GI version 0.1

Documentation

Andreas Fichtner

Department of Earth Sciences, ETH Zurich, Switzerland

Contents

Foreword

`GI` is a code package for the computation of inter-station correlations, effective forward models for correlations functions, and sensitivity kernels for wave field sources and Earth structure. It is written in Python.

`GI` is deliberately simplistic. It operates under the assumption that the Earth is two-dimensional, acoustic, homogeneous and infinitely extended. Thus, it is made to understand basic physics.

`GI` is not a black box. It does not perform any checks on the meaningfulness of the input parameters. `GI` is not optimised or paralellised at all. Some parts of the code package may be very slow slow.

This documentation explains the input files of `GI` and the different code elements that may be used to compute correlation functions, sensitivity kernels, effective media, etc. Additional information is available in the form of numerous comments in the programme code itself.

Zurich and Stanford, Spring 2016 *Andreas Fichtner*

Chapter 1
Structure and functionality of the programme package

1.1 Programming language and packages

`GI` is written entirely in Python. This language was chosen in order to ensure that (i) no commercial software is needed, and (ii) also relatively unexperienced programmers can modify the code and adapt it to their needs. Clearly, `GI` is not designed to be highly performant. The following standard Python packages are needed to run `GI`: matplotlib, numpy, time.

1.2 Directory structure and basic functionality

Most of the code, in the form of Python programmes, is located directly in the main directory. The following is a list of codes with a very brief description of their purpose:

1. `correlation_function.py`: Compute and plot an inter-station correlation function based on the power-spectral density ditribution of the sources.

2. `correlation_field.py`: Compute snapshots of the correlation wavefield for specific times or make a movie from a sequence of snapshots.

3. `source.py`: Compute and plot the spatial and frequency distributions of the sources.

4. `green.py`: The 2D Green function for a homogeneous, acoustic full space.

5. `correlation_random.py`: Compute inter-station correlations based on the summation of random wavefields.

6. `processing.py`: Apply various processing schemes during the computation of correlations based on random wavefields.

7. `correction_factors.py`: Compute source and propagation correctors for a specific processing scheme.

8. `correlation_effective.py`: Compute the effective correlation function based on previously computed source and propagation correctors.

9. `kernels.py`: Compute Fréchet kernels for sources and structure.

10. `adsrc.py`: Compute a frequency-domain adjoint source needed for the computation of Fréchet kernels.

The directory `PLOT` contains several pieces of code to visualise output written by some of the codes mentioned above. A brief listing:

1. `correctors.py`: Plot source and propagation correctors.

2. `correlations.py`: Plot raw, processed and effective correlation functions.

3. `earthquakes.py`: Plot the spatial and temporal distribution of the transient, point-localised sources (acoustic ‘earthquakes’).

All output is written to the `OUTPUT` directory. `GI` is not very flexible here, just to keep the code as simple as possible.

Chapter 2
Input files

All input files are located in the directory `INPUT`. The following is a brief description of their content.

2.1 `setup.txt`

This file contains the basic geometric setup, the medium properties and the properties of the sources. The variables `xmin`, `xmax` and `dx` are the boundaries of the computational domain in x-direction in km and the grid point spacing in km, respectively. The variables `ymin`, `ymax` and `dy` are the same in y-direction. The grid spacing should be sufficiently fine to ensure that space integrals are approximated accurately. `GI`does not check if the spacing fine enough.

The acoustic medium is described in terms of its density `rho` [kg/m^3], velocity `v` [m/s], and Q.

The spatial source distribution is given by the parameter `type`. The various options are listed in the function `space_distribution` in the source code `source.py`. The natural source spectrum (e.g. the spectrum of the noise sources) `natural` and the instrument response `instrument` are implemented in the function `frequency_distribution`, also in the source code `source.py`. Possible values for these parameters are listed there. New options are likely to be added from time to time.

