ShakerMaker Documentation

Release 1.0

Jose Antonio Abell, Jorge Crempien and Matias Recabarren

Nov 12, 2020

Contents:

ShakerMaker is intended to provide a simple tool allowing earthquake engineers and seismologists to easily use the frequency-wavenumber method (hence FK) to produce ground-motion datasets for analysis using the Domain Reduction Method (DRM). DRM motions are stored directly into the H5DRM format.

FK method, the core of *ShakerMaker*, is impemented in fortran (originally from http://www.eas.slu.edu/People/LZhu/home.html with several modifications), and interfaced with python through f2py wrappers. Classes are built on top of this wrapper to simplify common modeling tasks such as crustal model specification, generation of source faults (from simple point sources to full kinematic rupture specifications), generating single recording stations, grids and other arrays of recording stations and stations arranged to meet the requirements of the DRM. Filtering and simple plotting tools are provided to ease model setup. Finally, computation of motion traces is done by pairing all sources and all receivers, which is parallelized using MPI. This means that *ShakerMaker* can run on simple personal computers all the way up to large supercomputing clusters.

Contents

CHAPTER 1

Dependencies

We stand on the shoulder's of giants. `ShakerMaker` depends on the following python modules (most) to work its magic.

- *mpi4py* (optional but highly recommended for parallel computing of the response)
- *h5py*
- *numpy*
- *scipy*
- *f2py*
- *matplotlib* (optional, for plotting)

Get them all with *pip*:

```
sudo pip install mpi4py h5py numpy scipy matplotlib tqdm
```

CHAPTER 2

Installation

For now, only though the git repo:

```
git clone git@github.com:jaabell/ShakerMaker.git
```

Use the *setup.py* script, using setuptools, to compile and install:

```
sudo python setup.py install
```

If you dont' have *sudo*, you need:

```
sudo python setup.py install --user
```

Build the documenation with:

```
sudo python setup.py build_sphinx
```

CHAPTER 3

Quickstart usage

Using *ShakerMaker* is simple. You need to specify a `shakermaker.crustmodel` (choose from the available predefined models or create your own), a `fkdrm.SourceModel` (from a simple `fkdrm.Receivers.PointSource` to a complex fully-customizable extended source with `fkdrm.Receivers.MathyFaultPlane`) and, finally, a `fkdrm.Receiver` specifying a place to record motions (and store them in memory or text format).

In this simple example, we specify a simple strike-slip (strike=30, clockwise from north) point source at the origin and a depth of 4km, on a custom two-layer crustal model, and a single receiver 5km away to the north.

Start by importing the needed components:

```
#./doc/examples/example_0_quick_example.py
from shakermaker.shakermaker import ShakerMaker
from shakermaker.crustmodel import CrustModel
from shakermaker.pointsource import PointSource
from shakermaker.faultsource import FaultSource
from shakermaker.station import Station
from shakermaker.stationlist import StationList
from shakermaker.tools.plotting import ZENTPlot
```

Create a new two-layer CrustModel:

```
crust = CrustModel(2)

#Slow layer
Vp=4.000                        # P-wave speed (km/s)
Vs=2.000                        # S-wave speed (km/s)
rho=2.600                       # Density (gm/cm**3)
Qp=10000.                       # Q-factor for P-wave
Qs=10000.                       # Q-factor for S-wave
thickness = 1.0          # Self-explanatory
crust.add_layer(thickness, Vp, Vs, rho, Qp, Qs)

#Halfspace
Vp=6.000
```

```
Vs=3.464
rho=2.700
Qp=10000.
Qs=10000.
thickness = 0   #Zero thickness --> half space
crust.add_layer(thickness, Vp, Vs, rho, Qp, Qs)
```

Specify source location (xyz coordinates: x is north, y is east, z is down, see [*Coordinate system in shakermaker*]) and strike, dip and rake angles:

```
source = PointSource([0,0,4], [90,90,0])
fault = FaultSource([source], metadata={"name":"single-point-source"})
```

Specify receiver location (xyz as before):

```
s = Station([0,4,0],metadata={"name":"a station"})
stations = StationList([s], metadata=s.metadata)
```

These are fed into the FKDRM model class:

```
model = ShakerMaker(crust, fault, stations)
```

Which is executed:

```
model.run()
```

Results at the station can be readily visualized using the utility function `Tools.Plotting.ZENTPlot()`:

```
ZENTPlot(s, xlim=[0,60], show=True)
```

This script can be executed from the command line as follows:

```
python3 example_0_quick_example.py
```

