CGFD3D Seismic Wave Simulation Program
User Manual

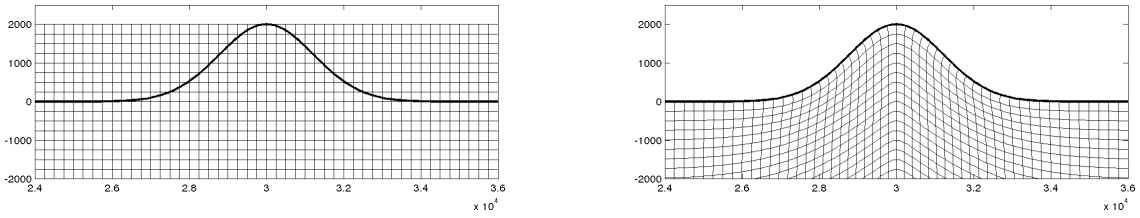
February 27, 2022

Contents

Chapter 1

Generation of Computational Grid

The computational grid is the basis of the grid-based numerical solver. The simplest and most easily generated grid is a Cartesian grid (Figure **??**), whose grid lines are parallel with Cartesian coordinate axes, thus it can be easily generated by giving the triple value: the starting point, number of points and the spacing interval. But the Cartesian grid has difficulty to produce high accurate solution if there is surface topography in seismic wave simulation. For surface topography, a general cuvilinear grid system is more accurate (Figure **??**). Its grid lines conform with the surface topography, thus the surface topography can be accurately represented by the grid. CGFD3D supports both the Cartesian grid for high efficient simulation and also the curvilinear grid for high accurate calculation with surface topography.



(a) Cartesian grid

(b) Curvilinear grid

Figure 1.1: Computational grids

1.1 Set Grid Parameters in Main Par

Listing 1.1: Grid related parameters in .json

```
"grid_generation_method" : {
  "key name" : value
},
"is_export_grid" : 1,
"grid_export_dir"   : "/home/user/prj/output",
```

As shown in List **??**, user needs to set following parameters related to the grid in .json file:

-
- `grid_generation_method`: how the grid is generated. There are current three possible methods:
  - `import`: import previoused generated curvilinear grid, thus no need to re-generate grid for repeated run. select this option and give the folder of the exported curvilinear grid, e.g.:
    `"import" : "/home/user/prj/grid"`

  - `cartesian`: generate Cartesian grid, see next section for details.
  - `layer_interp`: generate general curvilinear grid, see later section for details.
- `is_export_grid`: if the coordiantes of the curvilinear grid are exported for display purpose or reuse for later calculation without re-generateion of the curvilinear grid.
  - 0: do not export,
  - 1: export.
- `grid_export_dir`: set the output dir if `is_export_grid : 1`.

1.2 Generation of Cartesian Grid

To use the Cartesian grid, we can simply set parameters in the .json file as List **??**. The Cartesian grid needs following parametgers:

- `origin`: the x, y and z coordina of the first point,
- `interval`: the x, y and z grid spacing.

Listing 1.2: Example of Using Cartesian Grid in .json

```
"grid_generation_method" : {
    "cartesian" : {
      "origin"  : [0.0, 0.0, -5900.0 ],
      "inteval" : [ 100.0, 100.0, 100.0 ]
    }
},
"is_export_grid" : 1,
"grid_export_dir"   : "/home/user/prj/output",
```

1.3 Generation of General Curvilinear Grid

We implement a simplified curvilinear grid generation method: input several topographic grid interfaces as intermidiate interfaces, then interpolate grid points between these grid interfaces with given number of cells between interfaces, which we name as `layer_interp` method.

Usage Example of `layer_interp` in .json

```
"grid_generation_method" : {
    "layer_interp" : {
      "in_grid_layer_file" : "/home/user/prj/input/test_grid.gdlay",
      "refine_factor" : [ 1, 1, 1 ],
      "horizontal_start_index" : [ 3, 3 ],
      "vertical_last_to_top" : 0
    }
},
"is_export_grid" : 1,
"grid_export_dir"   : "/home/user/prj/output",
```

To use `layer_interp` to generate the curvilinear grid, we should provide following informations in the .json file (List **??**):

- `in_grid_layer_file`: the .gdlay file that represents the curvilinear grid using serveral topographic grid interfaces and number of cells between interfaces. Please see Section **??** of the file format description.

- `refine_factor`: The .gdlay file describes a whole curvinear grid with pre-defined nx, ny, nz, number of grid points, and grid spacings. If we want to use a smaller grid spacing (denser grid) or larger grid spacing (corser grid), we can use this `refine_factor` to change the grid density along different dimensions, e.g.,
  - 1: no resampling;
  - `2` or `3`: increase density by two or three times using interpolating;
  - `-2` or `-3`: minus means downsampling by keeping points every two or three points.