2.2 `receivers.txt`

The number of positions of the receivers in km are listed in this input file. As in all other input files, the format of this file is not allowed to change. Specifically, blank lines should neither be added or deleted.

2.3 `correlation_field_and_kernels.txt`

This file contains parameters needed for the computation of correlation functions and sensitivity kernels based on the power-spectral density distribution of the sources. The frequency band [Hz] extends from `fmin-fwidth` to `fmax+fwidth`, where `fwidth` is the width of the taper that drops off linearly for frequencies lower than `fmin` and higher than `fmax`. The frequency increment used in the inverse Fourier transforms is given by `df`.It is up to the user to choose `df` sufficiently small.

The time variables `tmin`, `tmax` and `dt` denote the minimum and maximum times after the inverse Fourier transform. The time increment is `dt`.

2.4 `ensemble_correlation.txt`

This input file contains parameters describing the computation of correlation functions based on a random wavefield, i.e. via the explicit computation of (noise) traces originating from sources with random phase. This task is performed by the programme code `correlation_random.py`. The input parameter `Nwindows` is the number of time windows. Inter-station correlations for these individual time windows are averaged to form the final ensemble correlation. `Twindows` is the length of these windows in s.

As mentioned above, the random wavefield is excited by sources with random phase. Thus, in principle, each realisation is different. To ensure repeatability, as specific `seed` for the random number generator can be chosen. This can be any integer number. Using the same seed will produce the same results.

2.5 `earthquake_catalogue.txt`

In the computation of correlations based on the actual random wavefield, performed by `correlation_random.py`, transient deterministic sources, i.e. earthquake equivalents, may be added. This input file lists the number of earthquakes (`Neq`), their origin time in s (`t*`), *x*- and *y*-coordinates (`x*`, `y*`) in km, and source strength (`m*`) in N^2s/m^6.

2.6 `processing.txt`

All available processings that may be applied to compute processed correlations are listed in this file. Processed correlations can be computed with the code `correlation_random.py`, which computes correlations explicitly from single station traces for individual time windows.

The bandpass that may also be used as source and instrument spectrum (see `setup.txt` above) is described by its cutoff-frequencies `fmin` and `fmax`, and by the width of the linear taper `fwidth`. The remaining parameters can be set to either 0 (processing is not applied) or 1 (processing is applied). Their meaning is as follows: `onebit`: apply one-bit normalisation to raw recordings, `rms_clip`: clip raw recordings above their rms value, `whiten`: apply spectral whitening to raw traces, `causal_acausal_average`: compute the average of the causal and acausal branches, `correlation_normalisation`: normalise correlations for each time window by their maximum, `phase_weighted_stack`: compute a phase-weighted stack, instead of a linear stack. The processing is implemented by the source code `processing.py`.

2.7 `windows.txt`

This file contains a list of measurement windows needed for the computation of Fréchet kernels. The first datum is the number of windows. It is followed by a listing of the windows, each line containing the left and right boundaries of a window in s, and the width of the cosine taper in s that determines the sharpness of the window. The file is read by `adsrc.py`, which is called by `kernels.py`.

Chapter 3
Task descriptions

This chapter is split in two parts: The first one is about computations using actual random wavefields, excited by sources with random phase. This includes, for instance, the computation of correlation functions by summing (processed) correlations for many time windows, and the computation of source and propagation correctors. The second part is about deterministic computations based on correlation wavefields excited by a deterministic power-spectral density distribution of sources. This includes the computation of ensemble correlations and sensitivity kernels.

3.1 Random wavefield computations

3.1.1 Computing inter-station correlations from random wavefields

The function `correlation_random` computes raw and processed correlation functions through the summation of correlations for individual time windows. The individual time window correlations are computed by modelling a wavefield excited by sources with random phase.