Yielding:

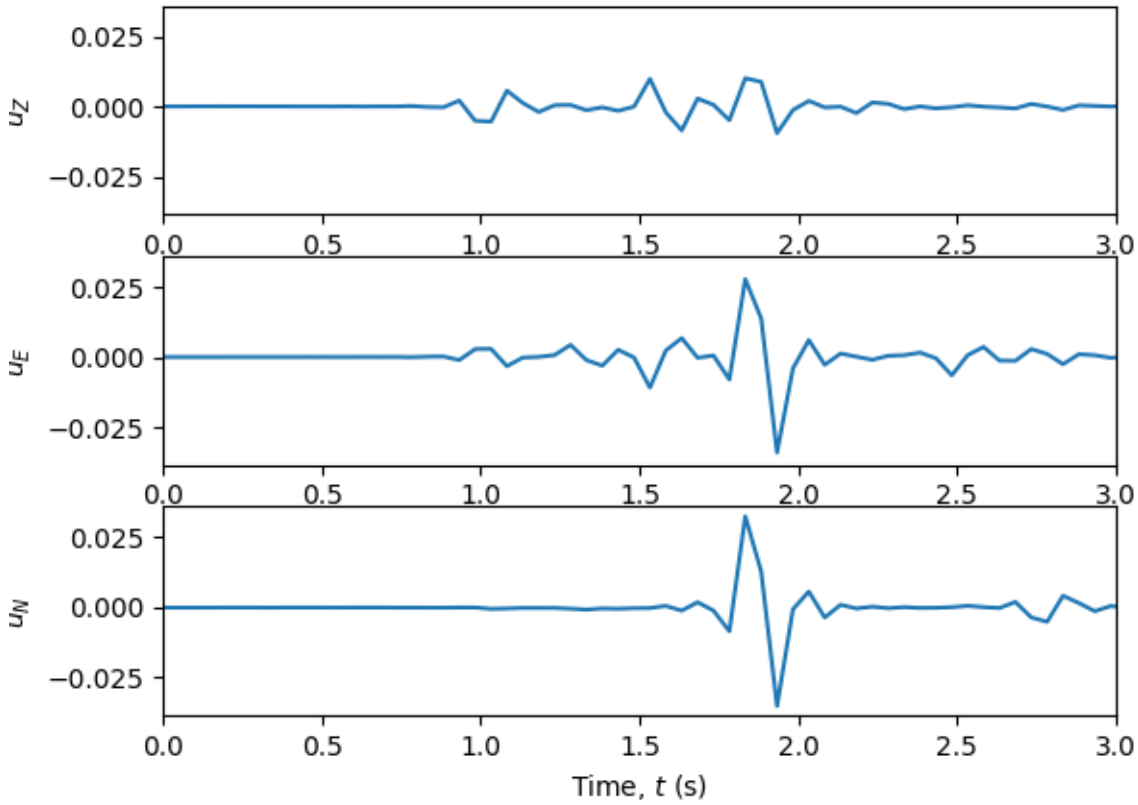


Fig. 3.1: Response at receiver station. No source time function was specified, so this is just a Greens function (impulse response) of the medium.

CHAPTER 4

Parallel computation capabilities

To make use of multi-processor architechtures and computer clusters, *ShakerMaker* is parallelized internally using *mpi4py*. Running a script in parallel processing mode is simply done using an *mpirun* call, no modifications to the script are necessary:

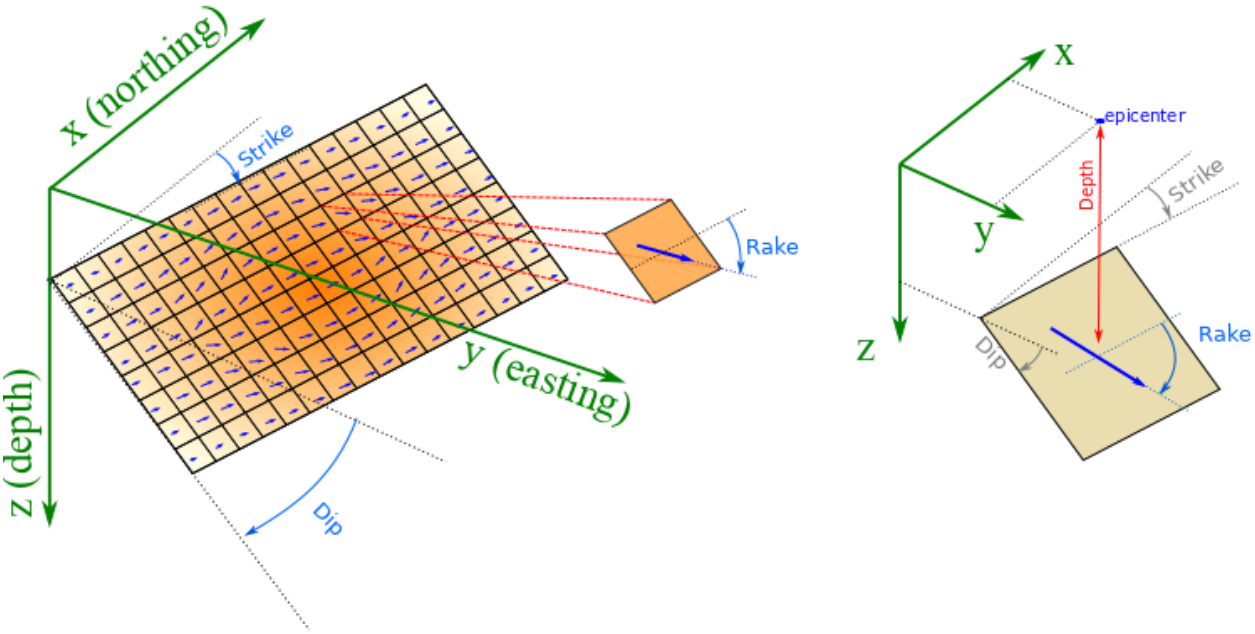
```
mpirun -np 10 python <script-name>
```

CHAPTER 5

Documentation

5.1 ShakerMaker

5.1.1 Coordinate system in shakermaker



ShakerMaker defines it's coordinate system with x positive towards the north, y positive towards the east and z positive downwards.

Strike is defined clockwise from the north, dip is measured from the horizontal, and rake increases in the down-dip direction.

5.1.2 ShakerMaker main class

class shakermaker.shakermaker.**ShakerMaker**(*crust*, *source*, *receivers*)
This is the main class in ShakerMaker, used to define a model, link components, set simulation parameters and execute it.

Parameters

- **crust** (CrustModel) – Crustal model used by the simulation.
- **source** (FaultSource) – Source model(s).
- **receivers** – Receiver station(s).

run(*dt=0.05*, *nfft=4096*, *tb=1000*, *smth=1*, *sigma=2*, *taper=0.9*, *wc1=1*, *wc2=2*, *pmin=0*, *pmax=1*, *dk=0.3*, *nx=1*, *kc=15.0*, *writer=None*)
Run the simulation.

Arguments: :param sigma: Its role is to damp the trace (at rate of exp(-sigma*t)) to reduce the wrap-arround. :type sigma: double :param nfft: Number of time-points to use in fft :type nfft: integer :param dt: Simulation time-step :type dt: double :param tb: Num. of samples before the first arrival. :type tb: integer :param taper: For low-pass filter, 0-1. :type taper: double :param smth: Densify the output samples by a factor of smth :type smth: double :param wc1: (George.. please provide one-line description!) :type wc1: double :param wc2: (George.. please provide one-line description!) :type wc2: double :param pmin: Max. phase velocity, in 1/vs, 0 the best. :type pmin: double :param pmax: Min. phase velocity, in 1/vs. :type pmax: double :param dk: Sample interval in wavenumber, in Pi/x, 0.1-0.4. :type dk: double :param nx: Number of distance ranges to compute. :type nx: integer :param kc: It's kmax, equal to 1/hs. Because the kernels decay with k at rate of exp(-k*hs) at w=0, we require kmax > 10 to make sure we have have summed enough. :type kc: double :param writer: Use this writer class to store outputs :type writer: StationListWriter

5.2 CrustModel

class shakermaker.crustmodel.**CrustModel**(*nlayers*)
Define a 1-D layered crust model.

Parameters **nlayers** (*int*) – Number of layers that the new CrustModel will have.

Initialize the crust model with how many layer it has:

from shakermaker.crustmodel import CrustModel model = CrustModel(2)

See :mod:shakermaker.cm_library for some pre-defined models.

a

add_layer(*d*, *vp*, *vs*, *rho*, *qp*, *qs*)
Add a new layer to the model.

This function must be called as many times as layers were specified when the CrustModel was defined. Layer are stacked from top (surface) to bottom.