  This parameter must be an integer and cannot be zero.
- `horizontal_start_index` and `vertical_last_to_top`: We can also use a smaller part of the total .gdlay grid in simulation . To do so, we need to tell the program the index of the starting point in the .gdlay to be used. Because CGFD3D uses a z-axis positive upward coordinate system, the free surface is located at the end grid size with large number of index of the third dimension. Thus we specify the staring points for first and second dimensions through `horizontal_start_index`, while end points of the cut grid to that of the .gdlay through `vertical_last_to_top`. E.g.,
  - `"vertical_last_to_top" : 0`: the last layer along 3rd-dim of the cut grid conincides with the free surface in the .gdlay.
  - `"vertical_last_to_top" : 40`: the last layer along 3rd-dim of the cut grid conincides with the last 40 points along 3rd-dim in the .gdlay.

  Note: if used with resampling by `refine_factor`, there should be at least three points (for current DRP/opt MacCormack scheme) left outside the resulting computational grid as the ghost points, thus the `horizontal_start_index` should be set to ensure this condition satisfied. E.g.:
  - If you do not resample, the minimum value of `horizontal_start_index` is 3;
  - If you do two or three times downsampling, the minimum value of `horizontal_start_index` is 6 or 10;
  - If you do two or three times interpolating, the minimum value of `horizontal_start_index` is 2 or 1.

1.4 Format of Grid Layer Interp File (.gdlay)

Listing 1.3: Example of .gdlay file

```
      4
      43     10     10
       1      0      1
   126    106
     -300.00      -300.00     -5484.82
     -200.00      -300.00     -5483.59
     -100.00      -300.00     -5481.34
        0.00      -300.00     -5479.30
      100.00      -300.00     -5478.44
      200.00      -300.00     -5478.95
      300.00      -300.00     -5480.11
.
.
.
```

An example of .gdlay is shown in Listing **??** and Figure **??**. The detailed format of the .gdlay is:

- The first line: the number of grid interfaces (NI, 4 in this example).
- The second line: number of cells of each layer (the space between two grid interfaces is called a layer). There are 43 cells between the 1st and 2nd grid interfaces, 10 between 2nd and 3rd interfaces, 10 between 3rd and 4th interfaces in this example.
- The third line: $NI-1$ int values to set if the grid spacing along the 3rd-dim is equal:
  - 1: is equal, thus the grid spacing of the 3rd-dim is easily calculated by the two points of the grid interfaces in the .gdlay and number of cells of this layer specified in previous line.

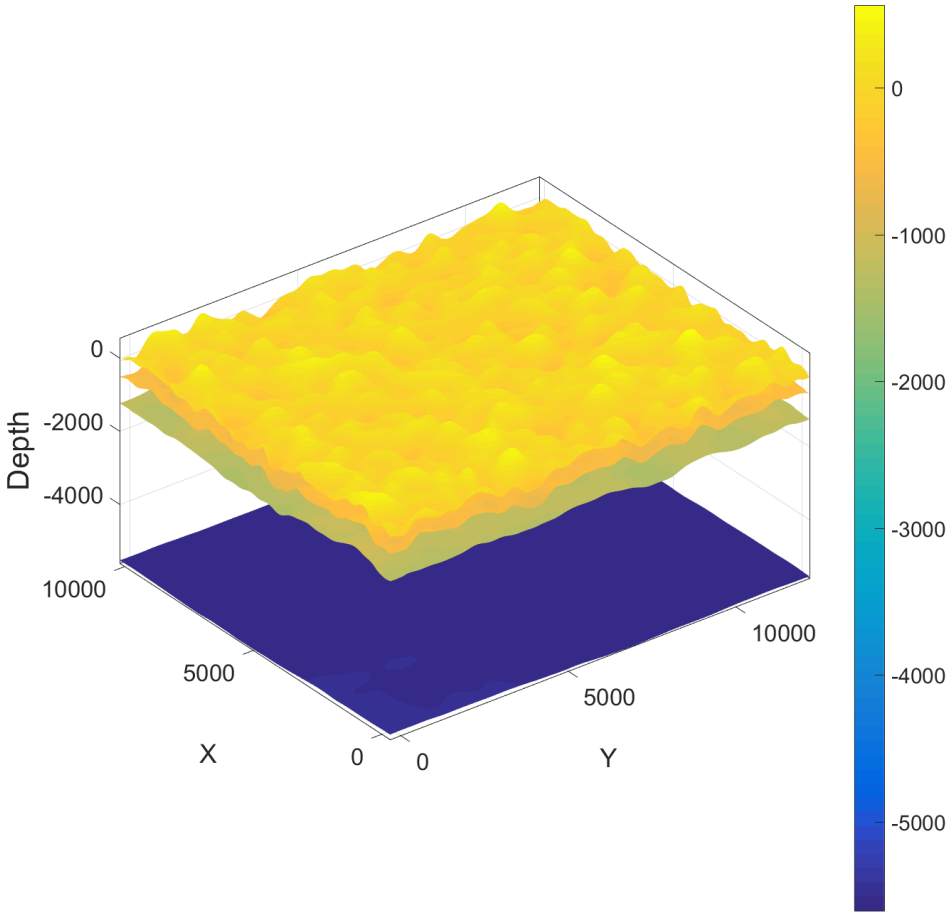


Figure 1.2: Examples of the grid interfaces in the .gdlay file.

  * 0: not equal or should smoothly vary from the grid spacing of the above grid interfaces to that of lower grid interfaces.