In the `setup.txt` file, the source distribution can be set via the parameter `type`. It can be visualised using the functions contained in `source.py`. The processing applied to the raw synthetics or raw correlations is speficied in `processing.txt`. The input file `ensemble_correlation.txt` contains the length of the individual time windows `Twindow` and the number of time windows `Nwindow`. To ensure reproducibility of the random simulations, the random seed can be specified via the parameter `seed`. It can be any integer number.

Transient, deterministic sources mimicking earthquakes may be added by editing the input file `earthquake_catalogue.txt`.

Output in the form of individual correlation functions, i.e. for the individual time windows, is written to `OUTPUT/correlations_individual/`. This is needed for the computation of effective correlations using `correlation_effective` (see below).

3.1.2 Computing source and propagation correctors

Source and propagation correctors for synthetic correlations can be computed with `correction_factors`, which calls `correlation_random`. When the parameter `save` is set to `1`, the correction factors are saved to `OUTPUT/correctors`, and the ensemble correlations for all station pairs are saved to `OUTPUT/correlations`.

3.1.3 Computing effective correlations

After computing and storing correlations for individual time windows with `correlations_random` and correction factors with `correction_factors`, effective correlations can be computed with `correlation_effective`. When the parameter `save` is set to `1`, the effective correlations are saved to `OUTPUT/correlations`.

3.1.4 Plotting source and propagation correctors

The source file `PLOT/correctors.py` contains various functions to plot source correctors (`source`), propagation correctors (`propagation`) and the frequency-dependent geometric spreading of the effective Green function (`geometric_spreading`). They that correctors have previously been computed and stored.

3.1.5 Plotting raw, processed and effective correlations

Provided that these have been computed, `PLOT/correlations.py` can be used to visualise raw, processed and effective correlations.

3.1.6 Plotting the spatial and temporal distribution of earthquakes

This can be done using `PLOT/earthquakes.py`.

3.2 Deterministic wavefield computations

3.2.1 Computing inter-station correlation functions for a given power-spectral density distribution

To compute an inter-station correlation function, run `correlation_function.py`. It requires input from `setup.txt`, `correlation_field_and_kernels.txt`, `receivers.txt`, and `processing.txt` in case the bandpass is used for the natural source spectrum or the instrument response. The function returns the time- and frequency-domain correlation functions, and plots them when the parameter `plot` is set to `1`.

3.2.2 Computing snapshots and movies of the interferometric wavefield

The code `correlation_field.py` contains two functions: `snapshot` to compute the interferometric wavefield for a specific time, and `movie` to save .png files for a sequence of snapshots.

Both functions read input from the same files as texttt_correlation_function.py, described above. To speed up computations, `snapshot` uses a very simplistic multi-grid method. In the first stage, the wavefield is computed only on a coarse grid, e.g. every 5 grid points when the parameter `mg_level` is set to `5`. In the second stage, the remaining grid points around the previously visited grid points are filled up, provided that the wavefield on the coarse-grid point is larger than `mg_tol` times the maximum on the coarse grid. `snapshot` returns the wavefield snapshot. The snapshot can be plotted and saved when the parameters `plot` and `save` are set to `1`, respectively.

The function `movie` operates similar to `snapshot`, the only difference being that it does not employ a multi-grid method. Internally, `movie` computes a three-dimensional array containing the wavefield for all the snapshot times provided in the input array `time_axis`. This means that the function is rather slow, and it requires large memory. So, be a bit careful.

3.2.3 Computing source kernels

Before computing Fréchet kernels for sources, a time-domain correlation function must be computed by running `correlation_function.py` (see above). The time-domain correlation and the time axis are input for the function `source_kernel` in `kernels.py`. The former computes and plots a source kernel for the measurement windows defined in the input file `windows.txt`.

Internally, `source_kernel` first computes the frequency-domain adjoint source for a specific type of measurement by calling `adsrc.py`. The adjoint source then acts as a frequency-domain multiplier with a product of two Green functions. The kernel is then computed by integrating over all frequencies, which is equivalent to assuming that the power-spectral density of the sources is frequency-independent, i.e. only dependent on the space coordinate.