Parameters

- **d** (*double > 0*) – Thickness of new layer. d=0 defines an infinite half-space layer. The last layer, and only that layer, must can be a half-space.
- **vp** (*double > 0*) – Compression-wave speed (V_p) of new layer.
- **vs** (*double*) – Shear-wave speed (V_s) of new layer.

- **rho** (*double > 0*) – Mass density (ρ) of the new layer.
- **qp** (*double > 0*) – Q-factor (Q_P) for compression-waves for the new layer.
- **qs** (*double > 0*) – Q-factor (Q_S) for shear-waves for the new layer.

Example:

```
#This is a two-layer model
#
# ---------------------------------------- surface (layer 1)    ---
# vp  = 1.5 (km/s)     vs = 0.8 (km/s)                           |
# Qp  = 50  (    )     Qs = 100 (    )                          500m
# rho = 2.1 (gr/cm^3)  d  = 0.5 (km)                             |
# ---------------------------------------- halfspace (layer 2)  ---
# vp  = 3.2 (km/s)     vs = 1.6 (km/s)                           |
# Qp  = 80  (    )     Qs = 200 (    )                           v
# rho = 2.8 (gr/cm^3)  d  = 0   (km)                             z+
#
model = CrustModel(2)
model.add_layer(0.5, 1.5, 0.8, 2.1, 50., 100.)
model.add_layer(0  , 3.2, 1.6, 2.8, 80., 200.)
```

Note: **Must** use the units of `km` for length, `km/s` for speed, and `gr/cm^3` for density.

b

d

get_layer(*z*, *tol=0.01*)

Split the layer at depth `z`.

Parameters

- **z** (*double*) – Depth for which layer number is needed
- **tol** (*double*) – Tolerance for detection

Returns Index of layer

Return type int

modify_layer(*layer_idx*, *d=None*, *vp=None*, *vs=None*, *rho=None*, *gp=None*, *gs=None*)

Modify the properties of layer number `k`.

Parameters

- **k** (*int*) – Layer to modify.
- **d** (*double >= 0*) – New thickness of layer-`k`. `d=0` defines an infinite half-space layer.
- **vp** (*double >= 0*) – New compression-wave speed (V_p) of layer-`k`.
- **vs** (*double >= 0*) – New shear-wave speed (V_s) of layer-`k`.
- **rho** (*double >= 0*) – New mass density (ρ) of the layer-`k`.
- **qp** (*double >= 0*) – New Q-factor (Q_P) for compression-waves for the layer-`k`.
- **qs** (*double >= 0*) – New Q-factor (Q_S) for shear-waves for the layer-`k`.

Positive values of parameters means change that parameter, zero values (default) leave that property unaltered.

Example:

```
#Change Vs for layer 2.
model.modify_layer(2, vs=2.5)
```

Note: **Must** use the units of `km` for length, `km/s` for speed, and `gr/cm^3` for density.

nlayers

properties_at_depths(*z*, *kind='previous'*)
Return (interpolated) properties at depths specified by vector `zz`.

Internally uses `scipy.interpolate.interp1d` to do interpolation with `kind='previous'`.

Parameters

- **zz** (*double or np.array of shape (N,)*) – Positions at which to interpolate.
- **kind** (*string*) – Kind of interpolation to use. See options in `scipy.interpolate.interp1d`.

qa

qb

rho

split_at_depth(*z*, *tol=0.01*)
Split the layer at depth `z`.

Parameters

- **z** (*double*) – Depth at which to split.
- **tol** (*double*) – Split tolerance. Will not split if there is a layer interface within `z-tol < z < z + tol`.

5.2.1 Predefined CrustModel Library

A small library of pre-defined crustal models collected over the years.

