
- **station** (*str*) – Station name.
- **onset_data** (*OnsetData* object) – Light class encapsulating data generated during onset calculation.
- **picks_df** (*pandas.DataFrame* object) – DataFrame that contains the measured picks with columns: ["Name", "Phase", "ModelledTime", "PickTime", "PickError", "SNR"] Each row contains the phase pick from one station/phase.
- **traveltimes** (*list of float*) – Modelled traveltimes from the event hypocentre to the station for each phase to be plotted.

- **run** (*Run* object) – Light class encapsulating i/o path information for a given run.

quakemigrate.signal.local_mag

The `quakemigrate.local_mag` extension module handles the calculation of local magnitudes from Wood-Anderson simulated waveforms.

> **Warning:** The local_mag modules are an ongoing work in progress. We hope to continue to extend their functionality, which may result in some API changes. If you have comments or suggestions, please contact the QuakeMigrate developers at: quakemigrate.developers@gmail.com, or submit an issue on GitHub.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

quakemigrate.signal.local_mag.local_mag

Module containing methods to calculate the local magnitude for an event located by `QuakeMigrate`.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

class `quakemigrate.signal.local_mag.local_mag.`**`LocalMag`**(*amp_params*, *mag_params*, *plot_amplitudes=True*)

Bases: `object`

QuakeMigrate extension class for calculating local magnitudes.

Provides functions for measuring amplitudes of earthquake waveforms and using these to calculate local magnitudes.

Parameters

- **amp_params** (*dict*) – All keys are optional, including: signal_window : float

  Length of S-wave signal window, in addition to the time window associated with the marginal_window and traveltime uncertainty. (Default 0 s)

  noise_window
  [float] Length of the time window before the P-wave signal window in which to measure the noise amplitude. (Default 10 s)

  noise_measure
  [{"RMS", "STD", "ENV"}] Method by which to measure the noise amplitude; root-mean-quare, standard deviation or average amplitude of the envelope of the signal. (Default "RMS")

  loc_method
  [{"spline", "gaussian", "covariance"}] Which event location estimate to use. (Default "spline")

highpass_filter
[bool] Whether to apply a highpass filter to the data before measuring amplitudes. (Default False)

highpass_freq
[float] High-pass filter frequency. Required if highpass_filter is True.

bandpass_filter
[bool] Whether to apply a band-pass filter before measuring amplitudes. (Default: False)

bandpass_lowcut
[float] Band-pass filter low-cut frequency. Required if bandpass_filter is True.

bandpass_highcut
[float] Band-pass filter high-cut frequency. Required if bandpass_filter is True.

filter_corners
[int] Number of corners for the chosen filter. Default: 4.

prominence_multiplier
[float] To set a prominence filter in the peak-finding algorithm. (Default 0. = off). NOTE: not recommended for use in combination with a filter; filter gain corrections can lead to spurious results. Please see the *scipy.signal.find_peaks* documentation for further guidance.

- `mag_params` (*dict*) – Required keys: A0 : str or func

  Name of the attenuation function to use. Available options include {"Hutton-Boore", "keir2006", "UK", . . . }. Alternatively specify a function which returns the attenuation factor at a specified (epicentral or hypocentral) distance. (Default "Hutton-Boore")

  All other keys are optional, including: station_corrections : dict {str : float}

  Dictionary of trace_id : magnitude-correction pairs. (Default None)

amp_feature
[{"S_amp", "P_amp"}] Which phase amplitude measurement to use to calculate local magnitude. (Default "S_amp")

amp_multiplier
[float] Factor by which to multiply all measured amplitudes.

use_hyp_dist
[bool, optional] Whether to use the hypocentral distance instead of the epicentral distance in the local magnitude calculation. (Default False)

trace_filter
[regex expression] Expression by which to select traces to use for the mean_magnitude calculation. E.g. '.*H[NE]$'. (Default None)

station_filter
[list of str] List of stations to exclude from the mean_magnitude calculation. E.g. ["KVE", "LIND"]. (Default None)

dist_filter
[float or False] Whether to only use stations less than a specified (epicentral or hypocentral) distance from an event in the mean_magnitude() calculation. Distance in kilometres. (Default False)

pick_filter
[bool] Whether to only use stations where at least one phase was picked by the autopicker in the mean_magnitude calculation. (Default False)

noise_filter
[float] Factor by which to multiply the measured noise amplitude before excluding amplitude observations below the noise level. (Default 1.)

weighted_mean
[bool] Whether to do a weighted mean of the magnitudes when calculating the mean_magnitude. (Default False)

- **plot_amplitudes** (*bool, optional*) – Plot amplitudes vs. distance plot for each event. (Default True)

`amp`

The Amplitude object for this instance of LocalMag. Contains functions to measure Wood-Anderson corrected displacement amplitudes for an event.

Type
Amplitude object

`mag`

The Magnitude object for this instance of LocalMag. Contains functions to calculate magnitudes from Wood-Anderson corrected displacement amplitudes, and to combine them into a single magnitude estimate for the event.

Type
Magnitude object

calc_magnitude(*event*, *lut*, *run*)

calc_magnitude(*event*, *lut*, *run*)

Wrapper function to calculate the local magnitude of an event by first making Wood-Anderson corrected displacement amplitude measurements on each trace, then calculating magnitudes from these individual measurements, and a network-averaged (weighted) mean magnitude estimate and associated uncertainty.

Additional functionality includes calculating an r^2 fit of the predicted amplitude with distance curve to the observed amplitudes, and an associated plot of amplitudes vs. distance.

Parameters

- **event** (*Event* object) – Light class encapsulating waveform data, onset, pick and location information for a given event.
- **lut** (*LUT* object) – Contains the traveltime lookup tables for seismic phases, computed for some pre-defined velocity model.
- **run** (*Run* object) – Light class encapsulating waveforms, coalescence information, picks and location information for a given event.