  The variation of the grid spacing between grid layers should be as smooth as possible. The vartions of the horizontal grid space is determined by the input grid interfaces in the .gdlay, while the variation of the 3rd-dim is affected by values of this line. CGFD3D will first generate the grid of the grid layers with equal grid spacing, then generate rest grid between these equal-spacing layers. Thus any layer of "0" equal value, should have both "1" layers above and below it.

- The fourth line: two int values *NX* and *NY*, the number of horizontal grid points of each grid interfaces of this .gdlay file. The numbers are same for all the grid interfaces.
- Rest are the X, Y and Z coordinates of each grid point per line. The order of the grid points is: 1st-dim, then the 2nd-dim of the first grid interface (the bottom one as the z-axis upward positive); then those of the 2nd grid interfaces, etc, which can can be expressed by the following pseduo code:

```
for (int ilay = 0; ilay < NI; ilay++) {
  for (int j = 0; j < NY; j++) {
    for (int i = 0; i < NX; i++) {
      fscanf(fp,"%g %g %g", &xcoord, &ycoord, &zoord);
    }
  }
}
```

We provide some examples of matlab scripts to generate .gdlay under the `example/prep_grid` directory.

Chapter 2

Input Layer or Grid Structure Models

One major reason to use a computational expensive grid-based seismic wave numerical code is to simulate seismic wave propagation in complex velocity models, which can significantly affect the seismic waveforms. The velocity structure of the Earth, especially near the surface, is vey complex. How to input the complex velocity model and accurately representing the velocity model using the discrete simulation grid are important components of a seismic wave simulation program.

CGFD3D designs two types of file formats to ease the input of the velocity models.

- One is the layer-based model format (.md3lay), which can be used for topographic layers with known medium paramters along the ingterface and vertical gradients for values inside the layer.
- The second one is the grid-based model (.md3grd), which is indeed a hybrid model format that also supports topographic layers structures, but internal value is interpolated from vertically discrete sampling values inside one layer. Regular grid models produced by tomography, can be taken as a special case of the .md3grd with a single layer and equal vertical invertal at all horizonal sampling points.

2.1 Different Medium Types Supported by CGFD3D

CGFD3D supports simulating wave propagation in following media:

- `acoustic_iso`: abbrevation of acoustic wave equation of isotropic media.
- `elastic_iso`: abbrevation of elastic wave equation of isotropic media.
- `elastic_vti`: abbrevation of elastic wave equation of vertical transversely isotropic media.
- `elastic_aniso`: abbrevation of elastic wave equation of anisotropic media.

More complex media will be implemented in the future.

2.2 Medium Parameterization Methods

One parameter the user should determine but the meaning is not so straightfowrad is "medium parameterization method". We explain this concept first to ease description of the medium input in following chapters.

If there are topographic interfaces with strong velocity contrast, medium paramters at a grid point directly taking values according to its coordinate in the input velocity model will result in stair-case representation of the interfaces, introduce interface-related errors and cause stong artificial diffraction. One advance feathure of CGFD3D is to evaluate equivalent medium parameters at grid points to accurately represent the location of the interface inside a grid cell and realize subgrid resolution of the layer velocity models. CGFD3D supports various medium parameterization approaches:

- loc: abbrevation of direct location, which is the simplest and most widely used one. In this approach, the medium parameter of a grid point, is the value just at that ccoordinate position in the input velocity model. In terms of processing time, LOC method is fastest comparing to rest methods.
- ari: abbrevation of volume arithmetic averaging (add ref) as

$$\langle var \rangle = \frac{1}{\Delta V} \int_{k-1/2}^{k+1/2} \int_{j-1/2}^{j+1/2} \int_{i-1/2}^{i+1/2} var \, dxdydz, \tag{2.1}$$

  which evaluats an averaging value for all the medium parameters in the cell volume representing by the grid point.
- har: abbrevation of volume harmonic averaging for medium modulus as

$$\langle var \rangle^H = \frac{\Delta V}{\int_{k-1/2}^{k+1/2} \int_{j-1/2}^{j+1/2} \int_{i-1/2}^{i+1/2} \frac{1}{var} dxdydz}, \tag{2.2}$$

  which evaluats an averaging value of medium modulus in the cell volume representing by the grid point, while and volume arithmetic average for density as proposed by (ref). The approach does not alter medium type, e.g., isotropic model is still an isotropic one, after parametrization.
- tti: abbrevation of tiled transversely isotropic equivalent medium approach (add ref), which evaluats an effictive TTI representation of the thin layers inside the cell volume representing by the grid point. The approach does alter medium type, e.g., isotropic model becomes a TTI anisotropic one, after parametrization.

The parametrization name may mean a slightly different combination of hormonic and arithmetic averaging of difference medium parameters for different medium types (Table **??**).

medium type	loc^L	ari^A	har^H	tti^T
`one_component`	all^L	all^A	all^H	NA
`acoustic_isotropic`	all^L	all^A	κ^H, ρ^A	todo
`elastic_isotropic`	all^L	all^A	λ^H,μ^H,ρ^A	todo
`elastic_vti_*`	all^L	all^A	C_{ij}^H, ρ^A	todo
`elastic_aniso_cij`	all^L	all^A	C_{ij}^H, ρ^A	todo
`elastic_tti_*`	all^L	all^A	C_{ij}^H, ρ^A	todo

Table 2.1: Medium parameterization on difference parameters for different medium types.