AbellThesis

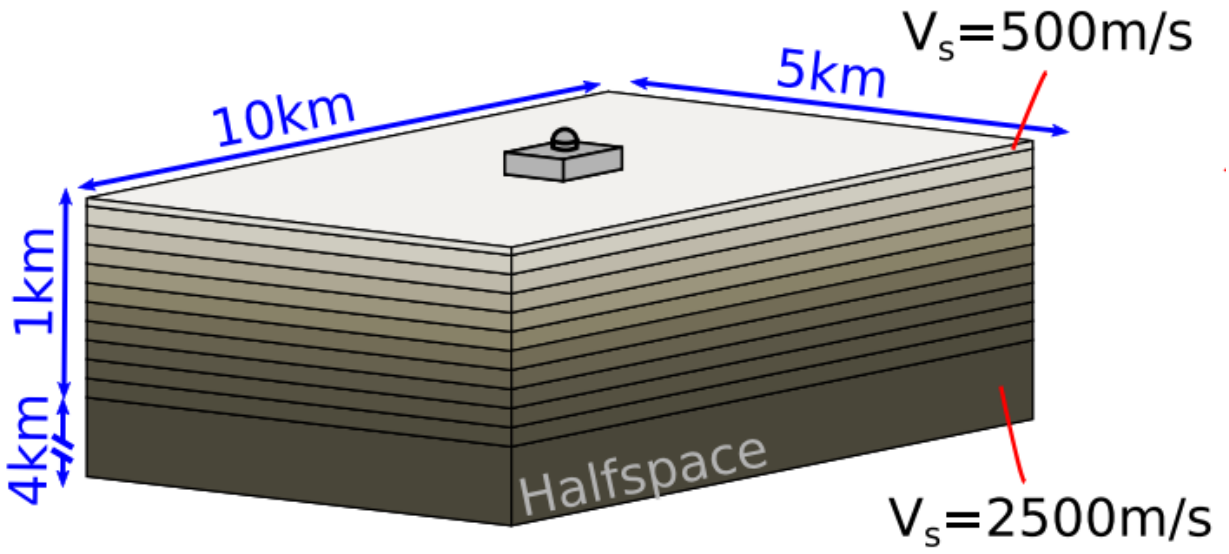
`shakermaker.cm_library.AbellThesis.`**AbellThesis**(*split=1*)
Crustal model in Jose Abell's PhD thesis and paper

Note: Zero anelastic attenuation has been approximated using high values for the Q-factor.

Arguments: :param split: The layering can be subdivided if needed. :type split: int

Returns: :returns: `shakermaker.CrustModel`

References: + Abell, J. A. (2016). Earthquake-Soil-Structure Interaction Modeling of Nuclear Power Plants for Near-Field Events. University of California, Davis. + Abell, J. A., Orbović, N., McCallen, D. B., & Jeremic, B. (2018). Earthquake soil-structure interaction of nuclear power plants, differences in response to 3-D, 3 × 1-D, and 1-D excitations. Earthquake Engineering and Structural Dynamics, 47(6), 1478–1495. https://doi.org/10.1002/eqe.3026



SCEC LOH

shakermaker.cm_library.LOH.**SCEC_LOH_1**()
This is an shakermaker Crustal Model for problem LOH.1 from the SCEC test suite.

This is a slow layer over a half-space with no attenuation.

Note: Zero anelastic attenuation has been approximated using high values for the Q-factor.

Reference: + Steven Day et al., Tests of 3D Elastodynamic Codes: Final report for lifelines project 1A01, Pacific Eartquake Engineering Center, 2001

shakermaker.cm_library.LOH.**SCEC_LOH_3**()
This is an shakermaker Crustal Model for problem LOH.3 from the SCEC test suite.

This is a slow layer over a half-space with attenuation.

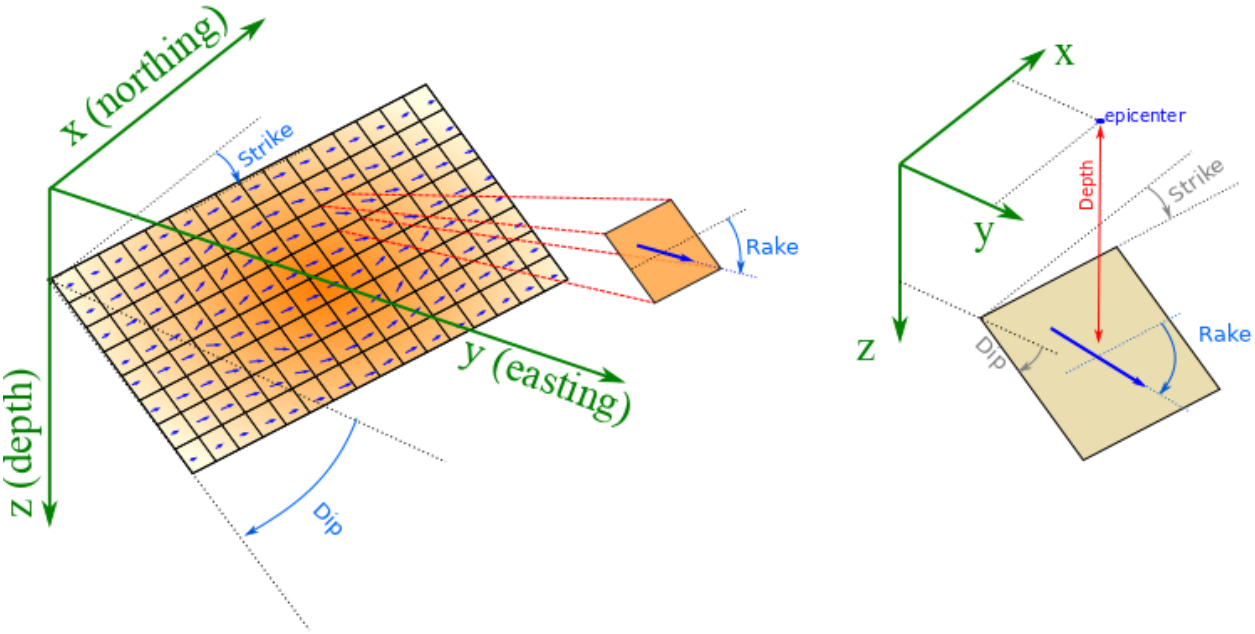
Reference: + Steven Day et al., Tests of 3D Elastodynamic Codes: Final report for lifelines project 1A01, Pacific Eartquake Engineering Center, 2001

