CGFD3D Seismic Wave Simulation Program
User Manual

September 23, 2022

Contents

Chapter 1

Brief Description of the Program

This manual is about how to use the program package for seismic wavefield forward modeling.This program is suitable for 3D acoustic wave equations and elastic wave equations.It can be applied to isotropic and anisotropic media. We can use the Cartesian grid and the curvilinear grid for calculation.

In this Program, we improved FDM to accurately simulate seismic wave propagation in the presence of surface topography in three ways:

1. Using boundary-conforming grid to conform the grid with the surface. The grid can be used in the topography with steep slopes, and can avoid the artifact diffraction caused by staircase grid;

2. Using non-staggered higher-order optimized DRP /opt MacCormack scheme to solve the first order velocity-stress equations, which improves the compatibility of the velocity- stress approach for the anisotropic media and irregular surface;

3. Developing a new numerical method, named as Traction Image method, to accurately and efficiently implement the traction-free boundary conditions in the presence of surface topography.

We implemented the boundary-conforming grid, non-staggered DRP /opt MacCor- mack scheme and Traction Image method in SH, P-SV and 3D seismic wave modelling in the presence of surface topography.

This manual is mainly introduced from compiling, grid discretization, medium model, source and receivers setup, absorption boundary, result output, example display, for the convenience of users.

Chapter 2

The Structure of Code and Compilation

2.1 The Structure of Code

CGFD3D-elastic package is download as a directory with the follow subdirectories.

```
CGFD3D-elastic
├── doc/ ..... This USER_MANUAL can be found here
├── forward/ ..... Source code for programs
├── Makefile
├── mfiles/ ..... Drawing code in MATLAB
├── run_make.mars ..... The bash-script to run to make on mars server
├── test/ ..... Code for test
├── example/ ..... Code for preparing and running script
├── lib/ ..... Library used by the programs
├── media/ ..... Source code related to media
├── pyfiles/ ..... Drawing code in Python
└── run_make.server1 ..... The bash-script to run to make on server1 server
```

2.2 Compilation and Linking

CGFD3D-elastic package is written in C and C++, so makesure C and C++ compilers are installed on your system. And you need to install MPI and NETCDF. MATLAB is also needed for generating data and displaying the results. Make clean is suggested before making by typing

```
> make distclean
```

Then set the ROOT variable. Here we provide the bash script `run_make.server1` (Listing 2.1) for making code. You need to provide `MPI_ROOT` and `NETCDFROOT` in `run_make.server1` to adapt to the system you are using, and add mpi `bin` file to PATH. The target executable is `main_curv_col_el_3d`.

Listing 2.1: run_make.server1

```
#-- add mpi to PATH if module is not used, on server1
MPI_ROOT=/share/apps/gnu-4.8.5/mpich-3.3/
export PATH=$MPI_ROOT/bin:$PATH
export NETCDFROOT=/share/apps/gnu-4.8.5/disable-netcdf-4.4.1

echo "mpicc and mpicxx will invoke:"
mpicc -show
```

```
mpicxx -show

echo
echo "start to make ..."
make -f Makefile
```

After setting ROOT variable, run the script with the command

```
> ./run_make.server1
```

If module is used on your system, we provide another bash script `run_make.mars`. Before running the script, you need to load compiler, mpi, netcdf and matlab first. After that, run the script

```
> ./run_make.mars
```

The object files and executables will be generated.

Some useful make commands:
`make cleanall` : remove all object files and executables.
`make cleanexe` : remove all executables.
`make cleanobj` : remove all objects files.

Chapter 3

Getting started

This chapter only gives a brief introduction to the running of the program and the configuration of the main parameter file and running script. For more detailed information about these files, please refer to concerning chapter.

3.1 Running CGFD3D

The program can be running by simply using the `sh` command, eg. for acoustic wave modeling

```
sh cgfd3d-wave-ac.sh
```

The parameters and output directions can be modified in the `.sh` file.

3.2 Parameters in Main Par File

- Number of grid points.

```
"number_of_total_grid_points_x" : 300,
"number_of_total_grid_points_y" : 300,
"number_of_total_grid_points_z" : 60,
```

- Number of mpi processes.

```
"number_of_mpiprocs_x" : 3,
"number_of_mpiprocs_y" : 3,
```

- Size and number of time steps or simply the length of time window, which will automatically give a stable time step.

```
 "size_of_time_step" : 0.008,
 "#size_of_time_step" : 0.020,
 "number_of_time_steps" : 500,
 "#time_window_length" : 4,
 "check_stability" : 1,
```

- Absorbing boundary parameters.

```
"boundary_x_left" : {
      "cfspml" : {
          "number_of_layers" : 10,
```

```
            "alpha_max" : 3.14,
            "beta_max" : 2.0,
            "ref_vel"  : 7000.0
            }
       },
 ....
 "boundary_z_top" : {
     "free" : "timg"
       },
```

- The grid configuration, which can be imported from outside or use the default cartesian coordinate system.

```
"grid_generation_method" : {
      "#import" : "$GRID_DIR",
      "cartesian" : {
      "origin"  : [0.0, 0.0, -5900.0 ],
      "inteval" : [ 100.0, 100.0, 100.0 ]
      },
      "#layer_interp" : {
       "in_grid_layer_file" : "$INPUTDIR/prep_grid/tangshan_area_topo.gdlay"
           ,
       "refine_factor" : [ 1, 1, 1 ],
       "horizontal_start_index" : [ 3, 3 ],
       "vertical_last_to_top" : 0
      }
  },
  "is_export_grid" : 1,
  "grid_export_dir"   : "$GRID_DIR",
```

- The method used to calculate the metric parameters.

```
 "metric_calculation_method" : {
       "#import" : "$GRID_DIR",
       "calculate" : 1
   },
 "is_export_metric" : 1,
```

- The medium configuration.

```
 "medium" : {
       "type" : "elastic_iso",
       "input_way" : "infile_layer",
       "#input_way" : "binfile",
       "#binfile" : {
         "size"    : [1101, 1447, 1252],
         "spacing" : [-10, 10, 10],
         "origin"  : [0.0,0.0,0.0],
         "dim1" : "z",
         "dim2" : "x",
         "dim3" : "y",
         "Vp" : "$INPUTDIR/prep_medium/seam_Vp.bin",
         "Vs" : "$INPUTDIR/prep_medium/seam_Vs.bin",
         "rho" : "$INPUTDIR/prep_medium/seam_rho.bin"
       },
       "code" : "func_name_here",
```

```
      "import" : "$MEDIA_DIR",
      "infile_layer" : "$INPUTDIR/prep_medium/basin_el_iso.md3lay",
      "infile_grid" : "$INPUTDIR/prep_medium/topolay_el_iso.md3grd",
      "equivalent_medium_method" : "loc",
      "#equivalent_medium_method" : "har"
  },
  "is_export_media" : 1,
  "media_export_dir"  : "$MEDIA_DIR",
```

- The viscosity configuration.