2.3 Parameters in Main Par File

Listing 2.1: Example of medium settings in .json

```
"medium" : {
    "type" : "elastic_iso",
    "#code" : 1,
    "infile_layer" : "/home/usr1/testCGFD3D/prep_medium/basin.md3lay",
    "#infile_grid" : "/home/usr1/testCGFD3D/prep_medium/basin.md3grd",
    "#equivalent_medium_method" : "har",
    "#import" : "/home/usr1/testCGFD3D/output/media"
},
"is_export_media" : 1,
"media_export_dir"  : "/home/usr1/testCGFD3D/output/media",

"#visco_config" : {
    "type" : "graves_Qs",
    "Qs_freq" : 1.0
}
```

There are several medium related keys in the main par .josn file (List **??**),

- `medium`: major block for medium input, tells the program:
  - `type` determins the solver equation, that is, what medium you want to simulate wave propagation in. Four options are currently supported in this code:
    - `acoustic_iso`: acoustic wave equation of isotropic media,
    - `elastic_iso`: elastic wave equation of isotropic media,
    - `elastic_vti`: elastic wave equation of vertical transversely isotropic media,
    - `elastic_aniso`: elastic wave equation of anisotropic media.
  - how the structure model is input or specified. There are four possible ways: `import`, `code`, `infile_layer` or `infile_grid`.
    - `code`: Choose this flag if you want to give the media by code. You should edit **forward/md_t.c** file and recompile the code.
    - `infile_layer`: If you want to give media by the intereface line, this option should be helpful. The media parameters are given by the **md3lay** file, and the file format is shown in Section **??**
    - `infile_grid`: If you want to give media by a given grid media, this option should be helpful. The media parameters are given by the **md3grd** file, the file format is shown in Section **??**.
    - `import`: If you already generated media from the above three options, and do not want to re-generate media, select this option and give the folder of the exported media.
  - `equivalent_medium_method`: If you specify structure mode by `infile_layer` or `infile_grid`, equivalent medium parameterization methods can be applied. For different media, we provide different equivalent medium parameterization methods, please see Section **??** for detail.
- `is_export_media`: if the discreted medium parameters at FD grid points are exported for display purpose or reuse for later calculation without re-discrete media calculation.
  - 0: do not export discrete media,
  - 1: export discrete media.
- `media_export_dir`: set the output dir if `is_export_media : 1`.
- `visco_config`: block for attuantion settings. If this block is enabled (appeared in the .json), the attuation will be implemented with following two parameters (add Grave's ref):
  - `type`: currently only `graves_Qs` can be set, which means to implement attuation following Grave (1996) approach.
  - `Qs_freq`: the frequency of the wavefield for which the Q effect is most accurately approximated.

2.4 Format of Layer-based Velocity Model (.md3lay)

The Earth structure of many regions has following features:

- the structure consists of many layers with topographic interfaces,
- the velocity or density in some layers may increase or decrease along depth, which may be expressed by the value at the interface and two polynomial parameters (coefficient and power) with respect to depth.

For such structures, we can use .md3lay file to input the structure into CGFD3D.

Listing 2.2: Example of .md3lay file

```
elastic_isotropic
4
126 106 -300.000000 -300.000000 100.000000 100.000000
-0 1000 0.2 1 1500 0.2 1 1000 0.2 1
-0 1000 0.2 1 1500 0.2 1 1000 0.2 1
-0 1000 0.2 1 1500 0.2 1 1000 0.2 1
-0 1000 0.2 1 1500 0.2 1 1000 0.2 1
```

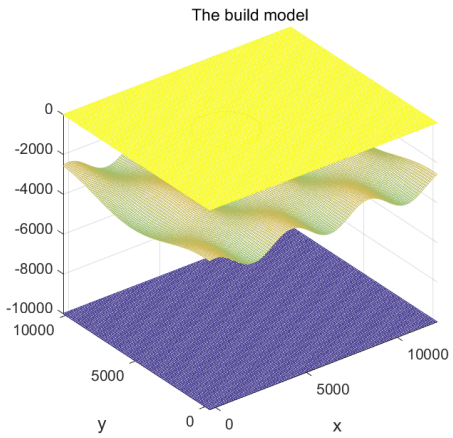


Figure 2.1: Layer-based velocity model with topographic internal interfaces.

```
-0 1000 0.2 1 1500 0.2 1 1000 0.2 1
-0 1000 0.2 1 1500 0.2 1 1000 0.2 1
-0 1000 0.2 1 1500 0.2 1 1000 0.2 1
.
.
.
```

An example of .md3lay is shown in Listing **??** and Figure **??**. The detailed format of the .md3lay is:

- The first line: sets the medium type, which can be
  - `one_component`: there is only a single medium parameter in this file.
  - `acoustic_isotropic`: two parameters ρ and Vp.
  - `elastic_isotropic`: three parameters ρ, Vp, Vs.
  - `elastic_vti_prem`: density and five VTI parameters same to PREM model, ρ, vph, vpv, vsh, vsv, η. Please see (**?**) for detail.
  - `elastic_vti_thomsen`: density and five Thomsen parameters, ρ, α_0, β_0, ε, δ, γ. Please see (**?**) for detail.
  - `elastic_vti_cij`: density and five VTI parameters given by C_{ij}, ρ, c_{11}, c_{33}, c_{55}, c_{66}, c_{13}.
  - `elastic_tti_thomsen`: density ,five Thomsen parameters, two angle of symmetric axis. ρ, α_0, β_0, ε, δ, γ, Φ, θ. Please see (**?**) for detail.
  - `elastic_tti_bond`: ρ, c_{11}, c_{33}, c_{55}, c_{66}, c_{13}, Φ, θ. Φ is azimuth angle, θ is the dip angle.
  - `elastic_aniso_cij`: ρ, c_{11}, c_{12}, c_{13}, c_{14}, c_{15}, c_{16}, c_{22}, c_{23}, c_{24}, c_{25}, c_{26}, c_{33}, c_{34}, c_{35}, c_{36}, c_{44}, c_{45}, c_{46}, c_{55}, c_{56}, c_{66}.
- The second line: the number of interface (`NI`) (4 in this example).
- The third line: six values (two integers and four float) specify the horizontal sampling points as:
  `NX NY X0 Y0 DX DY`,
  where
  - `NX` and `NY`: the number of sampling points along x and y direction (126 and 106 for this example);
  - `X0` and `Y0`: the x and y coordinates of the first point (-300 and -300 for this example);
  - `DX` and `DY`: spacing between sampling points along x and y (100 and 100 for this example).
- Rest are the elevation and media parameters of each sampling point per line. Take the `elastic_isotropic` media as an example, the media parameters are read as:

```
for (ni=0; ni<NI; ni++)
  for (iy=0; iy<NY; iy++)
```

```
for (ix=0; ix<NX; ix++)
  fscanf(in_3lay_file, "%f %f %f %f %f %f %f",
         &elevation,
         &rho, &rho_grad, &rho_pow,
         &vp, &vp_grad, &vp_pow,
         &vs, &vs_grad, &vs_pow);
```

where `*_grad` and `*_pow` are the coefficient and power of the parameters in z-direction as in following equation:

$$\text{value}^{\text{grid point}} = \text{value}^{\text{interface}} + (\text{z}^{\text{interface}} - \text{z}^{\text{grid point}})^{\text{var\_pow}} * \text{var\_grad}.$$

Please notice that the z-axis is positive upward in CGFD3D.

Please note that

- if the computational grid point is located above the top interface of the .md3lay model, the medium parameters are taken as the values at the top interface.
- Different media parameters can have different `*_grad` and `*_pow`.

You can find matlab example script to generate .md3lay in **example/prep_medium** directory.

2.5 Format of Grid-based Velocity Model (.md3grd)

If the velocity variation along depth inside a layer can not be expressed by a polynomial expression with respect to depth, then we have to give sampling values at discrete points along the depth. This is why we privoide the .md3grd, the grid-based velocity file.

We design the .md3grd also supporting layer interfaces, thus the interface effects can be simulated as accurate as possible. To do so, we need to also set number of layers in the .md3grd file, then discrete the velocity variation along depth for each layer.

Listing 2.3: Example of .mdgrd file

```
elastic_isotropic
3
 11 21 101
126 106 -300.000000 -300.000000 100.000000 100.000000
0 1000 1510 1020
0 1000 1510 1020
0 1000 1510 1020
0 1000 1510 1020
0 1000 1510 1020
0 1000 1510 1020
0 1000 1510 1020
.
.
.
```

An example of .md3grd is shown in Listing **??**. The detailed format of the .md3grd is:

- The first line: sets the medium type, similar to that of .md3lay. Please see descriptions in **??**.
- The second line is the number of layer or interfaces (`NL`) (3 in this example). If `NL` = 1, the model becomes the standard grid model that only has descrete sampling value along depth but without interfaces. If `NL` > 1, the equivalent medium parameterization can be applied across the interfaces.
- The next `NL` numbers (either at the same line or `NL` lines) set the number of grids in the z-direction of `NL` layer.
- The next line: six values (two integers and four float) specify the horizontal sampling points as:
  `NX NY X0 Y0 DX DY`,
  where

  * NX and NY: the number of sampling points along x and y direction (126 and 106 for this example);
  * X0 and Y0: the x and y coordinates of the first point (-300 and -300 for this example);
  * DX and DY: spacing between sampling points along x and y (100 and 100 for this example).

– Rest are the elevation and media parameters of every grid point per line. Take the elastic_isotropic media as an example, the media parameters are read as:

```
for (nl=0; nl< NL; nl++)
  for (ip=0; ip<np[nl]; np++)
    for(iy=0; iy<NY; iy++)
      for (ix=0; ix<NX;ix++)
        fscanf(in_3grd_file, "%f %f %f %f",
            &elevation, &rho, &vp, &vs);
```

The medium parameter at a computational grid point is calculated by linear interpolation of two values at the discrete velocity model points nearest to the computational grid point.

Please note that

– if the computational grid point is located above the top interface of the .md3grd model, the medium parameters are taken as the values at the top interface.
– if NL > 1, the z-axis of last point of the top layer should be equal to that of the first point of the lower layer, and the equivalent medium parameterization method can be applied at points close to the interface.