5.3 Point sources and faults

ShakerMaker defines it's coordinate system with x positive towards the north, y positive towards the east and z positive downwards.

Strike is defined clockwise from the north, dip is measured from the horizontal, and rake increases in the down-dip direction.

Faults are specified using the `FaultSource` which are just lists of sub-faults which are of the `PointSource` type. Faults can have arbitrary shape and complexity.



5.3.1 PointSource

class shakermaker.pointsource.**PointSource**(*x*, *angles*, *stf=<shakermaker.stf_extensions.dirac.Dirac object>*, *tt=0*)

Bases: `object`

A source that is a point.

Defined by its position and spatial orientation, this source can also be given a trigger time and a source time function (to be convolved after the Green's function is computed).

Parameters

- **x** (*numpy array (shape (3,))*) – Position of the source in xyz coordinates.
- **anlges** – Orientation of the fault `angles = [strike, dip rake]` in degrees.
- **tt** (*double*) – trigger time for the fault (s)
- **stf** (`fkdrm.fkdrmBase.SourceTimeFunction`) – source time function to convolver

angles
Numpy array with the (strike,dip,rake) angles of the source fault plane in degrees

stf
The source time-function to be convolved with.

tt
Scalar trigger time

x
Numpy array with the (x,y,z) coordinates of the source

5.3.2 FaultSource

class shakermaker.faultsource.**FaultSource**(*sources*, *metadata*)
Bases: object

A fault is a collection of point-sources

If you want to have more than one point-source in space generating motions, you use this class.

Parameters **sources** – A list of PointSources

Example:

```
#Two strike-slip sources at z=1.0 and 1.5 (km)
ps1 = PointSource([0, 0, 1],[0, 90, 0])
ps2 = PointSource([0, 0, 1.5],[0, 90, 0])

fault = FaultSource([ps1, ps2])
```

metadata
Source metadata, such as fault name.

nsources
Number of sub-faults

5.4 SourceTimeFunction

Ground motion responses (seismic traces) in *ShakerMaker* are computed by convolving the medium's Green's function evaluated at the receiver point with the source time function. This convolution is done numerically using `scipy.signal.convolve()`. Therefore, it is most convenient to specify source time functions as slip rate functions, with the resultant traces corresponding to the ground velocity history at the point of interest.

Note: **T.L.D.R.** These are all slip-rate functions. Treat them as such.

5.4.1 Dirac

class shakermaker.stf_extensions.dirac.**Dirac**
Bases: shakermaker.sourcetimefunction.SourceTimeFunction

The Dirac delta

Generate Green's functions using a Diract delta source-time-function.

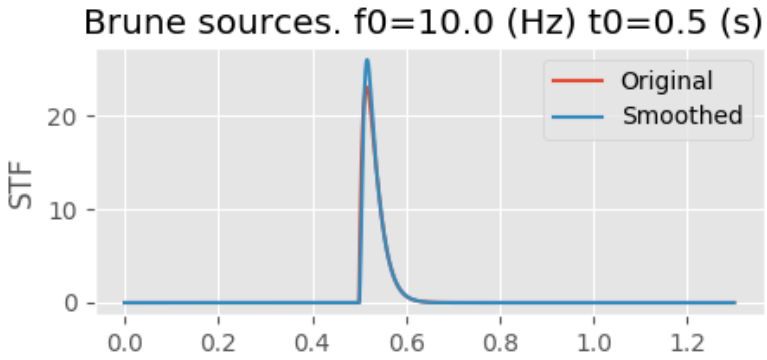
5.4.2 Brune

class shakermaker.stf_extensions.brune.**Brune**(*slip=1.0*, *f0=0.0*, *t0=0.0*, *dsigma=0.0*, *M0=1.0*, *Vs=0.0*, *smoothed=False*)
Bases: shakermaker.sourcetimefunction.SourceTimeFunction