Returns

- **event** (*Event* object) – Light class encapsulating waveforms, coalescence information, picks and location information for a given event. Now also contains local magnitude information.
- **mag** (*float*) – Network-averaged local magnitude estimate for this event.

quakemigrate.signal.local_mag.amplitude

Module containing methods to measure Wood-Anderson corrected waveform amplitudes to be used for local magnitude calculation.

copyright
: 2020–2023, QuakeMigrate developers.

license
: GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

`class quakemigrate.signal.local_mag.amplitude.`**`Amplitude`**`(`*amplitude_params={}*`)`

Bases: `object`

Part of the QuakeMigrate LocalMag class; measures Wood-Anderson corrected waveform amplitudes to be used for local magnitude calculation.

Simulates the Wood-Anderson waveforms using a user-supplied set of response removal parameters, then measures the maximum peak-to-trough amplitude in time windows around the P and S phase arrivals. These windows are calculated from the phase pick times from the autopicker, if available, or from the modelled pick times. The length of the S-wave signal window is calculated according to a user-specified *signal_window* parameter.

The user may optionally specify a filter to apply to the waveforms before amplitudes are measured, in order (for example) to reduce the impact of high-amplitude noise associated with the oceanic microseisms on the measurement of low-amplitude wavetrains associated with microseismic events. Note this will generally result in an underestimate of the true earthquake waveform amplitude, even when the filter gain is corrected for.

A measurement of the signal amplitude in a window preceding the P-wave arrival is used to characterise the "noise" amplitude. This can be used to filter out null observations, and to provide an estimate of the uncertainty on the max amplitude measurements contributed by extraneous noise.

`signal_window`

Length of S-wave signal window, in addition to the time window associated with the marginal_window and traveltime uncertainty. (Default 0 s)

Type
: float

`noise_window`

Length of the time window before the P-wave signal window in which to measure the noise amplitude. (Default 5 s)

Type
: float

`noise_measure`

Method by which to measure the noise amplitude; root-mean-quare, standard deviation or average amplitude of the envelope of the signal. (Default "RMS")

Type
: {"RMS", "STD", "ENV"}

`loc_method`

Which event location estimate to use. (Default "spline")

Type
: {"spline", "gaussian", "covariance"}

`highpass_filter`

Whether to apply a high-pass filter before measuring amplitudes. (Default False)

Type
bool

`highpass_freq`
High-pass filter frequency. Required if highpass_filter is True.

Type
float

`bandpass_filter`
Whether to apply a band-pass filter before measuring amplitudes. (Default False)

Type
bool

`bandpass_lowcut`
Band-pass filter low-cut frequency. Required if bandpass_filter is True.

Type
float

`bandpass_highcut`
Band-pass filter high-cut frequency. Required if bandpass_filter is True.

Type
float

`filter_corners`
number of corners for the chosen filter. (Default 4)

Type
int

`prominence_multiplier`
To set a prominence filter in the peak-finding algorithm. (Default 0. = off) NOTE: not recommended for use in combination with a filter; filter gain corrections can lead to spurious results. Please see the *scipy.signal.find_peaks* documentation for further guidance.

Type
float

`get_amplitudes`(*event*, *lut*)

Raises

- **`AttributeError`** – If both *highpass_filter* and *bandpass_filter* are selected, or if the user selects to apply a filter but does not provide the relevant frequencies.
- **`AttributeError`** – If response removal parameters are provided here instead of to the *`Archive`* object.

`get_amplitudes`(*event*, *lut*)
Measure phase amplitudes for an event.

Parameters

- **event** (*`Event`* object) – Light class encapsulating waveforms, coalescence information, picks and location information for a given event.
- **lut** (*`LUT`* object) – Contains the traveltime lookup tables for seismic phases, computed for some pre-defined velocity model.

Returns

amplitudes – P- and S-wave amplitude measurements for each component of each station in the look-up table. Columns:

epi_dist
[float] Epicentral distance between the station and the event hypocentre.

z_dist
[float] Vertical distance between the station and the event hypocentre.

P_amp
[float] Half maximum peak-to-trough amplitude in the P signal window. In *millimetres*. Corrected for filter gain, if applicable.

P_freq
[float] Approximate frequency of the maximum amplitude P-wave signal. Calculated from the peak-to-trough time interval of the max peak-to-trough amplitude.

P_time
[*obspy.UTCDateTime* object] Approximate time of amplitude observation (halfway between peak and trough times).

P_avg_amp
[float] Average amplitude in the P signal window, measured by the same method as the Noise_amp (see *noise_measure*) and corrected for the same filter gain as *P_amp*. In *millimetres*.

P_filter_gain
[float or NaN] Filter gain at *P_freq* - which has been corrected for in the P_amp measurements - if a filter was applied prior to amplitude measurement; Else NaN.

S_amp
[float] As for P, but in the S wave signal window.

S_freq
[float] As for P.

S_time
[*obspy.UTCDateTime* object] As for P.

S_avg_amp
[float] As for P.

S_filter_gain
[float or NaN.] As for P.

Noise_amp
[float] The average signal amplitude in the noise window. In *millimetres*. See *noise_measure* parameter.

is_picked
[bool] Whether at least one of the phase arrivals was picked by the autopicker.

Index = Trace ID (see *obspy.Trace* object property 'id')

Return type

pandas.DataFrame object

pad(*marginal_window*, *max_tt*, *fraction_tt*)

Calculate padding, including an allowance for the taper applied when filtering/ removing instrument response, to ensure the noise and signal window amplitude measurements are not affected by the taper.

Parameters

- **marginal_window** (*float*) – Half-width of window centred on the maximum coalescence time of the event over which the 4-D coalescence function is marginalised. Used here as an estimate of the origin time uncertainity when calculating the signal windows.
- **max_tt** (*float*) – Maximum traveltime in the look-up table.
- **fraction_tt** (*float*) – An estimate of the uncertainty in the velocity model as a function of a fraction of the traveltime. (Default 0.1 == 10%)

Returns

- **pre_pad** (*float*) – Time window by which to pre-pad the data when reading from the waveform archive.
- **post_pad** (*float*) – Time window by which to post-pad the data when reading from the waveform archive.

quakemigrate.signal.local_mag.magnitude

Module that supplies functions to calculate magnitudes from observations of trace amplitudes, earthquake location, station locations, and an estimated attenuation curve for the region of interest.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

class quakemigrate.signal.local_mag.magnitude.**Magnitude**(*magnitude_params={}*)

Bases: object

Part of the QuakeMigrate LocalMag class; calculates local magnitudes from Wood-Anderson corrected waveform amplitude measurements.