You can find matlab example script to generate .md3grd in **example/prep_medium** directory.

Chapter 3

Input Point or Finite-fault Sources

Seismic wave is actived by the seimisc source. In seismic wave equations, the seismic source is represented by the force term ($\mathbf{F}$) and/or stress gluss term ($\dot{\mathbf{M}}$) at the right-hand side (RHS) of the equations:

$$\rho \frac{\partial \mathbf{v}}{\partial t} = \nabla \cdot \sigma + \mathbf{F}, \tag{3.1}$$

$$\frac{\partial \sigma}{\partial t} = \mathbf{C} : \frac{1}{2}(\nabla \mathbf{v} + \mathbf{v}\nabla) - \dot{\mathbf{M}}. \tag{3.2}$$

The force term is used for sources acting as an external force to the simulatin region, while the stress gluss term could be used to implement a general moment tensor souce. It is relative easy to implement the source in the code by adding the respective terms onto the RHS values. We just need a way to input different combinations of how many sources, the locations, source time funcion (STF) and source mechanism for a single simulation into the code.

Different applications may require a source in different formats, e.g,

1. single force at a single point, which is a common source for Green function and exploration seismology;
2. general moment source at a single point, which is used for small earthquake and explosive source;
3. general moment sources along a fault, which is also called finite-fault source model and used for large earthquake;
4. time-reversed waveforms along surface, which is used for time reversal imaging;
5. plane wave incident for teleseismic wave problems.

The first two sources act at a single point, while the last three types are implemented by adding many single point sources simultaneously along a fault, a surface or a pre-defined line (plane). The first two types can be taken as a special case of the last three types when number of the source becomes one.

The CGFD3D package tries to implement different sources to support different applications. Thus we allow to input a single point source or many point sources. For easy usage, we also support:

– the location of the single point could be grid index or coordinate;
– the vertical coordinate could be given by the absolute axis or the depth relative the free surface (to be done!);
– STF of each single point source could be an analytical wavelet function or discrete values obtained by real data inversion or source dynamic simulation;
– the source could be a force source and/or a moment one;
– the moment source could be given by the six tensor components or the mechanism angles plus μDA.

To allow different input combinations of above different parameters, we use several flags in the input file to determine the specific input meaning of the related parameters. In the following, we will describe the format of the source input file of the CGFD3D package and give several examples to show how to input different sources.

3.1 Set Parameters in Main Par File

All the detailed specification of the source is written in a separate source file (.src). Throug the main input .json file, we need to tell the simulation code where is the source file, which is specified by `in_source_file`. There are total three paramters in the .json file related to the source input:

- `in_source_file`: set the input source file;
- `is_export_source`: determine if export the internal discreted source for QC (not implemented yet);
- `source_export_dir`: the path for source exporting.

Currently, only `in_source_file` is implemented and the other two have no effect yet.

Example of source settings in .json

```
  "in_source_file" : "/home/user/prj/input/test.src",
  "is_export_source" : 1,
  "source_export_dir" : "/home/user/prj/output",
```

3.2 Set Source Information by the Source Input File (.src)

The .src file is designed to input all the required source information using different representations. Considering parallel computing using MPI on a multi-nodes cluster, we want to allocate large array holdding discrete STF values for only the source located at the node. Thus we divide the information in the .src into three regions (line number refer that in the example):

1. global information related to all the sources (line 1-20);
2. location information of each source (line 21-25);
3. STF and component information of each source (after line 26).

In other words, we specify the global information/flags first, then list the locations of all the sources, and finally give the STF and component values of each source. The last part may be very large for discrete STF input.

A sample .src file using analytical wavelet function as STF is shown below. We will use this sample .src file to explain the file format of .src file.

Listing 3.1: Source input file using analytical wavelet

```
1  # name of this input source
2  event_1
3  # number of source
4  1
5  # flag for stf
6  #  1st value : 0 analytic stf or 1 discrete values
7  #    for analytical, 2nd value is time length
8  #    for discrete,  2nd value is dt and 3rd is nt
9  #    e.g.,
10 # 0 4.0
11 # 1 0.05 20
12 0 1.0
13 # flag for source component and mechanism format
14 #  1st value: source components, 1(force), 2(momoment), 3(force+moment)
15 #  2nd value: mechanism format for moment source: 0 moment, 1 angle + mu + D + A
16 2 0
17 # flag for location
18 #   1st value: meaning of the location: 0 computational coordinate, 1 physical
     coordinate
```

```
19 #   2nd value: 0 absolute axis or 1 depth relative to the free surface of the third
       coordinate
20 1 0
21 # location of each source
22 #   sx sy sz
23 #80 49 50
24 #80 49 50
25 8020 4930 -950
26 # stf and cmp
27 0.0 ricker 2.0 0.5   # t0  stf_name  ricker_fc ricker_t0
28 1e16  1e16  1e16 0 0 0
```

From above sample file, We see that we can insert comment line in the .src file using "#" as the line start, similar to the Bash script syntax. What we should set in the .src file are:

- name of the whole source (represented by EVTNM, following sac notation) (line 2), type should be string without whitespace. This name is used for output file naming. E.g., the output sac file will start with "event_1" for this sample file.
- number of the sources (NS) (line 4), type is interger. The second part should contain NS locations and the third part should contains NS STF and cmp settings.
- settings about STF (line 12). Two values or three values depends on the first value, which tells the code how STF is given (STF_flag):
  - 0: the STF whill be set using wavelet name and related coefficiences in the second part for each source. The second value here sets the same total time length of all the STF to reduce using computer memory to hold discrete STF in simulation.
  - 1: the STF will be given as discrete values. Then the second value is the time step (stf_dt) and the third one is the total number of time step (stf_nt).