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

- The input source file.

```
"in_source_file" : "$INPUTDIR/event_3moment_discrete.src",
"is_export_source" : 1,
"source_export_dir"  : "$SOURCE_DIR",
```

- The station list file.

```
"in_station_file" : "$INPUTDIR/station.list",
```

- The output data that needs to be stored, including the receiver line, the slice and the snapshot.

```
"receiver_line" : [
    {
      "name" : "line_x_1",
      "grid_index_start"    : [  0, 49, 59 ],
      "grid_index_incre"    : [  1,  0,  0 ],
      "grid_index_count"    : 20
    },
    {
      "name" : "line_y_1",
      "grid_index_start"    : [ 19, 49, 59 ],
      "grid_index_incre"    : [  0,  1,  0 ],
      "grid_index_count"    : 20
    }
  ],

  "slice" : {
      "x_index" : [ 190 ],
      "y_index" : [ 90 ],
      "z_index" : [ 59 ]
  },

  "snapshot" : [
    {
      "name" : "volume_vel",
      "grid_index_start" : [ 0, 0, 59 ],
      "grid_index_count" : [ 300,300, 1 ],
      "grid_index_incre" : [  1, 1, 1 ],
      "time_index_start" : 0,
```

```
      "time_index_incre" : 1,
      "save_velocity" : 1,
      "save_stress"   : 1,
      "save_strain"   : 1
    }
  ],
```

3.3 Parameters in Running Script

The running script mainly contains the initialization of the environment and the make command. The initialization makes sure that the desired versions of programs are added to the environment viriables. Environment Mudules package is a tool that simplify shell initialization and allows users to manage different versions of applictions with faster compiling speed. The export command can be used to temporarily initiate the environment if the Environment Modules packages are not installed. Both commands are optional and you need to modify the path corresponding to your current path of installation.

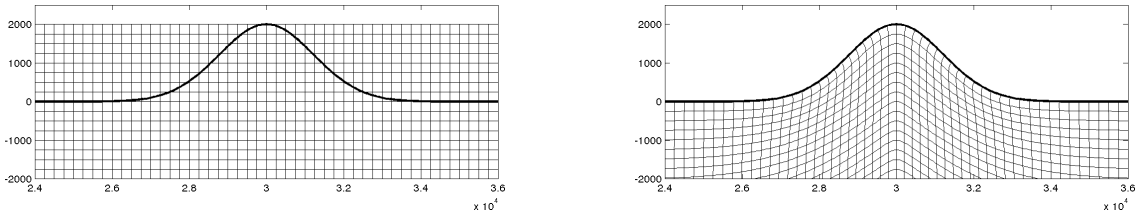
Listing 3.1: The running script in .json

```
#!/bin/bash

#-- use module to set mpicc etc, on Mars
#module load intel/2019.5
#module load mpi/mpich/3.4.1_intel_2019.5
#module load mpi/mpich/3.3.1_intel_2019.5
#module load netcdf-c/4.4.1

#-- add mpi to PATH if module is not used, on server1
MPI_ROOT=/share/apps/gnu-4.8.5/mpich-3.3/
export PATH=$MPI_ROOT/bin:$PATH
exprot NETCDFROOT=/share/apps/gnu-4.8.5/disable-netcdf-4.4.1

echo "mpicc and mpicxx will invoke:"
mpicc -show
mpicxx -show

echo
echo "start to make ..."
make -f Makefile 2>&1 | tee log.make
```

Chapter 4

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 4.1a), 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 4.1b). 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 4.1: Computational grids

4.1 Set Grid Parameters in Main Par

Listing 4.1: 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 4.1, 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`.

4.2 Generation of Cartesian Grid

To use the Cartesian grid, we can simply set parameters in the .json file as List 4.2. 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 4.2: 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",
```

4.3 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 4.3):

- `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 4.4 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.

4.4 Format of Grid Layer Interp File (.gdlay)

Listing 4.3: Example of .gdlay file

```
1      4
2      43     10     10
3       1      0      1
4    126    106
5      -300.00      -300.00     -5484.82
6      -200.00      -300.00     -5483.59
7      -100.00      -300.00     -5481.34
8         0.00      -300.00     -5479.30
9       100.00      -300.00     -5478.44
10      200.00      -300.00     -5478.95
11      300.00      -300.00     -5480.11
12 .
13 .
14 .
```

An example of .gdlay is shown in Listing 4.3 and Figure 4.2. The detailed format of the .gdlay is:

- The first line:
  the number of grid interfaces (NI, 4 in this example).

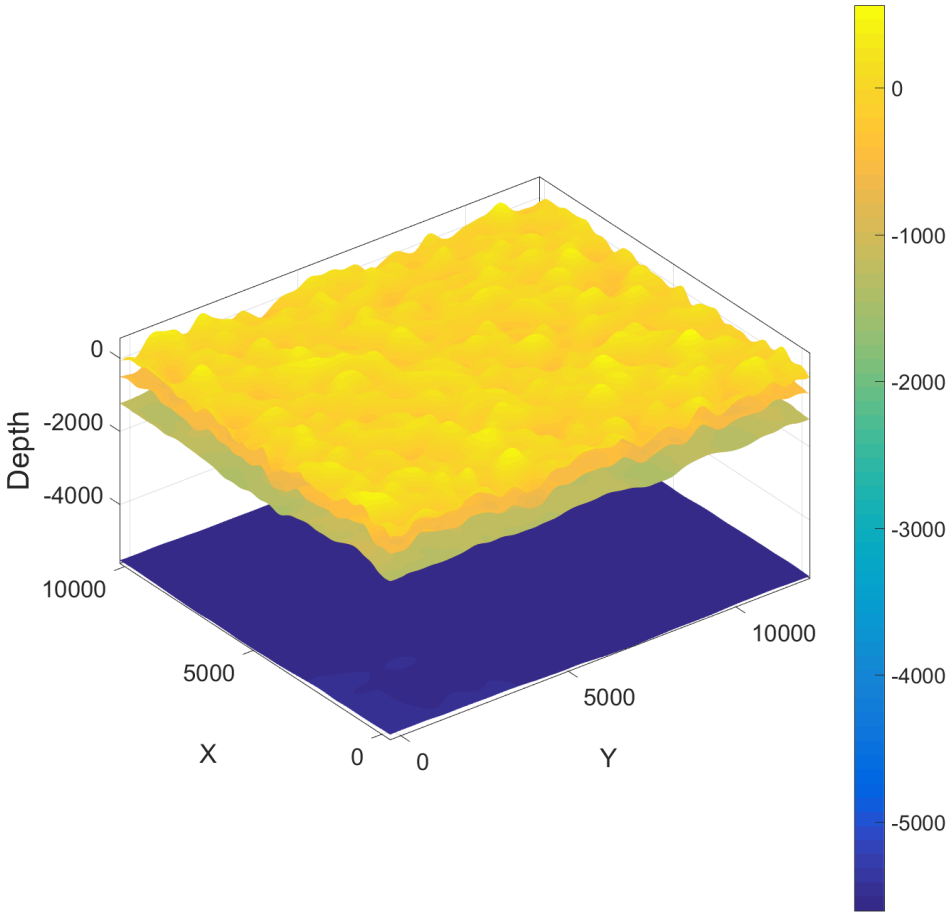


Figure 4.2: Examples of the grid interfaces in the .gdlay file.

- 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.
  - 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 5

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.

5.1 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.

5.2 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 as

$$\langle var \rangle = \frac{1}{\Delta V} \int_{k-1/2}^{k+1/2} \int_{j-1/2}^{j+1/2} \int_{i-1/2}^{i+1/2} var \, dxdydz, \tag{5.1}$$

  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

$$\langle var \rangle^H = \frac{\Delta V}{\int_{k-1/2}^{k+1/2} \int_{j-1/2}^{j+1/2} \int_{i-1/2}^{i+1/2} \frac{1}{var} dxdydz}, \tag{5.2}$$

  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 Moczo et al. [2002, 2014]. 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 [Jiang and Zhang, 2021], 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 5.1).

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	NA
`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 5.1: Medium parameterization on difference parameters for different medium types.

5.3 Parameters in Main Par File

Listing 5.1: 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 5.1),

- `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 5.4
    - `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 5.5.
    - `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 5.2 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.

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

The Earth structure of many regions has following features:

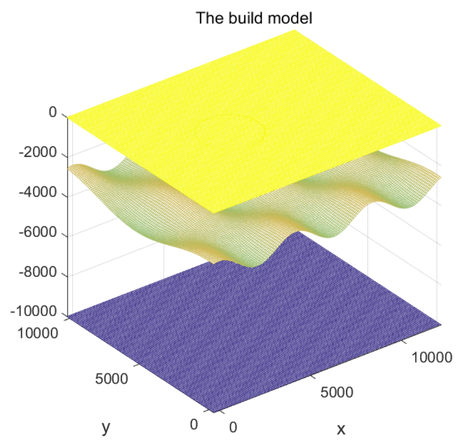


Figure 5.1: Layer-based velocity model with topographic internal interfaces.

- 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 5.2: Example of .md3lay file