Takes waveform amplitude measurements from the LocalMag Amplitude class, and from these calculates local magnitude estimates using a local magnitude attenuation function. Magnitude corrections for individual stations and channels thereof can be applied, if provided.

Individual estimates are then combined to calculate a network-averaged (weighted) mean local magnitude for the event. Also includes the function to measure the r-squared statistic assessing the goodness of fit between the predicted amplitude with distance from the nework-averaged local magnitude for the event and chosen attenuation function, and the observed amplitudes. This, provides a tool to distinguish between real microseismic events and artefacts.

A summary plot illustrating the amplitude observations, their uncertainties, and the predicted amplitude with distance for the network-averaged local magnitude (and its uncertainties) can optionally be output.

A0

Name of the attenuation function to use. Available options include {"Hutton-Boore", "keir2006", "UK", . . . }. Alternatively specify a function which returns the attenuation factor at a specified (epicentral or hypocentral) distance. (Default "Hutton-Boore")

Type
str or func

`use_hyp_dist`
Whether to use the hypocentral distance instead of the epicentral distance in the local magnitude calculation. (Default False)

Type
bool, optional

`amp_feature`
Which phase amplitude measurement to use to calculate local magnitude. (Default "S_amp")

Type
{"S_amp", "P_amp"}

`station_corrections`
Dictionary of trace_id : magnitude-correction pairs. (Default None)

Type
dict {str : float}

`amp_multiplier`
Factor by which to multiply all measured amplitudes.

Type
float

`weighted_mean`
Whether to use a weighted mean to calculate the network-averaged local magnitude estimate for the event. (Default False)

Type
bool

`trace_filter`
Expression by which to select traces to use for the mean_magnitude calculation. E.g. ".*H[NE]$" . (Default None)

Type
regex expression

`noise_filter`
Factor by which to multiply the measured noise amplitude before excluding amplitude observations below the noise level. (Default 1.)

Type
float

`station_filter`
List of stations to exclude from the mean_magnitude calculation. E.g. ["KVE", "LIND"]. (Default None)

Type
list of str

`dist_filter`
Whether to only use stations less than a specified (epicentral or hypocentral) distance from an event in the mean_magnitude() calculation. Distance in kilometres. (Default False)

Type
float or False

`pick_filter`

Whether to only use stations where at least one phase was picked by the autopicker in the mean_magnitude calculation. (Default False)

Type
bool

`r2_only_used`

Whether to only use amplitude observations which were used for the mean magnitude calculation when calculating the r-squared statistic for the goodness of fit between the measured and predicted amplitudes. Default: True; False is an experimental feature - use with caution.

Type
bool

`calculate_magnitudes`(*amplitudes*)

`mean_magnitude`(*magnitudes*)

`plot_amplitudes`(*event*, *run*)

Raises

- `TypeError` – If the user does not specify an A0 attenuation curve.
- `ValueError` – If the user specifies an invalid A0 attenuation curve.

`calculate_magnitudes`(*amplitudes*)

Calculate magnitude estimates from amplitude measurements on individual stations /components.

Parameters
`amplitudes` (*pandas.DataFrame* object) – P- and S-wave amplitude measurements for each component of each station in the look-up table. Columns:

epi_dist
[float] Epicentral distance between the station and the event hypocentre.

z_dist
[float] Vertical distance between the station and the event hypocentre.

P_amp
[float] Half maximum peak-to-trough amplitude in the P signal window. In *millimetres*. Corrected for filter gain, if applicable.

P_freq
[float] Approximate frequency of the maximum amplitude P-wave signal. Calculated from the peak-to-trough time interval of the max peak-to-trough amplitude.

P_time
[*obspy.UTCDateTime* object] Approximate time of amplitude observation (halfway between peak and trough times).

P_avg_amp
[float] Average amplitude in the P signal window, measured by the same method as the Noise_amp (see *noise_measure*) and corrected for the same filter gain as *P_amp*. In *millimetres*.

P_filter_gain
[float or NaN] Filter gain at *P_freq* - which has been corrected for in the P_amp measurements - if a filter was applied prior to amplitude measurement; Else NaN.

S_amp
[float] As for P, but in the S wave signal window.

S_freq
[float] As for P.

S_time
[*obspy.UTCDateTime* object] As for P.

S_avg_amp
[float] As for P.

S_filter_gain
[float or NaN.] As for P.

Noise_amp
[float] The average signal amplitude in the noise window. In *millimetres*. See *noise_measure* parameter.

is_picked
[bool] Whether at least one of the phase arrivals was picked by the autopicker.

Index = Trace ID (see *obspy.Trace* object property 'id')

Returns

magnitudes – The original amplitudes DataFrame, with columns containing the calculated magnitude and an associated error now added. Columns = ["epi_dist", "z_dist", "P_amp", "P_freq", "P_time",

"P_avg_amp", "P_filter_gain", "S_amp", "S_freq", "S_time", "S_avg_amp", "S_filter_gain", "Noise_amp", "is_picked", "ML", "ML_Err"]

Index = Trace ID (see *obspy.Trace.id*) Additional fields: ML : float

Magnitude calculated from the chosen amplitude measurement, using the specified attenuation curve and station_corrections.

ML_Err
[float] Estimate of the error on the calculated magnitude, based on potential errors in the maximum amplitude measurement according to the measured noise amplitude.

Return type
pandas.DataFrame object

Raises
`AttributeError` – If A0 attenuation correction is not specified.

`mean_magnitude`(*magnitudes*)

Calculate the network-averaged local magnitude for an event based on the magnitude estimates calculated from amplitude measurements made on each component of each station.

The user may specify distance, station, channel and a number of other filters to restrict which observations are included in this best estimate of the local magnitude of the event.

Parameters
`magnitudes` (*pandas.DataFrame* object) – Contains P- and S-wave amplitude measurements for each component of each station in the look-up table, and local magnitude estimates calculated from them (output by calculate_magnitudes()). Note that the amplitude observations are raw, but the ML estimates derived from them include station corrections, if provided. Columns:

epi_dist
[float] Epicentral distance between the station and the event hypocentre.

z_dist
[float] Vertical distance between the station and the event hypocentre.

P_amp
[float] Half maximum peak-to-trough amplitude in the P signal window. In *millimetres*. Corrected for filter gain, if applicable.

P_freq
[float] Approximate frequency of the maximum amplitude P-wave signal. Calculated from the peak-to-trough time interval of the max peak-to-trough amplitude.

P_time
[*obspy.UTCDateTime* object] Approximate time of amplitude observation (halfway between peak and trough times).

P_avg_amp
[float] Average amplitude in the P signal window, measured by the same method as the Noise_amp (see *noise_measure*) and corrected for the same filter gain as *P_amp*. In *millimetres*.

P_filter_gain
[float or NaN] Filter gain at *P_freq* - which has been corrected for in the P_amp measurements - if a filter was applied prior to amplitude measurement; Else NaN.

S_amp
[float] As for P, but in the S wave signal window.

S_freq
[float] As for P.

S_time
[*obspy.UTCDateTime* object] As for P.

S_avg_amp
[float] As for P.

S_filter_gain
[float or NaN.] As for P.

Noise_amp
[float] The average signal amplitude in the noise window. In *millimetres*. See *noise_measure* parameter.

is_picked
[bool] Whether at least one of the phase arrivals was picked by the autopicker.

ML
[float] Magnitude calculated from the chosen amplitude measurement, using the specified attenuation curve and station_corrections.

ML_Err
[float] Estimate of the error on the calculated magnitude, based on potential errors in the maximum amplitude measurement according to the measured noise amplitude.

Index = Trace ID (see *obspy.Trace* object property 'id')

Returns

- **mean_mag** (*float or NaN*) – Network-averaged local magnitude estimate for the event. Mean (or weighted mean) of the magnitude estimates calculated from each individual channel after optionally removing some observations based on trace ID, distance, etcetera.
- **mean_mag_err** (*float or NaN*) – Standard deviation (or weighted standard deviation) of the magnitude estimates calculated from individual channels which contributed to the calculation of the (weighted) mean magnitude.
- **mag_r_squared** (*float or NaN*) – r-squared statistic describing the fit of the amplitude vs. distance curve predicted by the calculated mean_mag and chosen attenuation model to the measured amplitude observations. This is intended to be used to help discriminate between 'real' events, for which the predicted amplitude vs. distance curve should provide a good fit to the observations, from artefacts, which in general will not.

`plot_amplitudes`(*magnitudes*, *event*, *run*, *unit_conversion_factor*, *noise_measure='RMS'*)

Plot a figure showing the measured amplitude with distance vs. predicted amplitude with distance derived from mean_mag and the chosen attenuation model.

The amplitude observations (both for noise and signal amplitudes) are corrected according to the same station corrections that were used in calculating the individual local magnitude estimates that were used to calculate the network-averaged local magnitude for the event.

Parameters

- `magnitudes` (*pandas.DataFrame* object) – Contains P- and S-wave amplitude measurements for each component of each station in the look-up table, and local magnitude estimates calculated from them (output by calculate_magnitudes()). Note that the amplitude observations are raw, but the ML estimates derived from them include station corrections, if provided. Columns = ["epi_dist", "z_dist", "P_amp", "P_freq", "P_time", "P_avg_amp",

  "P_filter_gain", "S_amp", "S_freq", "S_time", "S_avg_amp", "S_filter_gain", "Noise_amp", "is_picked", "ML", "ML_Err", "Noise_Filter", "Trace_Filter", "Station_Filter", "Dist_Filter", "Dist", "Used"]
- **event** (`Event` object) – Light class encapsulating waveforms, coalescence information, picks and location information for a given event.
- **run** (`Run` object) – Light class encapsulating i/o path information for a given run.
- **unit_conversion_factor** (`float`) – A conversion factor based on the lookup table grid projection, used to ensure the location uncertainties have units of kilometres.

quakemigrate.signal.scan

Module to perform core QuakeMigrate functions: detect() and locate().

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

class quakemigrate.signal.scan.QuakeScan(*archive*, *lut*, *onset*, *run_path*, *run_name*, ***kwargs*)

Bases: `object`

QuakeMigrate scanning class.

Provides an interface for the wrapped compiled C functions, used to perform the continuous scan (detect) or refined event migrations (locate).

Parameters

- **archive** (*Archive* object) – Details the structure and location of a data archive and provides methods for reading data from file.
- **lut** (*LUT* object) – Contains the traveltime lookup tables for seismic phases, computed for some pre-defined velocity model.
- **onset** (*Onset* object) – Provides callback methods for calculation of onset functions.
- **run_path** (*str*) – Points to the top level directory containing all input files, under which the specific run directory will be created.
- **run_name** (*str*) – Name of the current QuakeMigrate run.
- **kwargs** (**dict) – See QuakeScan Attributes for details. In addition to these:

continuous_scanmseed_write

Option to continuously write the .scanmseed file output by detect() at the end of every time step. Default behaviour is to write in day chunks where possible. Default: False.

Type
bool

cut_waveform_format

File format used when writing waveform data. We support any format also supported by ObsPy - “MSEED” (default), “SAC”, “SEGY”, “GSE2”.

Type
str, optional

log

Toggle for logging. If True, will output to stdout and generate a log file. Default is to only output to stdout.

Type
bool, optional

loglevel

Toggle to set the logging level: “debug” will print out additional diagnostic information to the log and stdout. (Default “info”)

Type
{“info”, “debug”}, optional

`mags`

Provides methods for calculating local magnitudes, performed during locate.

Type

LocalMag object, optional

`marginal_window`

Half-width of window centred on the maximum coalescence time. The 4-D coalescence functioned is marginalised over time across this window such that the earthquake location and associated uncertainty can be appropriately calculated. It should be an estimate of the time uncertainty in the earthquake origin time, which itself is some combination of the expected spatial uncertainty and uncertainty in the seismic velocity model used. Default: 2 seconds.

Type

float, optional

`picker`

Provides callback methods for phase picking, performed during locate.

Type

PhasePicker object, optional

`plot_event_summary`

Plot event summary figure - see *quakemigrate.plot* for more details. Default: True.

Type

bool, optional

`plot_event_video`

Plot coalescence video for each located earthquake. Default: False.

Type

bool, optional

`post_pad`

Additional amount of data to read in after the timestep, used to ensure the correct coalescence is calculated at every sample.

Type

float

`pre_pad`

Additional amount of data to read in before the timestep, used to ensure the correct coalescence is calculated at every sample.

Type

float

`real_waveform_units`

Units to output real cut waveforms.

Type

{"displacement", "velocity"}

`run`

Light class encapsulating i/o path information for a given run.

Type

Run object

`scan_rate`

Sampling rate at which the 4-D coalescence map will be calculated. Currently fixed to be the same as the onset function sampling rate (not user-configurable).

Type
int, optional

`threads`

The number of threads for the C functions to use on the executing host. Default: 1 thread.

Type
int, optional

`timestep`

Length (in seconds) of timestep used in detect(). Note: total detect run duration should be divisible by timestep. Increasing timestep will increase RAM usage during detect, but will slightly speed up overall detect run. Default: 120 seconds.

Type
float, optional

`wa_waveform_units`

Units to output Wood-Anderson simulated cut waveforms.

Type
{"displacement", "velocity"}

`write_cut_waveforms`

Write raw cut waveforms for all data read from the archive for each event located by locate(). See ~*quakemigrate.io.data.Archive* parameter *read_all_stations*. Default: False. NOTE: this data has not been processed or quality-checked!

Type
bool, optional

`write_real_waveforms`

Write real cut waveforms for all data read from the archive for each event located by locate(). See ~*quakemigrate.io.data.Archive* parameter *read_all_stations*. Default: False. NOTE: the units of this data (displacement or velocity) are controlled by *real_waveform_units*. NOTE: this data has not been processed or quality-checked! NOTE: no padding has been added to take into account the taper applied during response removal.

Type
bool, optional

`write_wa_waveforms`

Write Wood-Anderson simulated cut waveforms for all data read from the archive for each event located by locate(). See ~*quakemigrate.io.data.Archive* parameter *read_all_stations*. Default: False. NOTE: the units of this data (displacement or velocity) are controlled by *wa_waveform_units*. NOTE: this data has not been processed or quality-checked! NOTE: no padding has been added to take into account the taper applied during response removal.

Type
bool, optional

`xy_files`

Path to comma-separated value file (.csv) containing a series of coordinate files to plot. Columns: ["File", "Color", "Linewidth", "Linestyle"], where "File" is the absolute path to the file containing the coordinates to be plotted. E.g: "/home/user/volcano_outlines.csv,black,0.5,-". Each .csv coordinate file should contain coordinates only, with columns: ["Longitude", "Latitude"]. E.g.: "-17.5,64.8". Lines pre-pended with # will be treated as a comment - this can be used to include references. See the Volcanotectonic_Iceland example XY_files for a template.

Note: Do not include a header line in either file.

Type
str, optional

`+++ TO BE REMOVED TO ARCHIVE CLASS +++`

`pre_cut`

Specify how long before the event origin time to cut the waveform data from.

Type
float, optional

`post_cut`

Specify how long after the event origin time to cut the waveform. data to

Type
float, optional

`+++ TO BE REMOVED TO ARCHIVE CLASS +++`

`detect`(*starttime*, *endtime*)

Core detection method – compute decimated 3-D coalescence continuously throughout entire time period; output as .scanmseed (in mSEED format).

`locate`(*starttime*, *endtime) or locate(file*)

Core locate method – compute 3-D coalescence over short time window around candidate earthquake triggered from continuous detect output; output location & uncertainties (.event file), phase picks (.picks file), plus multiple optional plots / data for further analysis and processing.

Raises

- ***OnsetTypeError*** – If an object is passed in through the *onset* argument that is not derived from the `Onset` base class.
- ***PickerTypeError*** – If an object is passed in through the *picker* argument that is not derived from the `PhasePicker` base class.
- **RuntimeError** – If the user does not supply the locate function with valid arguments.
- ***TimeSpanException*** – If the user supplies a starttime that is after the endtime.

`detect`(*starttime*, *endtime*)

Scans through data calculating coalescence in a (decimated) 3-D grid by continuously migrating onset functions.

Parameters

- **starttime** (*str*) – Timestamp from which to run continuous scan.
- **endtime** (*str*) – Timestamp up to which to run continuous scan. Note: the last sample returned will be that which immediately precedes this timestamp.

`locate`(*starttime=None*, *endtime=None*, *trigger_file=None*)

Re-computes the coalescence on an undecimated grid for a short time window around each candidate earthquake triggered from the (decimated) continuous detect scan. Calculates event location and uncertainties, makes phase arrival picks, plus multiple optional plotting / data outputs for further analysis and processing.

Parameters

- **starttime** (*str, optional*) – Timestamp from which to include events in the locate scan.
- **endtime** (*str, optional*) – Timestamp up to which to include events in the locate scan.
- **trigger_file** (*str, optional*) – File containing triggered events to be located.

property n_cores

Handler for deprecated attribute name 'n_cores'

property sampling_rate

Get sampling_rate

property scan_rate

Get scan_rate

property time_step

Handler for deprecated attribute name 'time_step'

quakemigrate.signal.trigger

Module to perform the trigger stage of QuakeMigrate.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

`class quakemigrate.signal.trigger.Trigger`(*lut*, *run_path*, *run_name*, ***kwargs*)

Bases: `object`

QuakeMigrate triggering class.

Triggers candidate earthquakes from the continuous maximum coalescence through time data output by the decimated detect scan, ready to be run through locate().

Parameters

- **lut** (*LUT* object) – Contains the traveltime lookup tables for the selected seismic phases, computed for some pre-defined velocity model.
- **run_path** (*str*) – Points to the top level directory containing all input files, under which the specific run directory will be created.
- **run_name** (*str*) – Name of the current QuakeMigrate run.
- **kwargs** (**dict) – See Trigger Attributes for details. In addition to these: log : bool, optional

  Toggle for logging. If True, will output to stdout and generate a log file. Default is to only output to stdout.

loglevel
[{“info”, “debug”}, optional] Toggle to set the logging level: “debug” will print out additional diagnostic information to the log and stdout. (Default “info”)

trigger_name
[str] Optional name of a sub-run - useful when testing different trigger parameters, for example.

`mad_window_length`

Length of window within which to calculate the Median Average Deviation. Default: 3600 seconds (1 hour).

Type
float, optional

`mad_multiplier`

A scaling factor for the MAD output to make the calculated MAD factor a consistent estimation of the standard deviation of the distribution. Default: 1.4826, which is the appropriate scaling factor for a normal distribution.

Type
float, optional

`marginal_window`

Half-width of window centred on the maximum coalescence time. The 4-D coalescence functioned is marginalised over time across this window such that the earthquake location and associated uncertainty can be appropriately calculated. It should be an estimate of the time uncertainty in the earthquake origin time, which itself is some combination of the expected spatial uncertainty and uncertainty in the seismic velocity model used. Default: 2 seconds.

Type
float, optional

`min_event_interval`

Minimum time interval between triggered events. Must be at least twice the marginal window. Default: 4 seconds.

Type
float, optional

`normalise_coalescence`

If True, triggering is performed on the maximum coalescence normalised by the mean coalescence value in the 3-D grid. Default: False.

Type
bool, optional

`pad`

Additional time padding to ensure events close to the starttime/endtime are not cut off and missed. Default: 120 seconds.

Type
float, optional

`plot_trigger_summary`

Plot triggering through time for each batched segment. Default: True.

Type
bool, optional

run

Light class encapsulating i/o path information for a given run.

Type
Run object

static_threshold

Static threshold value above which to trigger candidate events.

Type
float, optional

threshold_method

Toggle between a "static" threshold and a "dynamic" threshold, based on the Median Average Deviation. Default: "static".

Type
str, optional

xy_files

Path to comma-separated value file (.csv) containing a series of coordinate files to plot. Columns: ["File", "Color", "Linewidth", "Linestyle"], where "File" is the absolute path to the file containing the coordinates to be plotted. E.g: "/home/user/volcano_outlines.csv,black,0.5,-". Each .csv coordinate file should contain coordinates only, with columns: ["Longitude", "Latitude"]. E.g.: "-17.5,64.8". Lines pre-pended with # will be treated as a comment - this can be used to include references. See the Volcanotectonic_Iceland example XY_files for a template.

Note: Do not include a header line in either file.

Type
str, optional

plot_all_stns

If true, plot all stations used for detect. Otherwise, only plot stations which for which some data was available during the trigger time window. NOTE: if no station availability data is found, all stations in the LUT will be plotted. (Default: True)

Type
bool, optional

trigger(*starttime*, *endtime*, *region=None*, *interactive_plot=False*)

Trigger candidate earthquakes from decimated detect scan results.

Raises

- **ValueError** – If *min_event_interval* < 2 * *marginal_window*.
- ***InvalidTriggerThresholdMethodException*** – If an invalid threshold method is passed in by the user.
- ***TimeSpanException*** – If the user supplies a starttime that is after the endtime.

property min_event_interval

Get and set the minimum event interval.

property minimum_repeat

Handler for deprecated attribute name 'minimum_repeat'.

trigger(*starttime*, *endtime*, *region=None*, *interactive_plot=False*)

Trigger candidate earthquakes from decimated scan data.

Parameters

- **starttime** (*str*) – Timestamp from which to trigger events.
- **endtime** (*str*) – Timestamp up to which to trigger events.
- **region** (*list of floats, optional*) –

  Only retain triggered events located within this region. Format is:
  [Xmin, Ymin, Zmin, Xmax, Ymax, Zmax]

  As longitude / latitude / depth (units corresponding to the lookup table grid projection; in positive-down frame).
- **interactive_plot** (*bool, optional*) – Toggles whether to produce an interactive plot. Default: False.

Raises

TimeSpanException – If *starttime* is after *endtime*.

quakemigrate.signal.trigger.**chunks2trace**(*a*, *new_shape*)

Create a trace filled with chunks of the same value.

Parameters:

a
[array-like] Array of chunks.

new_shape
[tuple of ints] (number of chunks, chunk_length).

Returns:

b
[array-like] Single array of values contained in *a*.

4.3.7 quakemigrate.util

Module that supplies various utility functions and classes.

copyright
2020–2023, QuakeMigrate developers.

license
GNU General Public License, Version 3 (https://www.gnu.org/licenses/gpl-3.0.html)

exception quakemigrate.util.**ArchiveEmptyException**

Bases: `Exception`

Custom exception to handle empty archive.

exception quakemigrate.util.**ArchiveFormatException**

Bases: `Exception`

Custom exception to handle case where Archive.format is not set.

exception `quakemigrate.util.`**`ArchivePathStructureError`**(*archive_format*)

Bases: `Exception`

Custom exception to handle case where an invalid Archive path structure is selected.

exception `quakemigrate.util.`**`BadUpfactorException`**(*trace*)

Bases: `Exception`

Custom exception to handle case when the chosen upfactor does not create a trace with a sampling rate that can be decimated to the target sampling rate.

exception `quakemigrate.util.`**`ChannelNameException`**(*trace*)

Bases: `Exception`

Custom exception to handle case when waveform data header has channel names which do not conform to the IRIS SEED standard.

exception `quakemigrate.util.`**`DataAvailabilityException`**

Bases: `Exception`

Custom exception to handle case when all data for the selected stations did not pass the data quality criteria specified by the user.

exception `quakemigrate.util.`**`DataGapException`**

Bases: `Exception`

Custom exception to handle case when no data is found for the selected stations for a given timestep.

class `quakemigrate.util.`**`DateFormatter`**(*fmt*, *precision=3*)

Bases: `Formatter`

Extend the *matplotlib.ticker.Formatter* class to allow for millisecond precision when formatting a tick (in days since the epoch) with a *datetime.datetime.strftime* format string.

Parameters

- **fmt** (*str*) – *datetime.datetime.strftime* format string.
- **precision** (*int*) – Degree of precision to which to report sub-second time intervals.

exception `quakemigrate.util.`**`InvalidPickThresholdMethodException`**

Bases: `Exception`

Custom exception to handle case when the user has not selected a valid pick threshold method.

exception `quakemigrate.util.`**`InvalidTriggerThresholdMethodException`**

Bases: `Exception`

Custom exception to handle case when the user has not selected a valid trigger threshold method.

exception `quakemigrate.util.`**`InvalidVelocityModelHeader`**(*key*)

Bases: `Exception`

Custom exception to handle incorrect header columns in station file.

exception `quakemigrate.util.`**`LUTPhasesException`**(*message*)

Bases: `Exception`

Custom exception to handle the case when the look-up table does not contain the traveltimes for the phases necessary for a given function.

exception quakemigrate.util.**MagsTypeError**

Bases: Exception

Custom exception to handle case when an object has been provided to calculate magnitudes during locate, but it isn't supported.

exception quakemigrate.util.**NoOnsetPeak**(*pick_threshold*)

Bases: Exception

Custom exception to handle case when no values in the onset function exceed the threshold used for picking.

exception quakemigrate.util.**NoScanMseedDataException**

Bases: Exception

Custom exception to handle case when no .scanmseed files can be found by read_coastream().

exception quakemigrate.util.**NoStationAvailabilityDataException**

Bases: Exception

Custom exception to handle case when no .StationAvailability files can be found by read_availability().

exception quakemigrate.util.**NoTriggerFilesFound**

Bases: Exception

Custom exception to handle case when no trigger files are found during locate. This can occur for one of two reasons - an entirely invalid time period was used (i.e. one that does not overlap at all with a period of time for which there exists TriggeredEvents.csv files) or an invalid run name was provided.

exception quakemigrate.util.**NyquistException**(*freqmax*, *f_nyquist*, *tr_id*)

Bases: Exception

Custom exception to handle the case where the specified filter has a lowpass corner above the signal Nyquist frequency.

Parameters

- **freqmax** (*float*) – Specified lowpass frequency for filter.
- **f_nyquist** (*float*) – Nyquist frequency for the relevant waveform data.
- **tr_id** (*str*) – ID string for the Trace.

exception quakemigrate.util.**OnsetTypeError**

Bases: Exception

Custom exception to handle case when the onset object passed to QuakeScan is not of the default type defined in QuakeMigrate.

exception quakemigrate.util.**PeakToTroughError**(*err*)

Bases: Exception

Custom exception to handle case when amplitude._peak_to_trough_amplitude encounters an anomalous set of peaks and troughs, so can't calculate an amplitude.

exception quakemigrate.util.**PickOrderException**(*event_uid*, *station*, *p_pick*, *s_pick*)

Bases: Exception

Custom exception to handle the case when the pick for the P phase is later than the pick for the S phase.

exception quakemigrate.util.**PickerTypeError**

Bases: Exception

Custom exception to handle case when the phase picker object passed to QuakeScan is not of the default type defined in QuakeMigrate.

exception quakemigrate.util.**ResponseNotFoundError**(*e*, *tr_id*)

Bases: `Exception`

Custom exception to handle the case where the provided response inventory doesn't contain the response information for a trace.

Parameters

- **e** (*str*) – Error message from ObsPy *Inventory.get_response()*.
- **tr_id** (*str*) – ID string for the Trace for which the response cannot be found.

exception quakemigrate.util.**ResponseRemovalError**(*e*, *tr_id*)

Bases: `Exception`

Custom exception to handle the case where the response removal was not successful.

Parameters

- **e** (*str*) – Error message from ObsPy *Trace.remove_response()* or *Trace.simulate()*.
- **tr_id** (*str*) – ID string for the Trace for which the response cannot be removed.

exception quakemigrate.util.**StationFileHeaderException**

Bases: `Exception`

Custom exception to handle incorrect header columns in station file.

exception quakemigrate.util.**TimeSpanException**

Bases: `Exception`

Custom exception to handle case when the user has submitted a start time that is after the end time.

quakemigrate.util.**calculate_mad**(*x*, *scale=1.4826*)

Calculates the Median Absolute Deviation (MAD) of the input array x.

Parameters

- **x** (*array-like*) – Input data.
- **scale** (*float, optional*) – A scaling factor for the MAD output to make the calculated MAD factor a consistent estimation of the standard deviation of the distribution.

Returns

scaled_mad – Array of scaled mean absolute deviation values for the input array, x, scaled to provide an estimation of the standard deviation of the distribution.

Return type

array-like

quakemigrate.util.**decimate**(*trace*, *sampling_rate*)

Decimate a trace to achieve the desired sampling rate, sr.

NOTE: data will be detrended and a cosine taper applied before decimation, in order to avoid edge effects when applying the lowpass filter before decimating.

Parameters:

trace
: [*obspy.Trace* object] Trace to be decimated.

sampling_rate
: [int] Output sampling rate.

Returns:

trace
: [*obspy.Trace* object] Decimated trace.

`quakemigrate.util.gaussian_1d(x, a, b, c)`

Create a 1-dimensional Gaussian function.

Parameters

- **x** (*array-like*) – Array of x values.
- **a** (*float / int*) – Amplitude (height of Gaussian).
- **b** (*float / int*) – Mean (centre of Gaussian).
- **c** (*float / int*) – Sigma (width of Gaussian).

Returns
f – 1-dimensional Gaussian function

Return type
function

`quakemigrate.util.gaussian_3d(nx, ny, nz, sgm)`

Create a 3-dimensional Gaussian function.

Parameters

- **nx** (*array-like*) – Array of x values.
- **ny** (*array-like*) – Array of y values.
- **nz** (*array-like*) – Array of z values.
- **sgm** (*float / int*) – Sigma (width of gaussian in all directions).

Returns
f – 3-dimensional Gaussian function

Return type
function

`quakemigrate.util.logger(logstem, log, loglevel='info')`

Simple logger that will output to both a log file and stdout.

Parameters

- **logstem** (*str*) – Filestem for log file.
- **log** (*bool*) – Toggle for logging - default is to only print information to stdout. If True, will also create a log file.
- **loglevel** (*str, optional*) – Toggle for logging level - default is to print only "info" messages to log. To print more detailed "debug" messages, set to "debug".

`quakemigrate.util.`**`make_directories`**`(`*run*, *subdir=None*`)`

Make run directory, and optionally make subdirectories within it.

Parameters

- **`run`** (*pathlib.Path* object) – Location of parent output directory, named by run name.
- **`subdir`** (`str, optional`) – subdir to make beneath the run level.

`quakemigrate.util.`**`merge_stream`**`(`*stream*`)`

Merge all traces with contiguous data, or overlapping data which exactly matches (== st._cleanup(); i.e. no clobber). Apply this on a channel by channel basis so that if any individual merge fails then only that channel will be omitted.

Parameters

`stream` (*obspy.Stream* object) – Stream to be merged.

Returns

stream_merged – Merged Stream.

Return type

obpsy.Stream object

`quakemigrate.util.`**`pairwise`**`(`*iterable*`)`

Utility to iterate over an iterable pairwise.

`quakemigrate.util.`**`resample`**`(`*stream*, *sampling_rate*, *resample*, *upfactor*, *starttime*, *endtime*`)`

Resample data in an *obspy.Stream* object to the specified sampling rate.

By default, this function will only perform decimation of the data. If necessary, and if the user specifies *resample = True* and an upfactor to upsample by *upfactor = int*, data can also be upsampled and then, if necessary, subsequently decimated to achieve the desired sampling rate.

For example, for raw input data sampled at a mix of 40, 50 and 100 Hz, to achieve a unified sampling rate of 50 Hz, the user would have to specify an upfactor of 5; 40 Hz x 5 = 200 Hz, which can then be decimated to 50 Hz.

NOTE: assumes any data with off-sample timing has been corrected with `shift_to_sample()`. If not, the resulting traces may not all contain the correct number of samples.

NOTE: data will be detrended and a cosine taper applied before decimation, in order to avoid edge effects when applying the lowpass filter.

Parameters

- **`stream`** (*obspy.Stream* object) – Contains list of *obspy.Trace* objects to be decimated / resampled.
- **`resample`** (`bool`) – If true, perform resampling of data which cannot be decimated directly to the desired sampling rate.
- **`upfactor`** (`int or None`) – Factor by which to upsample the data to enable it to be decimated to the desired sampling rate, e.g. 40Hz -> 50Hz requires upfactor = 5.

Returns

stream – Contains list of resampled *obspy.Trace* objects at the chosen sampling rate *sr*.

Return type

obspy.Stream object

`quakemigrate.util.`**`shift_to_sample`**`(`*stream*, *interpolate=False*`)`

Check whether any data in an *obspy.Stream* object is "off-sample" - i.e. the data timestamps are *not* an integer number of samples after midnight. If so, shift data to be "on-sample".

This can either be done by shifting the timestamps by a sub-sample time interval, or interpolating the trace to the "on-sample" timestamps. The latter has the benefit that it will not affect the timing of the data, but will require additional computation time and some inevitable edge effects - though for onset calculation these should be contained within the pad windows. If you are using a sampling rate < 10 Hz, contact the QuakeMigrate developers.

Parameters

- **stream** (*obspy.Stream* object) – Contains list of *obspy.Trace* objects for which to check the timing.
- **interpolate** (*bool, optional*) – Whether to interpolate the data to correct the "off-sample" timing. Otherwise, the metadata will simply be altered to shift the timestamps "on-sample"; this will lead to a sub-sample timing offset.

Returns
stream – Waveform data with all timestamps "on-sample".

Return type
obspy.Stream object

quakemigrate.util.**time2sample**(*time*, *sampling_rate*)

Utility function to convert from seconds and sampling rate to number of samples.

Parameters

- **time** (*float*) – Time to convert.
- **sampling_rate** (*int*) – Sampling rate of input data/sampling rate at which to compute the coalescence function.

Returns
out – Time that correpsonds to an integer number of samples at a specific sampling rate.

Return type
int

quakemigrate.util.**timeit**(**args_*, ***kwargs_*)

Function wrapper that measures the time elapsed during its execution.

quakemigrate.util.**trim2sample**(*time*, *sampling_rate*)

Utility function to ensure time padding results in a time that is an integer number of samples.

Parameters

- **time** (*float*) – Time to trim.
- **sampling_rate** (*int*) – Sampling rate of input data/sampling rate at which to compute the coalescence function.

Returns
out – Time that correpsonds to an integer number of samples at a specific sampling rate.

Return type
int

quakemigrate.util.**upsample**(*trace*, *upfactor*, *starttime*, *endtime*)

Upsample a data stream by a given factor, prior to decimation. The upsampling is carried out by linear interpolation.

NOTE: assumes any data with off-sample timing has been corrected with *shift_to_sample()*. If not, the resulting traces may not all contain the correct number of samples (and desired start and end times).

Parameters

- **trace** (*obspy.Trace* object) – Trace to be upsampled.
- **upfactor** (*int*) – Factor by which to upsample the data in trace.

Returns
out – Upsampled trace.

Return type
obpsy.Trace object

`quakemigrate.util.wa_response`(*convert='DIS2DIS'*, *obspy_def=True*)

Generate a Wood Anderson response dictionary.

Parameters

- **convert** (*{'DIS2DIS', 'VEL2VEL', 'VEL2DIS'}*) – Type of output to convert between; determines the number of complex zeros used.
- **obspy_def** (*bool, optional*) – Use the ObsPy definition of the Wood Anderson response (Default). Otherwise, use the IRIS/SAC definition.

Returns
WOODANDERSON – Poles, zeros, sensitivity and gain of the Wood-Anderson torsion seismograph.

Return type
dict

PYTHON MODULE INDEX

q

INDEX

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X