  We should note that in the third part, we will give each source a separte start time to implement finite-fault source combining above time window setting.
- two values to set force and/or moment type of each source, and how the moment source is given (line 16). The first value (CMP_flag):
  - 1: force;
  - 2: moment;
  - 3: force and momemnt.

  The second value (Mechanism_flag) determines how the mechanism is given if moment source is used:
  - 0: values of the six components of the moment tensor;
  - 1: strke, dip and rake angels, plus μ, D, and A.
- two flags about the meaning of the location values (line 20). The first flag is the meaning of the location (LOC_flag),
  - 0: the location is given by the computational coordinate. The value is same to the grid index in this implementation. The decimal part is the relative shift of the source to the grid point;
  - 1: the location is given as the physical coordinate values, which is the Cartesian coordinate in this implementation.

  The second flag (Vertical_flag) is used when the first flag set 1 to determine the third coordinate is
  - 0: absolute z-axis value;
  - 1: depth relative to the free surface (need to be implemented!).
- Then the second part, the locations of each sources (line 22-25). There should be NS lines for NS sources. Each line contains three values representing the location of one source. If LOC_flag = 0 (computational coordinate), the location should be given by grid index with decimal as

Listing 3.2: Source location by computational coordinate

```
# location of each source (e.g., for NS = 3)
20.0 20.0 50.0
30.0 29.0 50.0
40.5 50.2 55.1
```

If LOC_flag = 1 (physical coordinate), the location should be given by its coordinate values as

Listing 3.3: Source location by physical coordinate

```
# location of each source (e.g., for NS = 3)
2000.0 2000.0 5000.0
3000.0 2900.0 5000.0
4050.0 5020.0 5510.0
```

– The third part specifies STF and component values of each source (after line 26). The formats are different for different STF_flag.
If STF_flag = 0 (wavelet name), each source has two lines,

1. two or more values to set the STF:
   * first value: the activated (or start) time, which is important for finite-fault model to represent fault rupture.
   * second value: a string, the wavelet function name. The valid wavelet names for current CGFD3D are shown in table **??** (more wavelent functions will be implemented in the future). You can easily add a wavelet by modifying function `src_cal_wavelet` in src_t.c.
   * 3rd-12th values: the coefficients required by the wavelet function. Different wavelet may require different number of coefficients. Here we allow up to maximum 10 coefficients to specify the values. It will no problem only setting one or two values if the wavelet only needs one or two coefficients. Please see table **??** for the number and meanings of coefficients of current supported wavelets.
2. 3, 6 or 9 values (depending on CMP_flag) to give the magnitudes of the force and/or moment components.
   * CMP_flag=1: 3 values to set values of F_x F_y and F_z;
   * CMP_flag=2: 6 values to set values of the moment tensor, ordered as
     · $m_{xx}, m_{yy}, m_{zz}, m_{yz}, m_{xz}, m_{xy}$ following the Viogt nonation (Fig **??**) if Mechanism_flag=0
     · strike, dip, rake, μ, D, A if Mechanism_flag=1.
   * CMP_flag=3: 9 values to set values of both the force and the moment tensor, ordered as
     · $F_x, F_y, F_z, m_{xx}, m_{yy}, m_{zz}, m_{yz}, m_{xz}, m_{xy}$ if Mechanism_flag=0
     · F_x, F_y, F_z, strike, dip, rake, μ, D, A if Mechanism_flag=1.

If STF_flag = 1 (discrete values), each source has (1+stf_nt) lines (see sample Listing **??**),

1. first line: one value, the activated (or start) time.
2. stf_nt lines of 3, 6 or 9 values (depending on CMP_flag) to give the magnitudes of the force and/or moment components at each time step:
   * CMP_flag=1: 3 values of F_x F_y and F_z;
   * CMP_flag=2: 6 values of the moment tensor, ordered as
     · $m_{xx}, m_{yy}, m_{zz}, m_{yz}, m_{xz}, m_{xy}$ following the Viogt nonation (Fig **??**) if Mechanism_flag=0
     · strike, dip, rake, μ, D, A if Mechanism_flag=1.
   * CMP_flag=3: 9 values of both the force and the moment tensor, ordered as
     · $F_x, F_y, F_z, m_{xx}, m_{yy}, m_{zz}, m_{yz}, m_{xz}, m_{xy}$ if Mechanism_flag=0
     · F_x, F_y, F_z, strike, dip, rake, μ, D, A if Mechanism_flag=1.