```
# media type
elastic_isotropic
# number of interfaces (NI)
4
# NX  NY    X0     Y0     DX    DY    (interface mesh)
 126 106 -300.0 -300.0 100.0 100.0
# elevation   rho    rho_grad rho_pow    vp   vp_grad  vp_pow    vs    vs_grad  vs_pow
    0.0      1000.0    0.2     1.0     1500.0   0.2     1.0     1000.0   0.2     1.0
    0.0      1000.0    0.2     1.0     1500.0   0.2     1.0     1000.0   0.2     1.0
    0.0      1000.0    0.2     1.0     1500.0   0.2     1.0     1000.0   0.2     1.0
    0.0      1000.0    0.2     1.0     1500.0   0.2     1.0     1000.0   0.2     1.0
    0.0      1000.0    0.2     1.0     1500.0   0.2     1.0     1000.0   0.2     1.0
    0.0      1000.0    0.2     1.0     1500.0   0.2     1.0     1000.0   0.2     1.0
    0.0      1000.0    0.2     1.0     1500.0   0.2     1.0     1000.0   0.2     1.0
    .
    .
    .
```

An example of .md3lay is shown in Listing 5.2 and Figure 5.1. 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 [Dziewonski and Anderson, 1981] for detail.

  - `elastic_vti_thomsen`:
    density and five Thomsen parameters, ρ, α_0, β_0, ε, δ, γ. Please see [Thomsen, 1986] 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 [Thomsen, 1986] 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 (-300.0 and -300.0 for this example);
  - `DX` and `DY`: spacing between sampling points along x and y (100.0 and 100.0 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.

5.5 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 5.3: Example of .mdgrd file

```
# media type
elastic_isotropic
# number of layers (NL)
3
# number of grids in the z-direction for each layer of the NL layers
 11 21 101
# NX  NY  X0     Y0    DX    DY    (mesh in xOy of the given model)
126 106 -300.0 -300.0 100.0 100.0
# elevation  rho     vp      vs
   0.0      1000.0  1510.0  1020.0
   0.0      1000.0  1510.0  1020.0
   0.0      1000.0  1510.0  1020.0
   0.0      1000.0  1510.0  1020.0
   0.0      1000.0  1510.0  1020.0
   0.0      1000.0  1510.0  1020.0
   0.0      1000.0  1510.0  1020.0
   .
   .
   .
```

An example of .md3grd is shown in Listing 5.3. The detailed format of the .md3grd is:

- The first line:
  sets the medium type, similar to that of .md3lay. Please see descriptions in 5.4.
- 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 (-300.0 and -300.0 for this example);
  - `DX` and `DY`: spacing between sampling points along x and y (100.0 and 100.0 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 6

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:

$$\rho \frac{\partial \mathbf{v}}{\partial t} = \nabla \cdot \sigma + \mathbf{F}, \tag{6.1}$$

$$\frac{\partial \sigma}{\partial t} = \mathbf{C} : \frac{1}{2} (\nabla \mathbf{v} + \mathbf{v} \nabla) - \dot{\mathbf{M}}. \tag{6.2}$$

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.

6.1 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",
```

6.2 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 6.1: 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:
```

```
16 #       0 : 6 moment components,
17 #       1 : 3 angles, mu, slip rate or D, A (mu <0 means to use internal mu value)
18 2 0
19 # flag for location
20 #   1st value: meaning of the location: 0 computational coordinate, 1 physical
       coordinate
21 #   2nd value: 0 absolute axis or 1 depth relative to the free surface of the third
       coordinate
22 1 0
23 # location of each source
24 #   sx sy sz
25 #80 49 50
26 #80 49 50
27 8020 4930 -950
28 # stf and cmp
29 0.0 ricker 2.0 0.5   # t0  stf_name  ricker_fc ricker_t0
30 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.μ <0 means to use internal μ value.
- 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 6.2: 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 6.3: 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 6.1 (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 6.1 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 6.1) 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 6.4),

  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 6.1) 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.

Table 6.1: 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

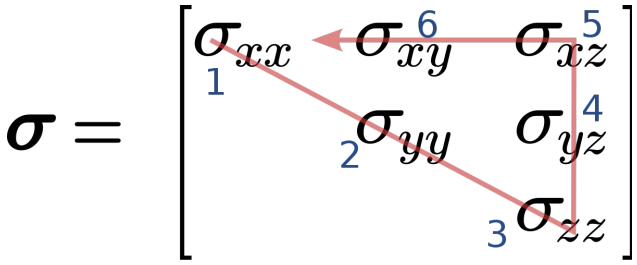


Figure 6.1: Viogt notation (From: Wiki)

Listing 6.4: Source input file using discrete STF

```
# name_of_this_input_src
evt_1_stf_value
# 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
```

Chapter 7

Output

The content of the output and the output path are set in the shell script. Grid, metric, medium, PGV/PGA/PGD, station, receiver, slice, snapshot ,check_nan information can be selected for output or not. There are several key words in the running shell script.

Listing 7.1: OUTPUT_DIR

```
#-- output and conf
PROJDIR=~/work/cgfd3d-wave-el/cfspml
EVTNM=codetest
... ...

PAR_FILE=${PROJDIR}/test.json
GRID_DIR=${PROJDIR}/output
MEDIA_DIR=${PROJDIR}/output
SOURCE_DIR=${PROJDIR}/output
OUTPUT_DIR=${PROJDIR}/output

#RUN_SCRIPT_FILE=${PROJDIR}/runscript.sh
RUN_SCRIPT_FILE=${PROJDIR}/runscript.lsf
... ...

#-- tmp dir should be in local, here is for test
TMP_DIR=${PROJDIR}/scratch
mkdir -p ${TMP_DIR}
```

- `PROJDIR:`
  total output directory, where you can find all the output file.
- `EVTNM:`
  name of input source.
- `PAR_FILE:`
  modeling parameter output file.
- `GRID_DIR:`
  directory for grid and metrices export.
- `MEDIA_DIR:`
  directory for medium export .
- `SOURCE_DIR:`
  directory for source export .
- `OUTPUT_DIR:`
  directory for station, receiver_line, slice, snapshot,check_nan export.

- `RUN_SCRIPT_FILE:`
  run script file showing the running information.

Listing 7.2: Example of export

```
"is_export_grid" : 1,
"grid_export_dir"   : "$GRID_DIR",
... ...

"is_export_metric" : 1,
... ...

"is_export_media" : 1,
"media_export_dir"  : "$MEDIA_DIR",
... ...

"is_export_source" : 1,
"source_export_dir"  : "$SOURCE_DIR",
```

- `is_export_grid:`
  if export the coordinates of grid.
  - 0: do not export,
  - 1: export.
- `grid_export_dir:`
  set the output dir for grid exporting if `is_export_grid : 1`.
- `is_export_metric:`
  if export metrices for curvilinear and cartesian grid transformation.
- `is_export_media:`
  if export the discreted medium parameters.
- `media_export_dir:`
  set the output dir for medium exporting if `is_export_media : 1`.
- `is_export_source:`
  if export the internal discreted source.
- `source_export_dir:`
  set the output dir for source exporting if `is_export_source : 1`.

For saving modeling results, we provide several choice. For saving information on several single stations, you can set `station` in station file(Listing 7.3). For receivers equaldistant on a line, `receiver_line`(Listing 7.4) can be used. For several single slice, `slice`(Listing 7.5) can be used. If mang slices are needed and you want to save information of a volume, `snapshot`(Listing7.6) can be used.

Listing 7.3: Example of setting stations

```
create_station_file()
{
cat << ieof > $PROJDIR/test.station
# number of station
1
# name is_grid_indx is_3dim_depth  x y z
r1  0  1  1000 1000 0
ieof

echo "+ created $PROJDIR/test.station"
}
```

- `test.station:`
  the station file genarated for input.
- `number of station:`
  total number of station.
- `name:`
  set station name.
- `is_grid_indx:`
  - 1: the station position `x y z` is grid index.
  - 0: the station position `x y z` is physical distance.
- `is_3dim_depth:`
  - 1: the station position `z` is depth, which means that, station is at the surface when `z` is 0.
  - 0: the station position `z` is not calculated by depth.
- `x y z:` set station position here.

Listing 7.4: Example of setting receiver_line

```
"receiver_line" : [
{
        "name" : "line_x_1",
        "grid_index_start"    : [  0, 49, 59 ],
        "grid_index_incre"    : [  1,  0,  0 ],
        "grid_index_count"    : 20
},
{
        "name" : "line_y_1",
        "grid_index_start"    : [ 19, 49, 59 ],
        "grid_index_incre"    : [  0,  1,  0 ],
        "grid_index_count"    : 20
}
],
```

- `name:`
  set receiver_line name.
- `grid_index_start:`
  the `x,y,z` of starting point of the receiver_line, the number you enter should be grid_index.
- `grid_index_incre:`
  the increment of receiver_grid_index in `x,y,z` direction, the example `[ 1,0,0 ]` means receivers are arranged along the x direction, and the distance between receivers is 1 grid point.
- `grid_index_count:`
  the number of receiver on the receiver line.

More or less receiver_line can be set by copying or deleting curly brackets `{}` and what's inside them.

Listing 7.5: Example of setting slice

```
"slice" : {
        "x_index" : [ 90, 149 ],
        "y_index" : [ 90, 149 ],
        "z_index" : [ 30, 59 ]
},
```

- `x_index:`
  set slice at x=`x_index`, the number you enter should be grid_index. The example `[ 90, 149 ]` means setting two slices, one at 90th grid point and one at 149th grid point in x direction.

- `y_index:`
  set slice at y=`y_index`, the number you enter should be grid_index.
- `z_index:`
  set slice at z=`z_index`, the number you enter should be grid_index.

Listing 7.6: Example of setting snapshot

```
"snapshot" : [
{
        "name" : "volume_vel",
        "grid_index_start" : [ 0, 0, $(( NZ - 1 )) ],
        "grid_index_count" : [ $NX, $NY, 1 ],
        "grid_index_incre" : [  1, 1, 1 ],
        "time_index_start" : 0,
        "time_index_incre" : 1,
        "save_velocity" : 1,
        "save_stress"   : 1,
        "save_strain"   : 0
}
],
```

- `name:`
  set volume name.
- `grid_index_start:`
  the `x,y,z` of starting point of the volume, the number you enter should be grid_index.
- `grid_index_count:`
  set the size of volume, the number you enter should be grid_index. The example `[ $NX,$NY,1 ]` means the volume is NX grid points long in x direction, NY grid points long in y direction, and 1 grid point long in z direction.
- `grid_index_incre:`
  the grid point interval of storing in `x,y,z` direction, the number you enter should be grid_index.
- `time_index_start:`
  set the time to start stroing, the number you enter should be time_index.
- `time_index_incre:`
  set time interval of stroing, the number you enter should be time_index. The example `1` means stroing once every $1 * \Delta t$.
- `save_velocity:`
  save velocity information of volume.
  -0: do not save velocity,
  -1: save velocity.
- `save_stress:`
  save stress information of volume.
  -0: do not save stress,
  -1: save stress.
- `save_strain:`
  save strain information of volume.
  -0: do not save strain,
  -1: save strain.

Listing 7.7: Example of output check_nan

```
"check_nan_every_nummber_of_steps" : 0,
"output_all" : 0
```

- `check_nan_every_number_of_steps:`
  if check value nan at each step.
  -0 : do not check value,
  -1 : check value.
- `output_all:`
  if export the all the information at all steps for cheaking.
  -0 : do not export,
  -1 : export.

After running, there are output files in the output dir you set in configuration file. Information on individual station and receiver line is stored in SAC format, the rest is stored in NETCDF format. Table7.1 shows the output filename corresponding to the output contents.

OUTPUT	OUTPUT_FILENAME
Gird	coord_px***mpi_process_number***_py***mpi_process_number***.nc
metrice	metric_px***mpi_process_number***_py***mpi_process_number***.nc
media	media_px***mpi_process_number***_py***mpi_process_number***.nc
PGV/A/D	PG_V_A_D_px***mpi_process_number***_py***mpi_process_number***.nc
station	***source_name*** . ***station_name*** . ***variable*** .sac
receiver_line	***source_name*** . ***receiver_line_name*** .no***grid_index*** . ***variable*** .sac
slice	slice***x/y/z_i/j/kslice_index***_px***mpi_process_number***_py***mpi_process_number***.nc
snapshot	***snapshot_name***_px***mpi_process_number***_py***mpi_process_number***.nc
check_nan	w3d_***mpi_process_number_mpi_process_number***_it***step***.nc

Table 7.1: Output filename conrresponding to the output contents.

Chapter 8

Visualization of the Results

The plotting codes in both matlab and python are provided under directors mfiles_ and pyfiles_. The codes mainly contain the plotting of the coordinate, media configuration, metric, traces along receiver line and stations.

8.1 Showing the results with matlab

- `draw_grid_coord.m` Draw the grid and coordinates configuration.
- `draw_media_pcolor.m` Draw the media cross section using pcolor. The media parameters include the p wave velocity V_p and the density ρ for acoustic isotropic medium, and V_p, ρ, s wave velocity V_s, elastic parameters λ and μ for elastic isotropic and elastic VTI medium.
- `draw_media_surf.m` Draw the media cross section using surf.
- `draw_media_surf_multi.m` Draw the three dimensional cross sections of the media.
- `draw_metric_pcolor.m` Draw the metric parameters with pcolor.
- `draw_metric_surf.m` Draw the metric parameters with surf.
- `draw_metric_surf_multi.m` Draw the three dimensional cross sections of the metric parameters with surf.
- `draw_PG_V_A_D.m` Draw the particle ground motion intensity. The drawable parameters include peak velocity, acceleration and displacement.
- `draw_seismo_line.m` Draw the traces along the receiver line.
- `draw_seismo_recv.m` Draw the single trace at each receiver given a range of station ID.
- `draw_slice_pcolor.m` Draw the slice using pcolor. The slice id should be consistent with that in the main par file.
- `draw_slice_surf.m` Draw the slice using surf.
- `draw_slice_surf_multi.m` Draw the three dimensional cross sections of the slice with surf.
- `draw_snap_pcolor.m` Draw the snapshot using pcolor.
- `draw_snap_surf.m` Draw the snapshot using surf.
- `draw_snap_surf_multi.m` Draw the three dimensional cross sections of the snapshot with surf.

8.2 show the results with python

The python code provided with CGFD3D has the same functionality with the matlab code. The open source packages that are accessible in python will enrich your choices of data visualization, and you can make modifications based on the following programs.

1. `draw_grid_coord.py` Draw the grid and coordinates configuration.
2. `draw_media_pcolor.py` Draw the media cross section using pcolor.
3. `draw_metric_pcolor.py` Draw the metric parameters with pcolor.
4. `draw_seismo_line.py` Draw the traces along the receiver line.
5. `draw_seismo_station.py` Draw the single trace at each receiver given a range of station ID.
6. `draw_slice_pcolor.py` Draw the slice using pcolor.
7. `draw_snap_pcolor.py` Draw the snapshot using pcolor.

8.3 A general example of usage

The codes have similar structures. And here we have a general process of using the code.
The inputs are all read from the jason file. To use the code, you need to first modify the path and name of the input .jason file and the output directory.
Then you need to set the parameters which should also be consistent with the parameters in jason file, for example, the number of grid points and the unit of the parameters, or it may report errors.
You can check the main parameter file to make sure that the content you want to plot is included in the output, for example, sometimes only part of the snapshots are output by the program.
After the checking, you can begin to set the ploting parameters, for examle, which slice to plot while using `draw_slice_pcolor.m`.
Finally you may want to choose whether to print and save the plot or not by switching the `flag` between 1 and 0.

Chapter 9

Appendix A

The solution of a physical problem is based on the solution of the governing equations. In order to adapt to the characteristics of complex shapes in discrete grids, the grids generated by different physical subregions are not the same, and the governing equations are also different in different grid systems. The governing equations in Cartesian coordinate system and curvilinear coordinate system are introduced in the following.

9.1 The Governing Equations in Cartesian Coordinates

The Cartesian coordinate system is used as the background coordinate system in the multi-block grid. The high performance scalability of the numerical method under Cartesian grid is superior to the curve grid numerical method with the same precision, so the numerical method under Cartesian grid is more suitable for accurate numerical simulation.

- The momentum equation:

$$\rho v_{,t} = \nabla \cdot \sigma + f, \tag{9.1}$$

- The geometric relationship:

$$\varepsilon = \nabla u + (\nabla u)^T, \tag{9.2}$$

- Constitutive relation (Hooke's law):

$$\sigma = C : \varepsilon. \tag{9.3}$$

The tensor C is Lame constants, which can be reduced to λ and μ in simple linear elastic media. Therefore, in the Cartesian coordinate system, the wave equation of any inhomogeneous medium can be expressed in the following different forms:

- Displacement equation of second order:

$$\rho u_{i,tt} = (\lambda u_{k,k})_{,i} + (\mu u_{i,j})_{,j} + (\mu u_{j,i})_{,j} + f_i, \tag{9.4}$$

- Displacement stress system:

$$\rho u_{i,tt} = \sigma_{ij,j} + f_i \tag{9.5}$$
$$\sigma_{ij} = \lambda u_{k,k}\delta_{ij} + \mu(u_{i,j} + u_{j,i}) \tag{9.6}$$
$$\tag{9.7}$$

- First order velocity stress equation:

$$\rho v_{i,t} = \sigma_{ij,j} + f_i \tag{9.8}$$
$$\sigma_{ij,t} = \lambda v_{k,k}\delta ij + \mu(v_{i,j} + v_{j,i}) \tag{9.9}$$
$$\tag{9.10}$$

This program selects the first order velocity stress equation as the governing equation.

9.2 The Governing Equations in Curvilinear Coordinates

When the study area contains undulating terrain and irregular interface, the Cartesian grid will no longer be able to discrete the medium accurately. In this case, it is necessary to introduce the curve grid and apply the fitting grid in the simulation of undulating terrain.

9.2.1 Coordinate Transformation in Curvilinear Coordinate

In curvilinear coordinate system, the essence of the generation of body-fitting mesh is coordinate transformation, which maps the irregular region in the physical region to the regular region in the computational region.Suppose that the grid points in the calculation area is:(ξ,η,ζ),The coordinates of the corresponding physical region are:(x,y,z):

$$x = x(\xi,\eta,\zeta) \tag{9.11}$$
$$y = y(\xi,\eta,\zeta) \tag{9.12}$$
$$z = z(\xi,\eta,\zeta) \tag{9.13}$$
$$\tag{9.14}$$

According to the above mapping relations, the corresponding coordinate transformation coefficients can be solved

$$x_{,\xi} = D_\xi x, \quad x_{,\eta} = D_\eta x, \quad x_{,\zeta} = D_\zeta x, \tag{9.15}$$
$$x_{,\xi} = D_\xi y, \quad x_{,\eta} = D_\eta y, \quad x_{,\zeta} = D_\zeta y, \tag{9.16}$$
$$x_{,\xi} = D_\xi z, \quad x_{,\eta} = D_\eta z, \quad x_{,\zeta} = D_\zeta z, \tag{9.17}$$
$$\tag{9.18}$$

By using the identity:

$$x_{,q}q_{,p} = \delta_{xp}, \tag{9.19}$$
$$y_{,q}q_{,p} = \delta_{yp}, \tag{9.20}$$
$$z_{,q}q_{,p} = \delta_{zp}, \tag{9.21}$$
$$\tag{9.22}$$

The $q \in \xi,\eta,\zeta, p \in x,y,z$ are the Dirac delta function.The coordinate transformation coefficient can be solved

$$\xi_{,x} = (y_{,\eta}z_{,\zeta} - y_{,\zeta}z_{,\eta})J^{-1}, \tag{9.23}$$
$$\xi_{,y} = (x_{,\zeta}z_{,\eta} - x_{,\eta}z_{,\zeta})J^{-1}, \tag{9.24}$$
$$\xi_{,z} = (x_{,\eta}z_{,\zeta} - x_{,\zeta}z_{,\eta})J^{-1}, \tag{9.25}$$
$$\eta_{,x} = (y_{,\zeta}z_{,\xi} - y_{,\xi}z_{,\zeta})J^{-1}, \tag{9.26}$$
$$\eta_{,y} = (x_{,\xi}z_{,\zeta} - x_{,\zeta}z_{,\xi})J^{-1}, \tag{9.27}$$
$$\eta_{,z} = (x_{,\zeta}y_{,\xi} - x_{,\xi}z_{,\zeta})J^{-1}, \tag{9.28}$$
$$\zeta_{,x} = (y_{,\xi}z_{,\eta} - y_{,\eta}z_{,\xi})J^{-1}, \tag{9.29}$$
$$\zeta_{,y} = (x_{,\eta}z_{,\xi} - x_{,\xi}z_{,\eta})J^{-1}, \tag{9.30}$$
$$\zeta_{,z} = (x_{,\xi}z_{,\eta} - x_{,\eta}z_{,\xi})J^{-1}, \tag{9.31}$$
$$\tag{9.32}$$

J is Jacobian matrix:

$$J = \begin{vmatrix} x_{,\xi} & x_{,\eta} & x_{,\zeta} \\ y_{,\xi} & y_{,\eta} & y_{,\zeta} \\ z_{,\xi} & z_{,\eta} & z_{,\zeta} \end{vmatrix}$$

To ensure that mesh generation does not twist and intersect, the Jacobian determinant cannot be zero.

9.2.2 The Governing Equations in Curvilinear Coordinates

In curvilinear coordinates, let a_1, a_2, a_3 be the covariant basis vector for ξ, η, ζ in the physical region.a^1, a^2, a^3 is the contravariant basis vector in the ξ, η, ζ direction in the physical region, and i, j, k is the basis vector in the x, y, z direction in the Cartesian coordinate system.So,the covariant basis vectors are:

$$a_1 = x_{,\xi}\boldsymbol{i} + y_{,\xi}\boldsymbol{j} + z_{,\xi}\boldsymbol{k}, \tag{9.33}$$
$$a_2 = x_{,\eta}\boldsymbol{i} + y_{,\eta}\boldsymbol{j} + z_{,\eta}\boldsymbol{k}, \tag{9.34}$$
$$a_3 = x_{,\zeta}\boldsymbol{i} + y_{,\zeta}\boldsymbol{j} + z_{,\zeta}\boldsymbol{k}, \tag{9.35}$$
$$\tag{9.36}$$

The contravariant basis vectors are:

$$a^1 = \xi_{,x}\boldsymbol{i} + \xi_{,y}\boldsymbol{j} + \xi_{,z}\boldsymbol{k}, \tag{9.37}$$
$$a^2 = \eta_{,x}\boldsymbol{i} + \eta_{,y}\boldsymbol{j} + \eta_{,z}\boldsymbol{k}, \tag{9.38}$$
$$a^3 = \zeta_{,x}\boldsymbol{i} + \zeta_{,y}\boldsymbol{j} + \zeta_{,z}\boldsymbol{k}, \tag{9.39}$$
$$\tag{9.40}$$

According to the chain rule, the first-order speed-degree stress equation in Cartesian grid can be extended to curvilinear coordinate system. Thus the speed-stress equation in curvilinear coordinates can be formulated by the following momentum equation and Hooke's law, ignoring the source term.
The momentum equations are expressed as:

$$\rho\frac{\partial v_x}{\partial t} = \xi_{,x}\sigma_{xx,\xi} + \xi_{,y}\sigma_{xy,\xi} + \xi_{,z}\sigma_{xz,\xi} \tag{9.41}$$
$$+ \eta_{,x}\sigma_{xx,\eta} + \eta_{,y}\sigma_{xy,\eta} + \eta_{,z}\sigma_{xz,\eta} \tag{9.42}$$
$$+ \zeta_{,x}\sigma_{xx,\zeta} + \zeta_{,y}\sigma_{xy,\zeta} + \zeta_{,z}\sigma_{xz,\zeta}, \tag{9.43}$$
$$\rho\frac{\partial v_y}{\partial t} = \xi_{,x}\sigma_{xy,\xi} + \xi_{,y}\sigma_{yy,\xi} + \xi_{,z}\sigma_{yz,\xi} \tag{9.44}$$
$$+ \eta_{,x}\sigma_{xy,\eta} + \eta_{,y}\sigma_{yy,\eta} + \eta_{,z}\sigma_{yz,\eta} \tag{9.45}$$
$$+ \zeta_{,x}\sigma_{xy,\zeta} + \zeta_{,y}\sigma_{yy,\zeta} + \zeta_{,z}\sigma_{yz,\zeta}, \tag{9.46}$$
$$\rho\frac{\partial v_z}{\partial t} = \xi_{,x}\sigma_{xz,\xi} + \xi_{,y}\sigma_{yz,\xi} + \xi_{,z}\sigma_{zz,\xi} \tag{9.47}$$
$$+ \eta_{,x}\sigma_{xz,\eta} + \eta_{,y}\sigma_{yz,\eta} + \eta_{,z}\sigma_{zz,\eta} \tag{9.48}$$
$$+ \zeta_{,x}\sigma_{xz,\zeta} + \zeta_{,y}\sigma_{yz,\zeta} + \zeta_{,z}\sigma_{zz,\zeta}, \tag{9.49}$$

The generalized Hooke's law is denoted as:

$$\frac{\partial \sigma_{xx}}{\partial t} = (\lambda+2\mu)\xi_{,x}v_{x,\xi} + \lambda\xi_{,y}v_{y,\xi} + \lambda\xi_{,z}v_{z,\xi} \tag{9.50}$$
$$+ (\lambda+2\mu)\eta_{,x}v_{x,\eta} + \lambda\eta_{,y}v_{y,\eta} + \lambda\eta_{,z}v_{z,\eta} \tag{9.51}$$
$$+ (\lambda+2\mu)\zeta_{,x}v_{x,\zeta} + \lambda\zeta_{,y}v_{y,\zeta} + \lambda\zeta_{,z}v_{z,\zeta}, \tag{9.52}$$
$$\frac{\partial \sigma_{yy}}{\partial t} = \lambda\xi_{,x}v_{x,\xi} + (\lambda+2\mu)\xi_{,y}v_{y,\xi} + \lambda\xi_{,z}v_{z,\xi} \tag{9.53}$$
$$+ \lambda\eta_{,x}v_{x,\eta} + (\lambda+2\mu)\eta_{,y}v_{y,\eta} + \lambda\eta_{,z}v_{z,\eta} \tag{9.54}$$
$$+ \lambda\zeta_{,x}v_{x,\zeta} + (\lambda+2\mu)\zeta_{,y}v_{y,\zeta} + \lambda\zeta_{,z}v_{z,\zeta}, \tag{9.55}$$
$$\frac{\partial \sigma_{zz}}{\partial t} = \lambda\xi_{,x}v_{x,\xi} + \lambda\xi_{,y}v_{y,\xi} + (\lambda+2\mu)\xi_{,z}v_{z,\xi} \tag{9.56}$$
$$+ \lambda\eta_{,x}v_{x,\eta} + \lambda\eta_{,y}v_{y,\eta} + (\lambda+2\mu)\eta_{,z}v_{z,\eta} \tag{9.57}$$
$$+ \lambda\zeta_{,x}v_{x,\zeta} + \lambda\zeta_{,y}v_{y,\zeta} + (\lambda+2\mu)\zeta_{,z}v_{z,\zeta}, \tag{9.58}$$
$$\frac{\partial \sigma_{xy}}{\partial t} = \mu(\xi_{,y}v_{x,\xi} + \xi_{,x}v_{y,\xi}) \tag{9.59}$$
$$+ \mu(\eta_{,y}v_{x,\eta} + \eta_{,x}v_{y,\eta}) \tag{9.60}$$
$$+ \mu(\zeta_{,y}v_{x,\zeta} + \zeta_{,x}v_{y,\zeta}), \tag{9.61}$$
$$\frac{\partial \sigma_{xz}}{\partial t} = \mu(\xi_{,z}v_{x,\xi} + \xi_{,x}v_{z,\xi}) \tag{9.62}$$
$$+ \mu(\eta_{,z}v_{x,\eta} + \eta_{,x}v_{z,\eta}) \tag{9.63}$$
$$+ \mu(\zeta_{,z}v_{x,\zeta} + \zeta_{,x}v_{z,\zeta}), \tag{9.64}$$
$$\frac{\partial \sigma_{yz}}{\partial t} = \mu(\xi_{,z}v_{y,\xi} + \xi_{,y}v_{z,\xi}) \tag{9.65}$$
$$+ \mu(\eta_{,z}v_{y,\eta} + \eta_{,y}v_{z,\eta}) \tag{9.66}$$
$$+ \mu(\zeta_{,z}v_{y,\zeta} + \zeta_{,y}v_{z,\zeta}). \tag{9.67}$$

Chapter 10

Appendix B

There are mainly staggered gird and collocated grid to solve the system of 1st-order velocity-stress equations in finite difference numerical simulation. In the staggered grid, velocity component and stress component are positioned half a grid apart, while in collocated grid velocity and stress are located on the same mesh. Because of the high calculation efficiency and low truncation error, staggered gird finite difference is used to solve cartesian grid cases in this work. In curvilinear grid, however, the spatial derivatives of the individual components along the three directions are required. And staggered grid is missing some information, so interpolation is needed. Interpolation leads to a decrese in computation accuracy and efficiency, thus, the collocated grid is used for curvilinear grid cases in this work.The following is a brief introduction to the finite difference scheme used in the code.

10.1 Collocated-grid DRP/opt MacCormack Scheme

MacCormack scheme realizes the dissipation of non-physical high frequency components of the wave field by splitting the central difference into forward one-sided difference and backward ont-sided difference. It does not require explicit filter and can achieve truncation error of central difference. DRP/opt MacCormack schemes of high precision (Hixon and Hixon [1997],Zhang and Chen [2006])greatly improved mesh resolution.

10.1.1 Space Discretization

For 3D 1st-order partial differential equations like

$$W = A\frac{\partial W}{\partial \xi} + B\frac{\partial W}{\partial \eta} + C\frac{\partial W}{\partial \zeta} \tag{10.1}$$

The one-sided difference operators:

$$\hat{W}_i^F = \frac{1}{\triangle \eta}\sum_{k=-1}^{k=3} a_n W_{i+k} \tag{10.2}$$

$$\hat{W}_i^B = \frac{1}{\triangle \eta}\sum_{k=-1}^{k=3} -a_n W_{i-k} \tag{10.3}$$

where $\hat{W}_i^F$ and$\hat{W}_i^B$denote the forward and backward difference operatos with respect to x, y, z, coefficients are

$$a_{-1} = -0.30874, \tag{10.4a}$$
$$a_0 = -0.6326, \tag{10.4b}$$
$$a_1 = 1.2330, \tag{10.4c}$$
$$a_2 = -0.3334, \tag{10.4d}$$
$$a_3 = 0.04168. \tag{10.4e}$$

The stencil width is 7 grid points,when close to the boundary, the difference scheme with a smaller stencil width is used.

3 gird points MacCormack difference scheme:

$$\hat{W}_i^F = \frac{1}{\triangle\eta}\sum_{k=-1}^{k=1} a_n W_{i+k} \tag{10.5}$$

$$\hat{W}_i^B = \frac{1}{\triangle\eta}\sum_{k=-1}^{k=1} -a_n W_{i-k} \tag{10.6}$$

where coefficients are

$$a_0 = -1.0, \tag{10.7a}$$

$$a_1 = 1.0. \tag{10.7b}$$

$$\tag{10.7c}$$

5 grid points MacCormack difference scheme:

$$\hat{W}_i^F = \frac{1}{\triangle\eta}\sum_{k=-1}^{k=2} a_n W_{i+k} \tag{10.8}$$

$$\hat{W}_i^B = \frac{1}{\triangle\eta}\sum_{k=-1}^{k=2} -a_n W_{i-k} \tag{10.9}$$

where coefficients are

$$a_{-1} = -\frac{7}{6}, \tag{10.10a}$$

$$a_0 = \frac{8}{6}, \tag{10.10b}$$

$$a_1 = -\frac{1}{6}. \tag{10.10c}$$

Here, 8 one-sided difference combinations are used.

$$\hat{L}^{BBB} = A\hat{W}^B + B\hat{W}^B + C\hat{W}^B, \tag{10.11a}$$

$$\hat{L}^{FFB} = A\hat{W}^F + B\hat{W}^F + C\hat{W}^B, \tag{10.11b}$$

$$\hat{L}^{FFF} = A\hat{W}^F + B\hat{W}^F + C\hat{W}^F, \tag{10.11c}$$

$$\hat{L}^{BBF} = A\hat{W}^B + B\hat{W}^B + C\hat{W}^F, \tag{10.11d}$$

$$\hat{L}^{BFB} = A\hat{W}^B + B\hat{W}^F + C\hat{W}^B, \tag{10.11e}$$

$$\hat{L}^{FBB} = A\hat{W}^F + B\hat{W}^B + C\hat{W}^B, \tag{10.11f}$$

$$\hat{L}^{FBF} = A\hat{W}^F + B\hat{W}^B + C\hat{W}^F, \tag{10.11g}$$

$$\hat{L}^{BFF} = A\hat{W}^B + B\hat{W}^F + C\hat{W}^F, \tag{10.11h}$$

10.1.2 Time Discretization

Multi-step and high-order Runge-Kutta is popular in partial differential numerical solutions(Hu et al. [1996], Zhang et al. [2012]).Here, four stages Runge-Kutta scheme is used.The following is the steps.

$$h^{(1)} =\triangle\, t\hat{L}^{FFF}(W^n), \tag{10.12a}$$
$$h^{(2)} =\triangle\, t\hat{L}^{BBB}(W^n+\alpha_2 h^{(1)}), \tag{10.12b}$$
$$h^{(3)} =\triangle\, t\hat{L}^{FFF}(W^n+\alpha_3 h^{(2)}), \tag{10.12c}$$
$$h^{(4)} =\triangle\, t\hat{L}^{BBB}(W^n+\alpha_4 h^{(3)}), \tag{10.12d}$$
$$W^{n+1} = W^n+\beta_1 h^{(1)}+\beta_2 h^{(2)}+\beta_3 h^{(3)}+\beta_4 h^{(4)}, \tag{10.12e}$$

equation10.12 is abbreviated as $W^{n+1}=\hat{L}^{FFF}\hat{L}^{BBB}\hat{L}^{FFF}\hat{L}^{BBB}W^n$, where $\hat{L}^{FFF}=A\hat{W}^F+B\hat{W}^F+C\hat{W}^F$, $\hat{L}^{BBB}=A\hat{W}^B+B\hat{W}^B+C\hat{W}^B$, $\triangle t$ is the time step, the difference coefficients are

$$\alpha_1=0.0,\beta_1=\frac{1}{6}, \tag{10.13a}$$
$$\alpha_2=0.5,\beta_2=\frac{1}{3}, \tag{10.13b}$$
$$\alpha_3=0.5,\beta_3=\frac{1}{3}, \tag{10.13c}$$
$$\alpha_4=1.0,\beta_4=\frac{1}{6}. \tag{10.13d}$$

4 stages Runge-Kutta can be represented as

$$W^{n+1}=\hat{L}^{BBB}\hat{L}^{FFF}\hat{L}^{BBB}\hat{L}^{FFF}W^n, \tag{10.14a}$$
$$W^{n+2}=\hat{L}^{FFB}\hat{L}^{BBF}\hat{L}^{FFB}\hat{L}^{BBF}W^{n+1}, \tag{10.14b}$$
$$W^{n+3}=\hat{L}^{BFB}\hat{L}^{FBF}\hat{L}^{BFB}\hat{L}^{FBF}W^{n+2}, \tag{10.14c}$$
$$W^{n+4}=\hat{L}^{FBB}\hat{L}^{BFF}\hat{L}^{FBB}\hat{L}^{BFF}W^{n+3}, \tag{10.14d}$$

10.2 Staggered-grid Finite Difference Scheme

Staggered-grid finite-difference method is one of the most popular numerical methods to simulate elastic wave propagation. In this work, the higher-order is mainly referred to the spatial disretization.And second-order leaf-frog time marching scheme is used.

10.2.1 Time Discretization

2nd-order center difference:

$$\sigma_{xx}|^{n+1/2}_{i,j,k} = \sigma_{xx}|^{n-1/2}_{i,j,k} + \Delta t[(\lambda+2\mu)v_{x,x} + \lambda(v_{y,y}+v_{z,z})]|^{n}_{i,j,k} \tag{10.15a}$$

$$\sigma_{yy}|^{n+1/2}_{i,j,k} = \sigma_{yy}|^{n-1/2}_{i,j,k} + \Delta t[(\lambda+2\mu)v_{y,y} + \lambda(v_{x,x}+v_{z,z})]|^{n}_{i,j,k} \tag{10.15b}$$

$$\sigma_{zz}|^{n+1/2}_{i,j,k} = \sigma_{zz}|^{n-1/2}_{i,j,k} + \Delta t[(\lambda+2\mu)v_{z,z} + \lambda(v_{x,x}+v_{y,y})]|^{n}_{i,j,k} \tag{10.15c}$$

$$\sigma_{xy}|^{n+1/2}_{i+1/2,j+1/2,k} = \sigma_{xy}|^{n-1/2}_{i+1/2,j+1/2,k} + \Delta t[\mu(v_{x,y}+v_{y,x})]|^{n}_{i+1/2,j+1/2,k} \tag{10.15d}$$

$$\sigma_{xz}|^{n+1/2}_{i+1/2,j,k+1/2} = \sigma_{xz}|^{n-1/2}_{i+1/2,j,k+1/2} + \Delta t[\mu(v_{x,z}+v_{z,x})]|^{n}_{i+1/2,j,k+1/2} \tag{10.15e}$$

$$\sigma_{yz}|^{n+1/2}_{i,j+1/2,k+1/2} = \sigma_{yz}|^{n-1/2}_{i,j+1/2,k+1/2} + \Delta t[\mu(v_{y,z}+v_{z,y})]|^{n}_{i,j+1/2,k+1/2}, \tag{10.15f}$$

$$v_x|^{n+1}_{i+1/2,j,k} = v_x|^{n}_{i+1/2,j,k} + \Delta t b_x[\sigma_{xx,x}+\sigma_{xy,y}+\sigma_{xz,z}]|^{n+1/2}_{i+1/2,j,k} \tag{10.15g}$$

$$v_y|^{n+1}_{i,j+1/2,k} = v_y|^{n}_{i,j+1/2,k} + \Delta t b_y[\sigma_{xy,x}+\sigma_{yy,y}+\sigma_{yz,z}]|^{n+1/2}_{i,j+1/2,k} \tag{10.15h}$$

$$v_z|^{n+1}_{i,j,k+1/2} = v_z|^{n}_{i,j,k+1/2} + \Delta t b_z[\sigma_{xz,x}+\sigma_{yz,y}+\sigma_{zz,z}]|^{n+1/2}_{i,j,k+1/2} \tag{10.15i}$$

where $b = 1/\rho$.

10.2.2 Space Discretization

In this work, 4th-order scheme is used(Graves [1996]),the following is the $v_{x,x}, v_{y,y}, v_{z,z}$.

$$v_{x,x}|_{i,j,k} = \frac{1}{\Delta h}[c_0(v_x|_{i+1/2,j,k} - v_x|_{i-1/2,j,k}) - c_1(v_x|_{i+3/2,j,k} - v_x|_{i-3/2,j,k})] \tag{10.16a}$$

$$v_{y,y}|_{i,j,k} = \frac{1}{\Delta h}[c_0(v_y|_{i,j+1/2,k} - v_y|_{i,j-1/2,k}) - c_1(v_y|_{i,j+3/2,k} - v_y|_{i,j-3/2,k})] \tag{10.16b}$$

$$v_{z,z}|_{i,j,k} = \frac{1}{\Delta h}[c_0(v_z|_{i,j,k+1/2} - v_z|_{i,j,k-1/2}) - c_1(v_z|_{i,j,k+3/2} - v_z|_{i,j,k-3/2})] \tag{10.16c}$$

where

$$c_0 = \frac{9}{8}, \tag{10.17a}$$

$$c_1 = \frac{1}{27}, \tag{10.17b}$$

Copyright

Main historical authors:

Evolution of the code:

Bibliography

A. M. Dziewonski and D. L. Anderson. Preliminary reference earth model. *Physics of the earth and planetary interiors*, 25(4):297–356, 1981.

R. W. Graves. Simulating seismic wave propagation in 3d elastic media using staggered-grid finite differences. *Bulletin of the seismological society of America*, 86(4):1091–1106, 1996.

R. Hixon and R. Hixon. On increasing the accuracy of maccormack schemes for aeroacoustic applications. In *3rd AIAA/CEAS Aeroacoustics Conference*, page 1586, 1997.

F. Hu, M. Y. Hussaini, and J. Manthey. Low-dissipation and low-dispersion runge–kutta schemes for computational acoustics. *Journal of computational physics*, 124(1):177–191, 1996.

L. Jiang and W. Zhang. Tti equivalent medium parametrization method for the seismic waveform modelling of heterogeneous media with coarse grids. *Geophysical Journal International*, 227(3):2016–2043, 2021.

P. Moczo, J. Kristek, V. Vavrycuk, R. J. Archuleta, and L. Halada. 3D heterogeneous staggered-grid finite-difference modeling of seismic motion with volume harmonic and arithmetic averaging of elastic moduli and densities. *Bulletin of the Seismological Society of America*, 92(8):3042–3066, 2002. doi: 10.1785/0120010167.

P. Moczo, J. Kristek, and M. Gális. *The finite-difference modelling of earthquake motions: waves and ruptures*. Cambridge University Press, New York, 2014. ISBN 978-1-107-02881-4.

L. Thomsen. Weak elastic anisotropy. *Geophysics*, 51(10):1954–1966, 1986.

W. Zhang and X. Chen. Traction image method for irregular free surface boundaries in finite difference seismic wave simulation. *Geophysical Journal International*, 167(1):337–353, 2006.

W. Zhang and Y. Shen. Unsplit complex frequency-shifted PML implementation using auxiliary differential equations for seismic wave modeling. *Geophysics*, 75(4):T141–T154, 2010.

W. Zhang, Z. Zhang, and X. Chen. Three-dimensional elastic wave numerical modelling in the presence of surface topography by a collocated-grid finite-difference method on curvilinear grids. *Geophysical Journal International*, 190 (1):358–378, 2012. doi: https://doi.org/10.1111/j.1365-246X.2012.05472.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-246X.2012.05472.x.