The Brune Source Time Function

Implements the classic STF as a slip rate function

$$f_s(t) = \Delta \cdot \omega_w 0^2 \cdot (t - t_0) \cdot \exp\{-w_0(t - t_0)\} \quad \text{for } t \geq t_0$$



Where Δ is the total slip across the fault, $w_0 = 2\pi f_0$ and f_0 is the corner-frequency defined by:

$$f_0 = 4.9 \times 10^6 V_s \left(\frac{\Delta\sigma}{M_0}\right)^{1/3}$$

V_s is the local shear-wave speed in km/s, M_0 is the seismic-moment in dyne-cm, and $\Delta\sigma$ is the stress-drop in bars.

The source is defined by the slip (`slip`) and the fault trigger time (`t0`) and either of: **(i)** the corner frequency directly `f0` or **(ii)** the stress drop `dsigma`, seismic moment `m0` and local shear-wave speed `Vs`.

Note: The `t0` parameter displaces the STF in its own time vector, it is more convenient to use the point source's trigger time‘‘tt‘‘ to specify the rupture process.

Parameters

- **slip** (*double*) – Total slip across the fault.
- **f0** (*double*) – Corner frequency.
- **t0** (*double*) – Trigger time.
- **dsigma** (*double*) – Stress-drop.
- **M0** (*double*) – Seismic moment.
- **Vs** (*double*) – Local shear-wave speed.
- **smoothed** (*bool*) – Use a smoothed version of the source function.

5.4.3 Discrete

class shakermaker.stf_extensions.discrete.**Discrete**(*data*, *t*)

Bases: shakermaker.sourcetimefunction.SourceTimeFunction

Specify the STF using discrete values at your discretion.

Parameters

- **data** (*numpy vector shape (Nt, 0)*) – STF values
- **t** (*numpy vector shape (Nt, 0)*) – STF time-values. Must start and end at 0, can be un-evenly spaced.

Note: If the supplied STF specification is un-evenly spaced it gets interpolated to the simulation time-step before numerical convolution.

Example:

```
t = np.array([0,0.01,0.02,0.1,0.2])
slip = np.array([0,0.2,1,0.4,0])
stf = Discrete(data,t)
```

5.5 Station

Contents

5.5.1 Station Class

class shakermaker.station.**Station**(*x*, *internal=False*, *metadata={}*)

Bases: object

This simple receiver stores response in memory.

Internally, numpy arrays are used for storage. Optional parameters allow filtering of the response before outputting, althoiugh it is always stored raw (unfiltered), therefore the user can experiment with different filtering settings.

Parameters

- **x** (*numpy array (3,)*) – xyz location of the station.
- **metadata** – metadata to store with the station

get_response()

Return the recorded response of the station.

Parameters

- **do_filter** (*bool*) – Will/won't filter if filter parameters have been set. (Most useful to disable filtering before return)
- **interpolate** (*bool*) – If True then will interpolate to a new time vector before filtering
- **interpolate_t** (*numpy array (Nt,)*) – New time vector (its best if this vector spans or encompasses the old vector... otherwise artifacts will ensue)

Returns Z (down), E (east), N (north), t (time) response of the station.

Retval tuple containing numpt arrays with z, e, n, t reponse (shape (Nt,))

Example:

```
z,e,n,t = station.get_response()
```

.

5.5.2 StationList Class

class shakermaker.stationlist.**StationList**(*stations*, *metadata*)

Bases: shakermaker.station.StationObserver

This is a list of stations.

Parameters

- **stations** (list containing Station) – A list of Stations
- **metadata** – metadata to store with the station list

Example:

```
sta1 = Station([20,20,0])
sta2 = Station([20,40,0])

stations = StationList([sta1, sta2])
```

.

5.5.3 DRMBox

class shakermaker.sl_extensions.DRMBox.**DRMBox**(*pos*, *nelems*, *h*, *metadata={}*, *azimuth=0.0*)

Bases: *shakermaker.stationlist.StationList*

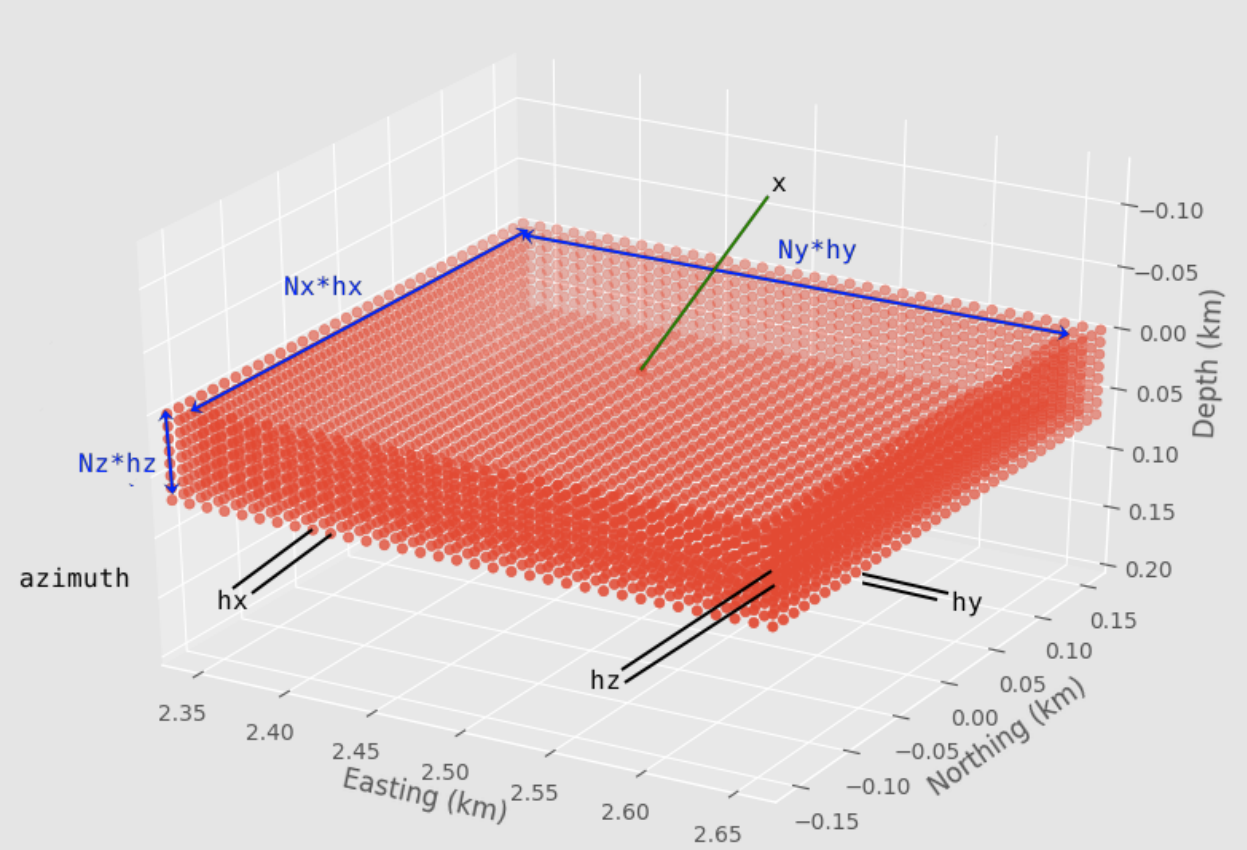
A class to generate receiver layout useful in DRM.

Parameters

- **pos** (*(numpy array (3,))*) – Center point of the DRM box in xyz coordinates.
- **nelems** (*(int)*) – Number of elements (stations) in each direction. Nelem = [Nx, Ny, Nz]
- **h** (*(double)*) – Spacings in each direction h = [hx, hy, hz]
- **azimuth** (*(double)*) – Azimuthal orientation of the box.

Note: Side lengths of the DRM box are [Nx*hx, Ny*hy, Nz*hz] up to the interior boundary of the box. Exterior boundary has side lengths: [(Nx+2)*hx, (Ny+2)*hy, (Nz+1)*hz]

.


x
Nx*hx
Ny*hy
Nz*hz
azimuth
hx
hy
hz
2.35
2.40
2.45
2.50
2.55
2.60
2.65
Easting (km)
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15
Northing (km)
−0.10
−0.05
0.00
0.05
0.10
0.15
0.20
Depth (km)

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

T

X