Listing 3.4: Source input file using discrete STF

```
# name_of_this_input_src
evt_1_stf_value
```

Table 3.1: Implemented Wavelet functions and their coefficients.

wavelet name	coef[0]	coef[1]
ricker	cetral frequency	central time shift
ricker_deriv	cetral frequency	central time shift
gaussian	RMS value	time shift
gaussian_deriv	RMS value	time shift

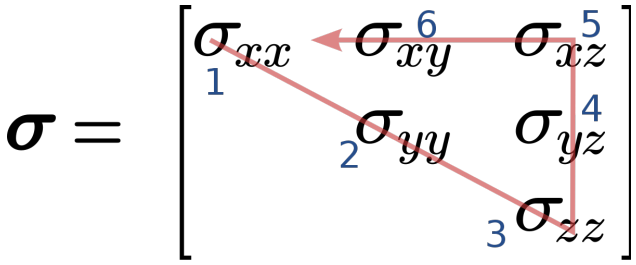


Figure 3.1: Viogt notation (From: Wiki)

```
# number of source
1
# meaning of the location: 0 computational coordinate, 1 physical coordinate
#     axis or depth of the third coordinate: 0 axis, 1 depth
0 1
# stf_input_type and time length info
#  0 4.0 : analytic and time window length of each stf
#  1 0.05 20  : 1 value and time_step num_of_step
1 0.025 40
# 1(force) 2(momoment) 3(force+moment)
#    mechanism type for moment source: 0 moment, 1 angle + mu + D + A
2 0
# meta data of each source
#   sx sy sz
80 49 50
# value data for each source
#  t0
0.0
# Mxx etc
4.29488e+12    4.29488e+12   4.29488e+12     0.0  0.0  0.0
1.0088e+13     1.0088e+13    1.0088e+13      0.0  0.0  0.1
2.24061e+13    2.24061e+13   2.24061e+13     0.0  0.0  0.0
4.70162e+13    4.70162e+13   4.70162e+13     0.0  0.0  0.0
9.31041e+13    9.31041e+13   9.31041e+13     0.0  0.0  0.0
1.73751e+14    1.73751e+14   1.73751e+14     0.0  0.0  0.0
3.05036e+14    3.05036e+14   3.05036e+14     0.0  0.0  0.0
5.02611e+14    5.02611e+14   5.02611e+14     0.0  0.0  0.0
7.7483e+14     7.7483e+14    7.7483e+14      0.0  0.0  0.0
1.11265e+15    1.11265e+15   1.11265e+15     0.0  0.0  0.0
1.47863e+15    1.47863e+15   1.47863e+15     0.0  0.0  0.0
1.79991e+15    1.79991e+15   1.79991e+15     0.0  0.0  0.0
1.97157e+15    1.97157e+15   1.97157e+15     0.0  0.0  0.0
1.87557e+15    1.87557e+15   1.87557e+15     0.0  0.0  0.0
1.41548e+15    1.41548e+15   1.41548e+15     0.0  0.0  0.0
5.58831e+14    5.58831e+14   5.58831e+14     0.0  0.0  0.0
-6.2831e+14    -6.2831e+14   -6.2831e+14     0.0  0.0  0.0
-1.97263e+15   -1.97263e+15  -1.97263e+15    0.0  0.0  0.0
```

```
-3.22222e+15   -3.22222e+15  -3.22222e+15    0.0  0.0  0.0
-4.1098e+15    -4.1098e+15   -4.1098e+15     0.0  0.0  0.0
-4.43113e+15   -4.43113e+15  -4.43113e+15    0.0  0.0  0.0
-4.1098e+15    -4.1098e+15   -4.1098e+15     0.0  0.0  0.0
-3.22222e+15   -3.22222e+15  -3.22222e+15    0.0  0.0  0.0
-1.97263e+15   -1.97263e+15  -1.97263e+15    0.0  0.0  0.0
-6.28308e+14   -6.28308e+14  -6.28308e+14    0.0  0.0  0.0
5.58831e+14    5.58831e+14   5.58831e+14     0.0  0.0  0.0
1.41548e+15    1.41548e+15   1.41548e+15     0.0  0.0  0.0
1.87557e+15    1.87557e+15   1.87557e+15     0.0  0.0  0.0
1.97157e+15    1.97157e+15   1.97157e+15     0.0  0.0  0.0
1.79991e+15    1.79991e+15   1.79991e+15     0.0  0.0  0.0
1.47863e+15    1.47863e+15   1.47863e+15     0.0  0.0  0.0
1.11265e+15    1.11265e+15   1.11265e+15     0.0  0.0  0.0
7.7483e+14     7.7483e+14    7.7483e+14      0.0  0.0  0.0
5.02611e+14    5.02611e+14   5.02611e+14     0.0  0.0  0.0
3.05036e+14    3.05036e+14   3.05036e+14     0.0  0.0  0.0
1.73751e+14    1.73751e+14   1.73751e+14     0.0  0.0  0.0
9.3104e+13     9.3104e+13    9.3104e+13      0.0  0.0  0.0
4.70162e+13    4.70162e+13   4.70162e+13     0.0  0.0  0.0
2.24061e+13    2.24061e+13   2.24061e+13     0.0  0.0  0.0
1.0088e+13     1.0088e+13    1.0088e+13      0.0  0.0  0.0
```

Copyright

Main historical authors:

Evolution of the code: