User Manual for NEXD 2D

Version 0.4

Lasse Lambrecht[1]
Andre Lamert[1]
Wolfgang Friederich[1]
Thomas Möller[1]
Marc S. Boxberg[1,2]

[1] Ruhr-Universität Bochum
Institut für Geologie, Mineralogie und Geophysik

[2] RWTH Aachen University
Aachen Institute for Advanced Study in Computational Engineering Science

Contents

Chapter 1

Introduction

This manual describes NEXD 2D (**N**odal **D**iscontinuous **G**alerkin Finite **E**lement in **X** **D**imensions) version 0.4. NEXD is a Fortran based implementation of the nodal version of the discontinuous Galerkin approach. It is designed to simulate seismic wave propagation in complex geological structures with general physical properties and is not restricted with regard to the size of the model. The programs are currently designed to work on CPUs, only. GPUs are not supported. To run the code, no programming knowledge is required.

1.1 Supplied programs

The following programs are part of the software package NEXD 2D:

- "mesher": The pre-processor designed to read in specific files related to the mesh (see chapter 3).
- "solver": The main program to simulate the wave-propagation.
- "movie": A post-processor to generate files that show the wave-field at certain time steps.

1.2 Functionality

1.2.1 General features

As of release version v0.4 the following features are included:

- Forward simulation of wave-propagation in elastic, anelastic and poroelastic media
- Wave-propagation across fractures (elastic media only)
- Full waveform inversion based on the adjoint method (elastic media only, viscoelastic media under restrictions)

Since no restrictions are imposed on size and shape of the media of interest, NEXD can be used in any situation, where the propagation of elastic waves is the key information. It is able to yield the wave field at every time step of propagating waves and the output of simulated receivers. Free surface and absorbing boundary conditions (BC) are implemented. In addition, Perfectly Matched Layers (PML) (Lambrecht et al., 2018) are available to enhance the absorbing BC.

1.2.2 Source time functions

A number of pre-defined source time functions are supported:

- Gauss
- Ricker

- cubed-sine (sin^3)
- an arbitrary discrete wavelet

With sufficient programming knowledge, it is possible to add new wavelet types to the program as described in appendix B.1.

1.2.3 Output

As possible output, the program generates seismograms from the data recorded at the stations placed in the model. Additionally, binary files for desired fields (velocity, displacement or stress) are created according to the parameters set in the parfile (see section 3.1.1 and chapter 4 for details).

1.3 License

NEXD is designed and developed by Lasse Lambrecht, Andre Lamert, Wolfgang Friederich, Thomas Möller and Marc S. Boxberg. NEXD 2D and its components, as well as documentation and some examples, are available under terms of the GNU General Public License (version 3 or higher).

1.4 Citation

Please cite Lambrecht et al. (2018) if you use NEXD. If you use poroelasticity please cite Boxberg et al. (2017) or Boxberg (2019). If you use the feature to simulate the effect of fractures please cite Möller and Friederich (2019). For applications of the inversion feature please cite Lamert (2020) or Lamert and Friederich (2019)

Chapter 2

Requirements and Installation

To install and run the software, a number of requirements have to be met.

2.1 Software requirements

To install NEXD the user needs the following software installed on his system:

- GNU Make (`https://www.gnu.org/software/make/`),
- Fortran compiler (GNU Fortran or Intel Fortran). The version 0.4 of this software has been tested using the following compiler versions:
  - Intel Fortran version 19.0
  - GNU Fortran version 4.9.2, 6.3, 7.4, 9.2 and 9.3

  Version 0.2 of NEXD 2D has also been tested with:
  - Intel Fortran version 17.0.4
  - GNU Fortran version 5.4 and 7.3
- LAPACK libraries (`http://www.netlib.org/lapack/`,
- MPI libraries, e.g. OpenMPI,
- METIS (`http://glaros.dtc.umn.edu/gkhome/metis/metis/overview`, tested with version 4.0.3) for parallel applications,
- a Python installation.

A number of Python scripts are used during the installation of NEXD 2D and as an aid in the creation of input files. To run those, a Python version (recommended 2.7, or 3.6 or higher) is required. A good and easy way to get Python is to install it via the Anaconda distribution, which is available for Windows, MacOS and Linux. It can be downloaded under `https://www.anaconda.com/download/`.

2.2 Installation

To install the software, follow these steps:

1. If not yet done, download the source code of the NEXD 2D main package by downloading it from `https://github.com/seismology-RUB/NEXD-2D`.
2. Install all software dependencies.
3. Adjust the software to your system and personal requirements by changing the Makefile appropriately (e.g., change the path to your METIS installation and set your compiler, see code listing 2.2).

4. Compile METIS.
5. Compile NEXD using the command “make all” in your console from your installation directory.

2.3 Changing the Makefile

As mentioned in step 3 of the instruction on how to install the software, the Makefile needs to be changed at certain position. The following sections need to be adjusted by the user:

```
31 # Choose your compiler:
32 F95 = mpif90
33 #F95 = mpiifort
```

Listing 2.1: Exemplary content of the Makefile.

Change these lines to select the compiler. The default compiler flags are set for optimum performance of the user and need not be adjusted. Recommended flags for developers are given by the comments in the Makefile. Please review the Makefile for more information.
Please adjust the path to your METIS library as specified in the code listing 2.2.

```
294 #  Library paths
295 #  (specify the path to your METIS library here!)
296 #
297 la = -llapack -lblas
298 lib = metis-4.0.3/libmetis.a
```

Listing 2.2: Exemplary content of the Makefile.

2.4 Clean installation directory

Below a clean tree for the installation directory with all sub-folders is shown.

```
NEXD2D installation directory
```

- `doc` Contains all files related to the documentation, including the Latex source files
- `simulations`
  - `2layer_pml_att`
    - `mesh` Contains files related to the mesh
    - `data` Contains the files: parfile, source and stations
    - `ref`
      - `att` Contains reference seismograms with attenuation and without PMLs (42 files)
      - `no_att` Contains reference seismograms without attenuation and without PMLs (42 files)
      - `pml` Contains reference seismograms without attenuation and with PMLs (42 files)
      - `pml_att` Contains reference seismograms with attenuation and with PMLs (42 files)
  - `example_inversion`
    - `mesh` Contains files related to the mesh
    - `data` Contains the files: parfile, source, stations and invpar
    - `ref` Contains reference seismograms and .vtk files (22 files)
    - `inversion` Contains the seismograms obtained from the real model
  - `example_linearSlipInterface`
    - `mesh` Contains files related to the mesh
    - `data` Contains the files: parfile, source, stations, interfaces and fracs
    - `ref`
      - `noPML` Contains reference seismograms with PML (60 files)
      - `withPML` Contains reference seismograms without PML (60 files)
  - `example_poro`
    - `mesh` Contains files related to the mesh
    - `data` Contains the files: parfile, source and stations
    - `ref` Contains reference seismograms (178 Files)
    - `tools` Contains Matlab Code to calculate velocities and required element size.
  - `lambs`
    - `mesh` Contains files related to the mesh
    - `data` Contains the files: parfile, source and stations
    - `ref` Contains reference seismograms and a README (13 Files)
- `src` Contains all fortran source files
- `tools`
  - `scripts` Contains a number of python scripts
- `LICENSE`
- `Makefile`
- `Readme.md`

For the remaining part of the manual the NEXD 2D installation directory will be referred to as `dg2d`.

Chapter 3

Input

There are three different parameter files for the basic version of the code. They contain all parameters necessary to run the program. These parameter files are stored in `yourSimulation/data/` and are called:

- parfile - general parameters
- source - parameters regarding the sources
- stations - parameters regarding the receivers

Additional parameter files are necessary to apply the special features of the code. If fractures are to be included in the simulation, two additional files are required:

- fracs - specifies the position of fractures
- interfaces - parameters regarding the fractures

If the code is used to perform a full waveform inversion, the following file is required:

- invpar - parameters regarding the inversion

If poroelasticity is to be considered during the simulation, no additional parameter files are reqiured. However, the requirements for files described in Section 3.2 need to be fullfilled. The contents of the files described above will be explained in detail in subsequent sections.

To run the simulation, a number of other files are required that are generated by the meshing program of the users choice. These files need to be stored in the directory `yourSimulation/mesh/` and are called

- absorb
- coord
- free
- mat
- matprop
- mesh

These files have a specific structure that will be explained for each file in a separate section. For the supplied examples they have been created using a python script called `cubit2dg2d.py` which uses the output (`*.cub`) from the commercial meshing software Trelis (formerly CUBIT). In addition, gmsh-scripts (`*.geo`) are available for many examples, too. They can be used to create meshs (`*.msh`) and these can be converted by the python script `gmsh2dg2d.py`.
If a different meshing software should be used, the user can change the script `gmsh2dg2d.py` to create the input for NEXD. This might be straightforward since this script uses the library meshio, that provides readers for many mesh formats.

General Advice Prior to discussing the contents of the parameter files some general advice is given to avoid mistakes (especially related to the input format). Parameters that appear as a certain type, e.g. integer, in the parameter files will be read in the same way. For integers that poses no issue. For floating point numbers (floats) there are a number of formatting options that will all be accepted. For example, 0.01 will be valid if entered in this way, but also if entered 1e-2. For boolean variables, either .false./.true. or F/T are valid. If any type error is detected an error message will be raised containing the affected parameter and the simulation will not run.

3.1 Files in `data`

The first section covers all parameter files that are located in `yourSimulation/data/`.

3.1.1 parfile

This general parameter file contains all general parameters necessary to configure the programs. Here, all relevant features can be enabled/disabled. For convenience specific sections of the parameter file will be discussed separately.

General Parameters

```
1  # Parameter file for 2D NDG software
2
3  # Title of the simulation
4  title                    = Example
5
6  # Choose the way in which the flux is calculated
7  fluxtype                 = 0                  # "0" Pre-calculated flux (only
       ↪ for elastic simulations), "1" existing methods
8
9  # Convenience Parameters
10 log                      = .true.             # ".true." if log should be
       ↪ displayed on screen, otherwise ".false."
11
12 # Number of Processors
13 nproc                    = 40                 # depending on setup
14
15 # Model parameters
16 externalfilename          = cold_flach2.grd    # File containing the external
       ↪ model
17 extvel                    = .false.            # ".true." if external velocity
       ↪ model should be loaded, otherwise ".false."
18
19 # Parameters regarding seismograms
```

Listing 3.1: General parameters

The section of the parfile displayed in listing 3.1 shows a number of general parameters.

- **title**: Currently "title" is just used to provide a name for the simulation in the output log files.
- **fluxtype**: This parameter switches between two ways to calculate the numerical fluxes. Option "0" selects the elastic fluxes calculated by Möller (2018) for the slip interface calculations. If these fluxes are selected and simultaneously attenuation is set to ".true.", a warning is raised. Option "1" selects the fluxes implemented by (Lambrecht, 2015). These fluxes do not support a calculation including fractures.

- **log**: Set ".true." if the log is to be created/displayed, ".false." otherwise.
- **nproc**: Defines the number of threads (processors) used in the simulation. The value is specific to the users system configuration. Single core calculations are currently not supported, thus a minimum of 2 should be entered.
- **extvel**: Select if an external velocity model is to be used. This works currently only for an elastic model.
- **externalfilename**: Set the file name of the external velocity model. Must contain the location of the file in relation to `yourSimulation`, or the absolute file path. This parameter is ignored as long as "extvel" is set to ".false.".

Parameters regarding seismograms

This part of the parfile is displayed in listing 3.2. There are no conflicts to other parameters.

```
20 subsampling_factor      = 1                 # reduce sampling rate of
     ↪ seismograms by this factor to create smaller but less accurate files
21 autoshift               = .false.           # Shifts the time-axis by the "
     ↪ width" of the used wavelet. For a Ricker or Gaussian the maximum will
     ↪  be at t=0. If set to .false. plott0 is used instead.
22 plott0                  = 3e-3              # Offset for the seismogram.
     ↪ Default is 0.
23 div                     = .false.           # ".true." if the radial
     ↪ component of the seimogram is to be calculated, otherwise ".false."
24 curl                    = .false.           # ".true." if the tangential
     ↪ component of the seimogram is to be calculated, otherwise ".false."
25
26 # Movie parameters
```

Listing 3.2: Parameters regarding seismograms.

- **subsampling_factor**: Can be used to reduce the sampling rate of the seismograms, where sampling_rate = subsampling_factor · dt.
- **autoshift**: Corrects for the "width" of the used wavelet. For a Ricker or Gaussian the maximum will be at t=0. If set to .false. plott0 is used instead.
- **autodt**: Offset for the seismogram.
- **div**: ".true." if the radial component of the seimogram is to be calculated, otherwise ".false.".
- **curl**: ".true." if the tangential component of the seimogram is to be calculated, otherwise ".false.".

Movie parameters

This part of the parfile is displayed in listing 3.3. These parameters influence the creation of binary files that are used by the program `movie` to generate "vtk"-files (Visualisation Toolkit).

- **movie**: This parameter enables or disables the creation of binary files that can be used to create vtk-files.
- **frame**: This parameter needs to be between 1 and the total number of time steps in the simulation. The parameter defines the number of time steps between individual snapshots for the output.
- **save_movie_trimsh**: Create files with average values for each element. These files take significantly less space but also have a poorer resolution of the wave field.

```
27 movie                    = .true.          # if ".true." movie files are
       ↪ created, otherwise ".false."
28 frame                    = 100             # Number of time steps for each
       ↪  frame of the movie
29 save_movie_trimesh       = .true.          # Create files with average in
       ↪ each element
30 save_movie_points        = .false.         # Create files with data for
       ↪ each point
31 save_movie_displacement  = .false.         # Plot displacement field
32 save_movie_velocity      = .true.          # Plot velocity field
33 save_movie_stress        = .false.         # Plot stress field
34 save_movie_p1            = .false.         # Plot pressure of the first
       ↪ fluid
35 save_movie_v1            = .false.         # Plot velocity of the first
       ↪ fluid
36 save_movie_p2            = .false.         # Plot pressure of the second
       ↪ fluid
37 save_movie_v2            = .false.         # Plot velocity of the second
       ↪ fluid
```

Listing 3.3: Movie parameters.

- **save_movie_points**: Create files with data for interpolation points within the mesh. *Warning:* These files take significantly more space and slow down the simulation but also show a much higher resolution of the wave field.
- **save_movie_displacement**: Enable to plot the displacement field.
- **save_movie_velocity**: Enable to plot the particle velocity field.
- **save_movie_stress**: Enable to plot the stress field.
- **save_movie_p1**: Enable to plot the pressure field of the first fluid (only for poroelasticity).
- **save_movie_v1**: Enable to plot the particle velocity field of the first fluid (only for poroelasticity).
- **save_movie_p2**: Enable to plot the pressure field of the second fluid (only for poroelasticity).
- **save_movie_v2**: Enable to plot the particle velocity field of the first fluid (only for poroelasticity).

Parameters regarding time integration

This part of the parfile is displayed in listing 3.4. These parameters influence the selection of the method that is used for time integration, the number of time steps and the selection of the time step.

- **timeint**: This parameter selects the method for time integration. Options are:
  1. Euler,
  2. total variation diminishing (TVD) second order Runge-Kutta: rk2 (TVD),
  3. third order TVD Runge-Kutta: rk3 (TVD),
  4. low-storage five-stage fourth order Runge-Kutta, rk4 (LSERK).

  Enter the number to select the appropriate method.
- **autont**: Can be used to automatically calculate the number of timesteps to reach the time specified in t_total.
- **nt**: Select the number of time steps for the simulation. Not used if autont is true.

```
40 timeint                  = 2                 # which timeintegration?  1:
   ↪ euler 2:rk2 (TVD) 3:rk3 (TVD) 4:rk4 (LSERK)
41 autont                   = .false.           # automatic calculation of
   ↪ number of timesteps based on dt and t_total (if .true. t_total will
   ↪ be used, if .false. nt will be used)
42 nt                       = 10000             # Number of timesteps
43 t_total                  = 1.53              # Total simulated time (t_total
   ↪  = nt*dt)
44 autodt                   = .true.            # automatic calculation of dt
45 dt                       = 0.0               # if autodt =.false. choose dt
   ↪ manually
46 cfl                      = 0.4               # cfl value for dt
47 simt0                    = 0.                # starting time of simulation
```

Listing 3.4: Parameters regarding time integration

- **t_total**: Select the total simulated time. Not used if autont is false.
- **autodt**: Automatically calculate the time step based on the Courant-Friedrichs-Lewy (CFL) criterion.
- **dt**: Specifies a dt different from the automatic one. This parameter is ignored as long as autodt is true.
- **cfl**: Parameter for the CFL stability criterion. Recommended values are: rk2, cfl = 0.4; rk4, cfl = 0.7.
- **simt0**: Specifies the starting time of the simulation.

Attention: The recommended time integration method is the second order Runge-Kutta method (rk2) with a cfl of 0.4.

Parameters regarding PML

This part of the parfile is displayed in listing 3.5. These parameters define the characteristics of Perfectly Matched Layers enhancing the absorbing boundary conditions.

```
50 set_pml                  = .false.           # if ".true." pml are set else
   ↪ absorbing boundary conditions are set
51 pml_delta                = 3.0               # pml thickness
52 pml_rc                   = 0.001             # pml reclection coff
53 pml_kmax                 = 1.0               # pml kmax
54 pml_afac                 = 1.0               # factor for amax
55 use_trigger              = .true.            # use sta_lta trigger for
   ↪ energy monitoring
56 avg_window1              = 10                # lta window
57 avg_window2              = 2                 # sta window
58 sta_lta_trigger          = 0.1               # threshold
```

Listing 3.5: Parameters regarding PML

- **set_pml**: Enable PML additionally to absorbing boundary conditions. This parameter should only be set to ".true." if the mesh has been designed with a PML layer (see section A for details).

- **pml_delta**: This parameter describes the thickness of the PML. A uniform thickness of the PML layer throughout the model is assumed. PML layers with non-uniform thickness are currently not supported.

The parameters below influence the dampening profile of the PML. It is advised to keep the suggested values.

- **pml_rc**: Theoretical reflection coefficient for the PML. Default is 0.01.
- **pml_kmax**: Maximum value for the variable k. Values may range between 1 and 20.
- **pml_afac**: Factor for α_{max}.

The following parameters are not directly related to the PML layer. They serve as a control mechanism to ensure that the PML are working properly. It is designed to detect instabilities inside the PML and disables the PML, if necessary. For most cases the parameters for the length of the trigger windows can be left to the default values.

- **use_trigger**: Parameter to enable the STA-LTA trigger.
- **avg_window1**: Size of the window to determine the long term average (LTA).
- **avg_window2**: Size of the window to determine the short term average (STA).
- **sta_lta_trigger**: Threshold for the STA-LTA trigger. Values above this threshold will lead to disabling the PML.

Parameters regarding attenuation

This part of the parfile is displayed in listing 3.6. If enabled, these parameters influence the calculation of the attenuation.

```
61 attenuation              = .false.           # ".true." to enable
   ↪ attenuation otherwise ".false."
62 f0_att                   = 400               # Frequency where the model
   ↪ parameters are applied
63 f_max_att                = 1200              # maximum of frequency band for
   ↪  anelastic modulus
64 att_factor               = 100.              # factor to define minimum
   ↪ frequency, f_min_att=f_max_att/att_factor
```

Listing 3.6: Parameters regarding attenuation

- **attenuation**: Enables the simulation of viscoelastic attenuation. This requires specific values to be set in the `yourSimulation/mesh/matprop` file (see section 3.2.3).
- **f0_att**: Attenuation is simulated by a number of Maxwell bodies (see Lambrecht (2015) for details) to introduce disperion. This parameter sets the frequency to which the model parameters are assigned to.
- **f_max_att**: Defines the upper end of the frequency range where the quality factors are held approximately constant.
- **att_factor**: Factor defining the lower end of the frequency range by f_min_att = f_max_att / att_factor.

```
67 lsi                    = .false.          # ".true." if the fracture
   ↪ influence is to be calculated, otherwise ".false."
68 normal                 = .true.           # Jump for P–Waves
69 tangential             = .true.           # Jump for SV–Waves
```

Listing 3.7: Parameters regarding fractures.

Fracture parameters

This part of the parfile is displayed in listing 3.7. These parameters influence if and how the influence of fractures is created. For a detailed description on the theory behind these calculations, the interested user is referred to Möller (2018).

- **lsi**: This parameter enables or disables the calculation of the influence of fractures. Enableling this feature requires the additional parameter files "fracs" and "interfaces". Currently this works not for calculation including attenuation.
- **normal**: Enables the calculation of the fracture influence normal to the fracture plane.
- **tangential**: Enables the calculation of the fracture influence parallel to the fracture plane.

Parameters regarding poroelasticity

This part of the parfile is displayed in listing 3.8. If this is enabeld, the materials are handled as poroelastic media. For a detailed description on the theory behind these calculations, the interested user is referred to Boxberg (2019).

```
72 poroelastic            = .false.          # Materials are poroelastic if
   ↪ ".true." otherwise elastic
73 fluidn                 = 1                # Number of immiscible fluids (
   ↪ either 1, i.e. saturated, or 2, i.e. unsaturated/saturated by 2
   ↪ fluids)
74 calculate_tortuosity   = .false.          # Tortuosity is calculated
   ↪ according to Berryman (1980): T = 1+r(1-1/phi) (note, that if this is
   ↪  set to ".true.", in the file porousmaterial r has to be specified
   ↪ instead of T!)
75 extmatprop             = .true.           # if ".true." the material file
   ↪  specified at 'external_material_name' will be used, else, matprop,
   ↪ created by e.g. 'Trelis' will be used.
76 extmatpropfilename     = mesh/matpropporo # name of external material
   ↪ file
```

Listing 3.8: Parameters regarding poroelasticity

- **poroelastic**: Enable poroelastic media. This requires a special choice of material properties (see section 3.2.3).
- **fluidn**: Number of immiscible fluids (1 or 2).
- **calculate_tortuosity**: If true, the material parameter tortuosity will be calculated according to $T = 1 + r\left(1 - \frac{1}{\varphi}\right)$, where φ is the porosity and r is a geometrical factor that is 0.5 for spheres (see Boxberg, 2019, for more information).
- **extmatprop**: If true, the material file specified at external_material_name will be used, else, matprop will be used.
- **extmatpropfilename**: Path and filename of the external material property file.

Adjoint inversion

This part of the parfile is displayed in listing 3.9. If this is enabeld, an iterative full waveform inversion using the adjoint method is performed. For a detailed description on the method please refer to Lamert (2020).

```
79 inversion                = .false.           # Perform inversion, ".false."
     ↪ = normal forward simulation
80 time_shift               = .false.           # Use higher time shift to
     ↪ prepare source-time-function for low-pass filtering
81 inv_steps                = 10                # Number of inversion steps
82 mask_radius              = 0.05              # Percentage of longest domain
     ↪ dimension to mask receivers and sources
83 lowfreq                  = 50                # Minimum low-pass-filter
     ↪ frequency used during inversion
84 highfreq                 = 500               # Maximum low-pass-filter
     ↪ frequency used during inversion
```

Listing 3.9: Parameters regarding full waveform inversion

- **inversion**: Enables an iterative full waveform inversion using the adjoint method when set to ".true.". This requires an additional parameter file (see section 3.1.6 and files with measured data (see section 3.3).
- **time_shift**: Overrides the time axis shift of "shift_sources" by a higher value to prepare the source-time-function for low-pass filtering. Can also be used for single forward simulations to prepare synthetic measurements. Please enter a suitable "lowfreq" in this case.
- **inv_steps**: Numer of iteration steps to be performed. If a sufficient misfit reduction is reached before this number of iterations is performed, the program can just be terminated.
- **maskradius**: Defines the radius of the mask around seismic sources, receivers and free surfaces where the values of the search direction are cut out. The entered value is the proportion of the longest dimension of the simulation domain which is used as radius.
- **lowfreq**: The lowest cutoff frequency which is used for low-pass filtering during the whole inversion. Important to obtain the shift for "time_shift".
- **highfreq**: The maximum cutoff frequency which is used for low-pass filtering during the whole inversion.

Global parameter for the sources

This part of the parfile is displayed in listing 3.10.

```
87 shift_sources            = .true.            # if ".true." then the sources
     ↪ will be shifted by 1.2/f0, otherwise the maximum of the wavelet is at
     ↪  t = simt0 + delay (see source parameter file).
```

Listing 3.10: Sources

- **shift_sources**: If true, the sources will be shifted by 1.2/f0, so that the maximum is at t = simt0 + delay + 1.2/f0, otherwise the maximum of the wavelet is at t = simt0 + delay (see section 3.1.2).

```
90 global_rec_angle        = .true.            # If ".true." the angle given
       ↪ below (rec_angle) will be used for all receivers, otherwise the angle
       ↪  has to be provided for each receiver indepentently in the receiver
       ↪ file.
91 rec_angle               = 0.0               # rotate receivers about degree
```

Listing 3.11: Receiver angle

Global parameter for the stations

This part of the parfile is displayed in listing 3.11.

- **rec_angle**: This parameter selects the angle by which **all** stations are rotated. For 0° the receiver points in the positive z-direction and the angle goes anti-clockwise, i.e., for 90° the receiver points in negative x-direction.

3.1.2 source

The example displayed in listing 3.12, shows a typical source parameter file containing one source.

```
 1 # List of Source(s)
 2 # Sources should be added sequentially. Otherwise problems will occur when
       ↪ reading the file
 3
 4 # Number of sources
 5 nsrc        = 1          # Total number of sources
 6
 7 # Source 1 - Parameters
 8 source      = 1          # Running Number of the sources
 9 xsource     = 25.        # x-value of the source
10 zsource     = 10.        # z-value of the source
11 delay       = 0.0        # time delay for the source
12 sourcetype  = 0          # Type of source: 1 = Moment tensor, 0 = single
       ↪ force
13 stf         = 2          # Type of source-time-function: 1 = gauss, 2 =
       ↪ ricker, 3 = sin^3, 4 = external
14 extwavelet  = data/1MHz0.5in_clean.txt  #file with external wavelet;
       ↪ required if stf = 4
15 f0          = 400.       # center frequency of stf
16 factor      = 1e6        # factor of the stf
17 angle_force = 00.        # angle of the force action in the media
18 Mxx         = 0.0        # Momenttensor Mxx
19 Mzz         = 0.0        # Momenttensor Mzz
20 Mxz         = 1.0        # Momenttensor Mxz
```

Listing 3.12: Typical source file

- **nsrc**: This parameter specifies the total number of sources in the model.
- **source**: Number of the current source. If more than one source is in the file, these number must increase sequentially.
- **xsource**: X-coordinate of the source. Needs to be inside the boundary of the model, but not inside the PML (if activated).

- **zsource**: Z-coordinate of the source. Needs to be inside the boundary of the model, but not inside the PML (if activated).
- **delay**: Parameter to specify a time-delay for the activation of the source.
- **sourcetype**: Select what type of source is used. Options are: "0" for a single force solution and "1" for a moment-tensor.
- **stf**: Select the source time function. Currently the following wavelets are available: "1" selects a Gaussian-function, "2" selects a Ricker-wavelet, "3" a cubed-sine function and "4" enables the user to input an arbitrary discrete external wavelet.
- **extwavelet**: Path and filename of an external wavelet. Is only used if option "4" is selected for "stf".
- **f0**: Sets the central frequency of the source wavelet .
- **factor**: Variable to scale the amplitude of the source time function.
- **angle_force**: This parameter is used if a single force solution for the source is used. It sets the direction of the force action. For 0° the source points in the positive z-direction and the angle goes anti-clockwise, i.e., for 90° the source points in negative x-direction.
- **Mxx, Mzz, Mxz**: Components of the moment-tensor, $M = \left(\begin{smallmatrix} M_{xx} & M_{xz} \\ M_{xz} & M_{zz} \end{smallmatrix}\right)$. Used only for a moment-tensor source (sourcetype = 1).

If an arbitrary defined wavelet is chosen by the user, the file containing the information on the wavelet will have to be designed as follows: Two columns separated by spaces. The first column contains the time and the second column contains the amplitude of the waveform.

When selecting the central frequency (f_0) of the source(s) the user has to keep in mind that this frequency directly influences the stability of the simulation, as the maximum length of an element directly depends on f_0. Assuming that 10 grid points per wavelength are sufficient to achieve stability, the maximum edge length, l, for a given frequency is calculated according to

$$l = \frac{v_{min}}{\frac{10}{N} * f_{max}},$$

where v_{min} is the slowest velocity and N is the polynomial order (default is $N = 4$). N can be changed in `src/constants.h`. Afterwards the program has to be compiled again. f_{max} is the maximum frequency selected for the source(s).

3.1.3 stations

This example shown in listing 3.13, shows a typical stations parameter file containing ten stations. If not specificly desired by the user, stations should be placed in model space that is not part of the PML (if enabled).

- **nrec**: This parameter specifies the total number of stations in the model.
- **No xrec zrec**: The first parameter is a running number of the receiver. The other two values represent the x- and z-coordinate of the individual station.

3.1.4 fracs

This is the first parameter file that is added as part of the feature to calculate the influence of fractures on the wave-field. This parameter file specifies the location of one or more fractures and is used if the global parameter "lsi" is enabled. A sample file is displayed in listing 3.14.
The first entry after "BEGIN" gives the total number of fractures to be read in.
Each fracture is defined by the following parameters (from left to right)

- **No**: Running number of the fracture.

```
nrec = 10
001  25.0  50.0
002  25.0  45.0
003  25.0  40.0
004  25.0  35.0
005  25.0  30.0
006  25.0  25.0
007  25.0  20.0
008  25.0  15.0
009  25.0  10.0
010  25.0  5.0
```

Listing 3.13: Typical stations file

```
# This input file defines the start and end points of crack in the medium.
# The first entry is the total number of cracks. Other values are:
# No. | Property index | Start x | Start y | End x | End y
BEGIN
1
001    1    0.0000    25.0    50.0    25.
END
```

Listing 3.14: Parameter file defining the location of fractures

- **Property index**: Indicates the type of interface that creates the fracture. The interface is selected from the selection given in "interfaces". For example: If there are 4 different slip interfaces given in "interfaces" this number may be either 1, 2, 3 or 4.
- **startx, startz**: x and z coordinates of the starting point of the fracture.
- **endx, endz** x and z coordinates of the end point of the fracture.

3.1.5 interfaces

This is the second parameter file that is added as part of the feature to calculate the influence of fractures on the wave-field. This parameter file specifies the properties that can be selected for a fracture via the property index in the previous file. A sample file is displayed in listing 3.15.

```
#--------------------------------------------------------------
#  Defines interfaces
#  First line: Number of interface types
#  Following lines: specification of interfaces from left to right
#  type (elastic [other interface-types are in development]),
#  nu_N. nu_T: Relaxationfrequencies of the interface part normal and
#  tangential to the fracture plan, respectively.
#--------------------------------------------------------------
BEGIN
1
elastic 100 100
END
```

Listing 3.15: Parameter file defining fracture properties

The first entry after "BEGIN" gives the total number of properties to be read in.

Each fracture property is defined by the following parameters (from left to right):

- **type**: currently only "elastic", viscoelastic interfaces are under development.
- $\nu_{\mathbf{N}}$: Relaxationfrequency of the part of the interface normal to the fracture plane.
- $\nu_{\mathbf{T}}$: Relaxationfrequency of the part of the interface tangential to the fracture plane. (see Möller, 2018, for details).

3.1.6 invpar

This is a parameter file that needs to be added for the full waveform inversion feature. It controls the inversion parameter that can be changed during the inversion. A sample file is displayed in listing 3.16.

```
1  upperfreq        = 50            ! cutoff frequency for low-pass filtering
2  inv_type         = 3             ! Method to calculate search direction, 1:
       ↪ SD, 2: CG, 3: BFGS
3  step_1           = 0.0           ! Three test step lengths at the start of
       ↪ the search for optimal step length
4  step_2           = 0.5
5  step_3           = 1.5
6  min_step         = 1.E-6         ! Minimum step length
7  max_step         = 1.E-1         ! Maximum step length
8  min_step_BFGS    = 1.E-4         ! Minimum step length for BFGS calculations
9  max_step_BFGS    = 10            ! Maximum step length for BFGS calculations
10 min_vp           = 100.          ! Minimum vp value that can be reached
       ↪ during inversion
11 max_vp           = 6000.         ! Maximum vp value that can be reached
       ↪ during inversion
12 min_vs           = 50.           ! Minimum vs value that can be reached
       ↪ during inversion
13 max_vs           = 3000.         ! Maximum vs value that can be reached
       ↪ during inversion
```

Listing 3.16: Parameter file defining the inversion parameters

- **upperfreq**: Defines the current cutoff frequency for the low-pass filter
- **inv_type**: Indicates the method to obtain the search direction based on the calculated gradients. 1: Steepest descent, 2: Conjugate gradient, 3: L-BFGS. In the first iteration or directly after increasing upperfreq always "inv_type" = 1 is automatically used
- **step_1,..,step_3**: Three test step lengths to obtain the misfit mimimum. For "inv_type" = 1,2 they represent the portion of the maxmimum wave velocity values of the current model. For "inv_type" = 3 they represent a factor for search direction obtained from the L-BFGS method. In the first iteration or directly after increasing upperfreq default values are used.
- **min_step, max_step**: Minimum and maximum test step lengths for "inv_type" = 1,2 that are allowed to prevent infinite loops.
- **min_step_BFGS, max_step_BFGS**: Minimal and maximal test step lengths for "inv_type" = 3 that are allowed to prevent infinite loops.
- **min_vp, max_vp, min_vs, max_vs**: Defines the minimum and maximum wave velocity values that are allowed in the model after a model change.

3.2 Files in mesh

This section explains the files related to the mesh. These files are designed in a certain way that is expected by NEXD. In general, any software can be used to create a mesh, but currently there are only scripts available that convert meshs created by Gmsh and Cubit/Trelis to a format that is understood by NEXD.

3.2.1 coord

```
1        3075
2           1          3.000000          50.000000
3           2          3.000000          25.000000
4           3          3.000000          49.000000
5           4          3.000000          48.000000
```

Listing 3.17: Excerpt from a coord file

This file contains a list that maps the number of a certain node in the mesh to its coordinates. The first entry is the total number of nodes in the mesh. All subsequent lines contain the running number of the nodes, the x-, and the z-coordinate.

Attention: Note that the coordinates referred to as z-coordinates are actually the y-coordinates in a Trelis/Cubit model. This is done with regard to the convention that in seismology the depth axis is always the z-axis. If Gmsh is used, please use the xz-plane.

3.2.2 absorb/free

```
1 151
2 513
3 517
4 518
5 519
```

Listing 3.18: Excerpt from a absorb/free file

These two files are set up identically. These files contain the number associated with the nodes that are used to calculate the simple absorbing BC or free surface BC. The first entry in each file is the total number of nodes contained in this file. Additional lines are added sequentially for each node. An example of the first few lines from such a file is shown in listing 3.18.

3.2.3 matprop

```
1 2
2 1 1 5800.0 3800.0 2600.0 9999.0 9999.0
3 2 1 5800.0 3800.0 2600.0 9999.0 9999.0
```

Listing 3.19: Entries for a matprop file with (visco-)elastic materials

The file in listing 3.19 lists the properties of the (visco-)elastic materials used in the model and the file in listing 3.20 lists the properties of the poroelastic materials. First the file scheme for (visco-)elastic simulations is presented:

The first entry lists the total number of materials. The subsequent lines contain the following values for (visco-)elastic materials (from left to right):

- Block number (Cubit block or physical material in Gmsh)
- Index identifying an elastic material (1).
- Compressional (P-) wave velocity (v_P).
- Shear (S-) wave velocity (v_S).
- Density (ρ).
- Quality factor regarding P-waves (Q_P).
- Quality factor regarding S-waves (Q_S).

The latter two values are related to viscoelastic calculations and influence attenuation. If attenuation is not selected in the parfile, these values will be irrelevant. If attenuation is selected and a quasi-elastic material is to be part of the simulation, these variables will need to have a high value ($Q \to \infty$ for an elastic material in reality, $Q = 9999$ for the simulation is sufficient).

Attention: This file needs to contain ar least two materials if PML are included in the mesh. PML are represented by an individual material within this file. See appendix A for more details

In the following, the scheme applying poroelastic materials is shown:

```
1 2
2 1 1 3000.0 13560000000.0 30720000000.0 0.3 1e-17 0.7 0.74 1.33e-07 0.099
    ↪ 0.5 2080000.0 0.0 0.055 0.5 40.0 0.0 2.86 0.79 0.0 0.0
3 2 1 2350.0 20000000000.0 15000000000.0 0.3 1e-13 0.7 0.74 1.33e-11 990.0
    ↪ 0.5 20800000000.0 5.4e-5 550.0 0.5 400000.0 2.0e-4 2.86 0.79 0.00
    ↪ 0.00
```

Listing 3.20: Entries for a matprop file with poroelastic materials

Also here, the first entry gives the total number of materials. The subsequent lines can be built by nine different models:

- Block number, 1, rhos, lambdaˆu, my, phi, kappa, b, 1/T, 1/N, rho1, S1, K1, ny1, rho2, S2, K2, ny2, fitting_n, fitting_chi, Sr1, Sr2
- Block number, 2, rhos, lambdaˆu, my, phi, kappa, b, 1/T, 1/N, rho1, S1, K1, ny1, rho2, S2, K2, ny2, p_b, lambda_BC, Sr1, Sr2
- Block number, 3, rhos, Kˆd, my, phi, kappa, 0., 1/T, Ks, rho1, S1, K1, ny1, rho2, S2, K2, ny2, fitting_n, fitting_chi, Sr1, Sr2
- Block number, 4, rhos, Kˆd, my, phi, kappa, 0., 1/T, Ks, rho1, S1, K1, ny1, rho2, S2, K2, ny2, p_b, lambda_BC, Sr1, Sr2
- Block number, 5, rhos, Kˆd, my, phi, kappa, 0., 1/T, 1/N, rho1, S1, K1, ny1, rho2, S2, K2, ny2, fitting_n, fitting_chi, Sr1, Sr2
- Block number, 6, rhos, Kˆd, my, phi, kappa, 0., 1/T, 1/N, rho1, S1, K1, ny1, rho2, S2, K2, ny2, p_b, lambda_BC, Sr1, Sr2
- Block number, 7, rhos, lambdaˆu, my, phi, kappa, b, 1/T, 1/N, rho1, S1, K1, ny1, rho2, S2, K2, ny2, A, 0., Sr1, Sr2
- Block number, 8, rhos, Kˆd, my, phi, kappa, 0., 1/T, Ks, rho1, S1, K1, ny1, rho2, S2, K2, ny2, A, 0., Sr1, Sr2

- Block number, 9, rhos, K^d, my, phi, kappa, 0., 1/T, 1/N, rho1, S1, K1, ny1, rho2, S2, K2, ny2, A, 0., Sr1, Sr2

The second entry of each line defines the used model. The different parameters are:

- Block number (Cubit block or physical material in Gmsh)
- rhos = density solid
- lambda^u = first Lamé parameter (undrained)
- K^d = bulk modulus of skeleton (drained)
- mu = shear modulus
- phi = porosity
- kappa = permeability
- b = biot coefficient
- 1/T = inverse tortuosity (Note, that this is replaced by r, if 1/T is calculated! See 3.1.1)
- 1/N = inverse of biot modulus
- Ks = bulk modulus of solid grain material
- rho1 = density fluid 1
- S1 = saturation fluid 1
- K1 = bulk modulus fluid 1
- ny1 = viscosity fluid 1 (this is the wetting fluid)
- rho2 = density fluid 2
- S2 = saturation fluid 2
- K2 = bulk modulus fluid 2
- ny2 = viscosity fluid 2 (this is the non-wetting fluid)
- fitting_n, fitting_chi = fitting parameters for van Genuchten model n, chi (see Boxberg, 2019)
- p_b = bubbling pressure
- lambda_BC = fitting parameter for Brooks & Corey model, see (Boxberg, 2019))
- A = capillary pressure coefficient for Douglas Jr. et al. model, see (Boxberg, 2019))
- Sr1 = residual saturation fluid 1
- Sr2 = residual saturation fluid 2

Instead of creating this material file, a matpropporo file can be specified (necessary for the Matlab code), which looks slightly different (see 3.21).

3.2.4 mat

This file maps the material properties from the matprop file to the elements. Each line consists of the running element number, the ID of the block (in the "matprop" file) and an index to tell if the element is part of a PML. The latter index is either "0" if it is part of the regular medium or "1", if it is part of a PML.

```
1  #-------------------------------------------------------------------------
2  # Defines porous materials:
3  #  First line: Number of material types
4  #  following lines: values of material parameters, the first number defines
       ↪  which model is used.
5  #  (rhos = density solid; lambda^u = first Lameparameter (undrained); K^d =
       ↪  bulk modulus of skeleton (drained); mu = shear modulus; phi =
       ↪ porosity;
6  #   kappa = permeability; b = biot coefficient; 1/T = inverse tortuosity (
       ↪ note, that this is replaced by r, if 1/T is calculated!);
7  #   1/N = inverse of biot modulus; Ks = bulk modulus of solid grain
       ↪ material;
8  #   rho1 = density fluid 1; S1 = saturation fluid 1; K1 = bulk modulus
       ↪ fluid 1; ny1 = viscosity fluid 1; (this is the wetting fluid)
9  #   rho2 = density fluid 2; S2 = saturation fluid 2; K2 = bulk modulus
       ↪ fluid 2; ny2 = viscosity fluid 2; (this is the non-wetting fluid)
10 #   fitting_n, fitting_chi = fitting parameters for van Genuchten model n,
       ↪ chi;
11 #   p_b = bubbling pressure; lambda_BC = fitting parameter for Brooks &
       ↪ Corey (1964) model);
12 #   A = capillary pressure coefficient for Douglas Jr. et al. (1993) model)
       ↪ ;
13 #   Sr1 = residual saturation fluid 1; Sr2 = residual saturation fluid 2:
14 # 1, rhos, lambda^u, my, phi, kappa, b,  1/T, 1/N, rho1, S1, K1, ny1, rho2,
       ↪  S2, K2, ny2, fitting_n, fitting_chi, Sr1, Sr2
15 # 2, rhos, lambda^u, my, phi, kappa, b,  1/T, 1/N, rho1, S1, K1, ny1, rho2,
       ↪  S2, K2, ny2, p_b,       lambda_BC,   Sr1, Sr2
16 # 3, rhos, K^d,      my, phi, kappa, 0., 1/T, Ks,  rho1, S1, K1, ny1, rho2,
       ↪  S2, K2, ny2, fitting_n, fitting_chi, Sr1, Sr2
17 # 4, rhos, K^d,      my, phi, kappa, 0., 1/T, Ks,  rho1, S1, K1, ny1, rho2,
       ↪  S2, K2, ny2, p_b,       lambda_BC,   Sr1, Sr2
18 # 5, rhos, K^d,      my, phi, kappa, 0., 1/T, 1/N, rho1, S1, K1, ny1, rho2,
       ↪  S2, K2, ny2, fitting_n, fitting_chi, Sr1, Sr2
19 # 6, rhos, K^d,      my, phi, kappa, 0., 1/T, 1/N, rho1, S1, K1, ny1, rho2,
       ↪  S2, K2, ny2, p_b,       lambda_BC,   Sr1, Sr2
20 # 7, rhos, lambda^u, my, phi, kappa, b,  1/T, 1/N, rho1, S1, K1, ny1, rho2,
       ↪  S2, K2, ny2, A,         0.,          Sr1, Sr2
21 # 8, rhos, K^d,      my, phi, kappa, 0., 1/T, Ks,  rho1, S1, K1, ny1, rho2,
       ↪  S2, K2, ny2, A,         0.,          Sr1, Sr2
22 # 9, rhos, K^d,      my, phi, kappa, 0., 1/T, 1/N, rho1, S1, K1, ny1, rho2,
       ↪  S2, K2, ny2, A,         0.,          Sr1, Sr2
23 #-------------------------------------------------------------------------
24 BEGIN
25 2
26 1 3000.0   13.56e9   30.720e9   0.3    1.0e-17   0.7    0.74    1.33e-7   0.099
       ↪   0.5    20.8e5    0.0       0.055    0.5    4.0e1    0.0       2.86    0.79
       ↪  0.00   0.00
27 1 2350.0    20.0e9    15.0e9    0.3    1.0e-13    0.7    0.74    1.33e-11    990.0
       ↪   0.5    20.8e9    5.4e-5    550.0    0.5    4.0e5    2.0e-4    2.86    0.79
       ↪  0.00   0.00
28 END
```

Listing 3.21: Structure of a matpropporo file

```
1          1          2          1
2          2          2          1
3          3          2          1
4          4          2          1
5          5          2          1
```

Listing 3.22: Excerpt from the mat file

```
1 5948
2          1          1         28          3
3          2         56        106         30
4          3          3         28         57
5          4        109         59        108
```

Listing 3.23: Excerpt from the mesh file

3.2.5 mesh

This file contains a mapping that relates the nodes and the material properties to the individual element. The first entry is the total number of elements in the mesh. Afterwards, each line contains the following entries (from left to right):

- Running number of the element
- The numbers of the three nodes that are used to construct the element.

A few lines from a mesh file are shown in code listing 3.23.

3.3 Files in `inversion`

Files in `inversion` are only necessary when performing a full waveform inversion. The measured data which shall be reconstructed during the inversion are saved here as displacement seismograms. For the current version of the code the measured data need to be complete. That means for every source a measured seismogram must exist for every station and every component.

The filename of the measurements is similar to the output files of the solver (see section 4.2):

seismo.(component).(number of the station).sdu.(number of the source).meas

The component is either “x” or “z”. The number of the station is a seven-digit number (e.g. 0000001) associated to the number of the station in the `station` file. The number of the source is a three-digit number (e.g. 001) corresponding to the number of the source in the `source` file.
Please make sure that the time axis of the measurements completely covers the time axis of the simulation. To obtain synthetic measurements from a forward simulation, a Python script `copy_seis.py` is provided in the folder `tools`. It copies the seismograms from the folder `out` into the folder `inversion` and adjusts the file name by the source number which needs to be entered in the Python script.

Chapter 4

Output

By default, the output of NEXD 2D is stored in a subdirectory to the simulation directory called `out`. If the script "process.sh" is used, the parameter files used for the current simulation are copied to the `out` folder for the users convenience. That way, the user is always able to reconstruct what parameters were used to create this simulation. If a full waveform inversion is applied, output files are also saved in `inversion` and `adjoint`.

4.1 Files created by `mesher`

For every simulation, files containing information of parts of the mesh called "meshVar0000001" are created. The number denotes the running number of the processor the file was created for.

If stations are placed in the model, files are created that contain the information on the location of the station, where it is located in the mesh and in which processor the receiver is located. These files are called for example "recVar000013".

If fractures are included, files that map the slip interfaces to their elements called for example "elementToLSI000032" will be created as well.

A number of visualisation files are created. They are listed in table 4.1.

Table 4.1: List of vtk-files given as output.

Name	Content
mesh.vtk	triangular mesh
rec.vtk	receiver positions
src.vtk	source positions
rho_model.vtk	density model
vp_model.vtk	p-wave velocity model (if not a poroelastic simulation)
vs_model.vtk	s-wave velocity model (if not a poroelastic simulation)
vmax_model.vtk	maximum wave velocity model (if poroelastic simulation)
vmin_model.vtk	minimum wave velocity model (if poroelastic simulation)

4.2 Files created by `solver`

The output from `solver` varies with the parameters selected in the parameter files. Seismograms will be created, if stations are placed in the model and the correct parameters have been set. Binary files to generate visualisations of the wave-field are only generated if the appropriate parameters are set in the "movie" portion of the parameter file (see section 3.1.1).

4.2.1 Seismograms

Seismograms are saved as plain ASCII text files with two columns of data: The first column contains the time-axis and the second column contains the respective data. Three types of data are available: displacement, particle velocity and acceleration. Each type has its own file distinguished by the file name. The file-name is constructed in the following way:

seismo.(component).(number of the station).sd(type)

The component is "x", "z" or "p" if it is pressure data. If the parameter div and/or curl are enabled this may also be "r" or "t" respectively, where "r" is the radial and "t" the tangential component. The number of the station is a seven-digit running number starting from 0000001 to the total number of stations (cf., nrec in sec. 3.1.3) placed in the model. The file extension .sd* tells the user which kind of data is contained in the file: "a" for acceleration, "v" for (solid) particle velocity, "v1" and "v2" for fluid particle velocity, "p1" and "p2" for fluid pressure, and "u" for displacement.

Example: Given these conventions, a seismogram for the x-component of station 23 containing displacement data is named: *seismo.x.0000023.sdu*.

4.2.2 Binary file to create vtk files

The exact amount and type of binary files created by `solver` depends on the choice of parameters. First of all, the parameter "movie" needs to be enabled so that any file may be created. If that is the case, a file is created for each time-step matching the "frame" parameter per processor.

Example: If frame = 100, a file will be created for each 100th time-step.

Additionally, the user can enable the creation of binary files for acceleration, particle velocity and/or displacement. If either is selected, files will be created that contain the x-component, the z-component and the norm of the selected property, respectively. The files are named in the following way:

moviedata.(type)(component).(number of the processor)_it(time-step).bin

Here, type is "a", "v", "v1", "v2", "u", "p1", "p2", or "stress", the component is "x", "z" or "norm". In case of the norm, type and component are reversed. The number of processor is a six-digit number of the processor that created the file and time-step is a seven-digit number that describes the time-step that the file represents.

Example: Given these conventions, a movie-binary file for the x-component of the particle velocity created in processor 23 at time-step 4000 is named: *moviedata_vx_000023_it0004000.bin*. If case of the norm, but otherwise identical parameters the file is named: *moviedata_normV_000023_it0004000.bin*.

4.2.3 Output during an inversion

During an inversion process a lot of data is produced. After each single forward or adjoint simulation the standard output described above is saved in folder `out`. Additionally, the program `movie` (see section 4.3) is called automatically. All these files will be overwritten by the next simulation since they are of minor interest for the inversion process. However, it is possible to check them during the inversion.

Output in `adjoint`

Here, binary files (*.bin files) to produce vtk files for the misfit gradient similar to the movie data (see section 4.2.2) are saved. A vtk file of the misfit gradient for each iteration step is automatically produced after the iteration step for each single source. The file name of the misfit gradient vtk files are named as follows:

movie_element_K(type)_src(number of source)_it(iteration step).vtk

Type is either "vp" or "vs" describing P- or S-wave velocity, respectively. The number of source is a three-digit number associated with the source number in the `source` file and iteration step is a seven-digit number describing simply the iteration step in which the gradient was used.
Additionally to these files the source time function of all adjoint sources are saved. Their names are built as follows:

adj_ stf(number of station)iter(iteration step)src(number of source)

The number of the station is a seven-digit number corrensponding to the station number in the `station` file, the iteration number is simply the running number of the iterations of the inversion procedure, and the number of the source corresponds to the source number in the `source` file.

Output in `inversion`

In this directory the four files:

- alpha
- frequency
- misfit
- time

can be found. All four files show the iteration step in the first column and in the second column they show the used step width (alpha), the cutoff frequency of the applied low-pass filter (frequency), the misfit corresponding to the model used at the beginning of the iteration step (misfit), and the total computational time (time).

Additionally, here are binary files (*.bin files) saved that are used to create vtk files for the search direction and the velocity model after the model change for every iteration step. They are named as follows:

movie_ element_ cvp_ it(iteration step).vtk
movie_ element_ cvs_ it(iteration step).vtk
movie_ element_ model_ vp_ it(iteration step).vtk
movie_ element_ model_ vs_ it(iteration step).vtk

The first two show the search direction for the P- and S-wave velocity, respectively, and the last two show the current velocity model after the model change of the specific iteration step is applied. The variable iteration step is a seven-digit number of the running iteration step during the inversion procedure.
Seismograms for each station and source are also saved here. They have the structure:

seismo.(component).(station number).sdu(iteration number)run1src(source number)

The component is either "x" or "z". The station number is a seven-digit number corresponding the station number in `station`, the iteration step is a three-digit number of the running iteration step during the inversion procedure, and the source number corresponds to the source number in the `source` file.
For every measurement file (see section 3.3) files with an appendix "filter(frequency)" are created. They contain the filtered measurements while frequency in the appendix is a four-digit number for the cutoff frequency of the low-pass filter in Hertz.

Temporal wave field files

During the inversion process large files containing the wave field of the forward simulation need to be saved to disk. These files take a lot of storage capacity and a low disk write rate can slow down the simulation strongly. By default, these files are saved in the `inversion` folder. If your local `inversion` folder is not able to handle such files, please change the variable "temppath" in the file `src/constants.h` to a suitable directory and recompile the complete code. Please have in mind that each processor will use this directory from its own point of view. If your simulation is distributed over processors of different computers on a large cluster every processor will call the directory from the computer it is running on.

4.3 Files created by `movie`

There are two possible outputs:

- trimesh: Creates file, where the average over an element is plotted.
- point: Creates files, which contain the information on each grid point.

Both variants can be used independently and together.

4.3.1 trimesh files

This version of the wave filed output is enabled by selecting the parameter "save_movie_trimesh". The files are named in the following way:

movie_element_(type)(component)_it(framenumber).vtk

The part of the filename containing type and component follows the same pattern as above. The framenumber is a seven-digit number that increases sequentially.

Example: Given these conventions, the 4^{th} movie-vtk for the x-component of the particle velocity is called: *movie_element_vx_it0000004.vtk*. In case of the norm, but otherwise identical parameters the file is called: *movie_element_norm_v_it0000004.vtk*.

4.3.2 points files

This version of the wave-field output takes significantly more computational time as well as space on the hard-disk, but contains more detailed information. It is enabled by selecting the parameter "save_movie_points". The files are named in the following way:

movie_points_(type)(component)_it(framenumber).vtk

The part of the filename containing type and component follows the same pattern as above. The framenumber is a seven-digit number that increases sequentially.

Example: Given these conventions, the 4^{th} movie-vtk for the x-component of the particle velocity is named: *movie_points_vx_it0000004.vtk*. In case of the norm, but otherwise identical parameters the file is named: *movie_points_norm_v_it0000004.vtk*.

Chapter 5

Simulation

This chapter explains a complete simulation and suggests an efficient order of tasks, including pre- and post-processing. To follow this chapter, a successful installation of NEXD and all its requirements is necessary (see chapter 2). It is assumed that the code has been compiled without errors. To run a simulation, follow these steps:

- Prepare your simulation (see section 5.1 below for more details).
- Prepare your model, either using Trelis and the supplied routines or a meshing software of your choice.
- Check if all parameter files are present (see chapter 3 for more details).
- Run the preprocessor called `mesher` by executing the shell-script "process.sh". The present working directory of the console should be your simulation directory, e.g., `simulations/lambs`.
- Run the solver (see section 5.2 for details).
- If a visualisation apart from the seismograms is required, run `movie` by executing `./bin/movie` (see chapter 4 for details).

5.1 Preparation

After a successful installation, the directory for the first simulation has to be prepared. Follow these steps to prepare the model and parameter files:

1. Create a simulation folder according to the structure displayed in the examples and provide the relevant input files.
2. Create the model and subsequent input files for `mesher` as described in appendix A.
3. Change the appropriate files to individual specifications (see chapter 3 for details).
4. Either copy the shell script "process.sh" to the new simulation directory and run it or follow the next steps.
5. Create the folders "bin" and "out" in the simulation directory. If you are doing an inversion create the folders "adjoint" and "inversion" as well.
6. Copy the executable files `mesher`, `solver` and `movie` from `dg2d/bin/` to the `yourSimulation/bin` directory in the simulation directory or create a symbolic link to these executable files.
7. run `mesher` by executing the command "./bin/mesher" in the console.

Recommendation: If "process.sh" is not used, it is advised that the parameter files from the "data" directory are copied to the "out" folder prior to running `mesher` (between steps 6 and 7). That way all information on how the simulation was created is stored with the output.

5.2 Calculation

This section is highly specialised for the individual user. Assuming that the user is able to run the program directly on his/her computer and OpenMPI is installed on the system there is only thing to do:

- run `solver` by executing `mpirun -np (nproc) bin/solver`.

The parameter "nproc" gives the number of processors as defined in the parfile. This number *must* be the same.
If desired, information on the process of the simulation are printed on screen. For select time steps the output listed in code listing 5.1 is generated. These time steps are defined by the "frame" parameter from the movie-parameter section. Each time a frame for the movie is generated, the output is printed into the file. It displays the time needed for the current interval, the estimated remaining time, the estimated total time and the mean elapsed time per time step. In addition an estimation is given on what date and time the simulation will be finished. This is particularly helpful for longer runs.

```
|-------------------------------------------------------------------------|
|      Time step number     100 (t = 6.12E-04 s) out of   10000           |
|      Elapsed time:                    0 h 00 m 11 s                     |
|      Estimated remaining time:        0 h 19 m 06 s                     |
|      Estimated total time:            0 h 19 m 18 s                     |
|      Mean elapsed time per timestep:    1.16E-01 s                      |
|      Current maximum norm of U    : 7.92007526E-08                      |
|      Current maximum norm of V    : 5.47094794E-04                      |
|      This run will finish on: Tue Aug 07, 2018 at 11:50                 |
|                    1 % of the simulation completed.                     |
|-------------------------------------------------------------------------|
```

Listing 5.1: Output for select time steps

If you are doing an inversion the output above will not be shown. Since a high number of simulations is performed during an inversion, the output shows which simulation simulation is performed at the moment at which iteration step. Additionally, the calculation time needed for each simulation or intermediate step is presented. An example is shown in code listing 5.2.

```
|-------------------------------------------------------------------------|
|                            starting timeloop...                         |
|                    Iteration number:            3                       |
|                             Source:      2 of  2                       |
|                          Run number:            2                       |
|                     Simulation type:      forward                       |
|                           Time step:  4.88473E-05                       |
|                Number of time steps:         6000                       |
|                Total simulated time:  2.93084E-01                       |
|-------------------------------------------------------------------------|
| Finished first test forward simulation for source   2                   |
| Calculation time   235.3 seconds, total computation time: 1.5 hours     |
|                                                                         |
| Change model and start second test forward simulation                   |
| Smallest vp/vs ratio: 1.41                                              |
| New time step:  4.88468E-05, with highest velocity: 1125.531            |
|-------------------------------------------------------------------------|
```

Listing 5.2: Output for an inversion procedure

The current iteration number is presented as well as the currently used source. The "Run number" describes which test step length is currently used (1,2, or 3) and "Simulation type" is either "forward" or

“adjoint”. “Time step” is the time step currently used for this simulation and “Number of time steps” the total number of time steps to be simulated. “Total simulated time” describes the simulated time span of wave propagation in seconds. Below that informatin on the computational time for each simulation and the total running time is presented. At the bottom information on an intermediate step betwenn two simulations is shown.

5.3 Evaluation

To view the seismograms, a program like GNUPLOT is sufficient. Python also offers an extended library on plotting such data.
If created, the vtk files may be viewed by a program like ParaView. According to their website “ParaView is an open-source, multi-platform data analysis and visualization application.”. It can be freely downloaded from the website `https://www.paraview.org/`. Otherwise a library called “vtk” exits for Python that contains functions to process and display vtk files. It is part of the Anaconda package but may be installed separately by the user.
For the included examples, reference seismograms are provided. If one of the examples is run, they are a basis for comparison.

Chapter 6

Contact and Support

For questions, feedback, or suggestions please contact us on `http://github.com/seismology-RUB` or via email (`seismology-coder@ruhr-uni-bochum.de`).

Chapter 7

License

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

   The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such

as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

(a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

(b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

(c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

(d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

(a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

(b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

(c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

(d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

(e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status

of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

(a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

(b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

(c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

(d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

(e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

(f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient,

for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

   You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim

or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

    The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

    If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

    Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

End of Terms and Conditions

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <textyear>  <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program>  Copyright (C) <year>  <name of author>

This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `https://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `https://www.gnu.org/licenses/why-not-lgpl.html`.

Bibliography

Boxberg, M. S. (2019). *Simulation of Seismic Wave Propagation in Porous Rocks Regarding the Exploration and the Monitoring of Geological Reservoirs.* PhD thesis, Ruhr-Universität Bochum.

Boxberg, M. S., Heuel, J., and Friederich, W. (2017). A nodal discontinuous Galerkin solver for modeling seismic wave propagation in porous media. In *Poromechanics VI*, pages 1490–1498.

Lambrecht, L. (2015). *Forward and inverse modeling of seismic waves for reconnaissance in mechanized tunneling.* PhD thesis, Ruhr-Universität Bochum.

Lambrecht, L., Lamert, A., Friederich, W., Möller, T., and Boxberg, M. S. (2018). A nodal discontinuous Galerkin approach to 3-D viscoelastic wave propagation in complex geological media. *Geophysical Journal International*, 212(3):1570–1587.

Lamert, A. (2020). *Tunnel reconnaissance by seismic full waveform inversion.* PhD thesis, Ruhr-Universität Bochum.

Lamert, A. and Friederich, W. (2019). Full waveform inversion for advance exploration of ground properties in mechanized tunneling. *International Journal of Civil Engineering*, 17(1):19–32.

Möller, T. (2018). *Numerical modelling of seismic wave propagation in fractured media for the characterization of geothermal reservoirs.* PhD thesis, Ruhr-Universität Bochum.

Möller, T. and Friederich, W. (2019). Simulation of elastic wave propagation across fractures using a nodal discontinuous Galerkin method—theory, implementation and validation. *Geophysical Journal International*, 219(3):1900–1914.

Appendix A

Creating a model (with Trelis)

This chapter explains very briefly how to generate a model for NEXD 2D using the Trelis software and the supplied journal (*.jou) files. Detail on how to use Trelis will not be given. If problems arise, the user is referred to the user manual of Trelis `https://www.csimsoft.com/help/trelishelp.htm`. This manual refers to Trelis 16.5. The models presented in the example were created and tested with Trelis version 15.2. A user manual of this version can be downloaded at `https://www.csimsoft.com/download?file=Documents/Trelis15UserDocumentation.pdf`.

A.1 The journal file

The journal file is a script that can be executed by Trelis to generate a pre-defined model. If the meshing process is to be done often (e.g., due to mesh refinement) it is advisable to create these files. By default, Trelis creates journal files for each run of the meshing process. If the desired model is created, it is advised to save that information in a new file related to the model.
The reset command at the beginning resets the current environment to its default condition. It is highly recommended to use this command ahead of each run. Lines 5 - 14 list the commands that are used to create the model (the geometric shape).

Warning: If the model is created using vertex points (like this example) the user has to pay attention on how the surfaces are created. The normal of all surfaces must point in the same outward direction. Otherwise an error is raised n the when "mesher" is run called ***Jacobian*** $<$ ***0***. The easiest way avoid this error, is to create the surface by calling the points anticlockwise according to their coordinates (see lines 13 -14).

Lines 17 - 19 are optional and are only needed if PML are to be used. They create the PML layer by cutting it from the original volume.

Attention: If PMLs are to be included in the model, they have to be of uniform thickness. Nonuniform thickness of PMLs is currently not supported and creation of such will result in errors.

Lines 26 - 30 describe the meshing process. Usually the only portion to be adjusted here is the size of the elements. This is done in line 28 in this example. The unit of this number is meters.
Next, the materials are defined. In Lines 37 and 38 each block is assigned a number of surfaces. They are assigned by number which are given to the surfaces by Trelis. In lines 41/42 the material properties are assigned to the blocks. The properties are assigned via the blocks name and consist of the following values: v_p, v_s, ρ, Q_p, Q_s, and PML index. v_p, v_s and ρ are given in SI units. The Q-factors relate to attenuation and the last index signifies whether that block is part of the PML or not. Note, the ID for the PML changes for block 2. This indicates that all surfaces contained in this block are part of the PML.
Similar to the blocks, nodesets are created that contain the nodes on the surfaces. These nodesets are used to assign boundary conditions. Each nodeset is assigned the nodes from a number of surfaces. They my be also left empty. If, for example, no free surface is used, the nodeset can be left empty. However, it needs to be present in the file otherwise the python script will not function properly. Each nodeset

```
reset

##########################################

#model
create vertex 0 0 0
create vertex 50 0 0
create vertex 50 25 0
create vertex 50 50 0
create vertex 0 50 0
create vertex 0 25 0

create surface vertex 1 2 3 6
create surface vertex 6 3 4 5

##########################################
#pml
webcut volume all with plane normal to curve 1 distance 3 from vertex 1
webcut volume all with plane normal to curve 4 distance 3 from vertex 1
webcut volume all with plane normal to curve 10 distance 3 from vertex 2

##########################################

#meshing

imprint all
merge all
surf all size 1.0
surface all scheme TriDelaunay
mesh surface all

##########################################
# materials
# allows for elements to be part of multiple blocks
set duplicate block elements on

block 1 tri in surf 14 16
block 2 tri in surf 6 9 10 11 12 13 15

# name nr vp vs rho qp qs pml
block 1 name "elastic 1 5800 3800 2600 9999 9999 0"
block 2 name "elastic 2 5800 3800 2600 9999 9999 1"

##########################################
#edges
nodeset 1 node in curve 19 41 43
nodeset 2 node in curve 5 6 7 8 14 23 25 28 32 34 38

nodeset 1 name "free_surf"
nodeset 2 name "absorb"

save as "testtri.cub" overwrite
```

Listing A.1: Excerpt from the mat file

is assigned a name. The user is asked to use the default names, as a modification of that name would require a modification of the python script, that prepares the model for NEXD 2D.
Finally, the model is saved as a .cub file. This file will be used to create the parameter files described in section 3.2.

A.2 Prepare the input files

The relevant input files are created by running the script "run_cubit.py". This scripts loads all necessary modules and calls the script "cubit2dg2d.py". The script requires the file "cubit2dg2d.py" to be present in the folder it is executed from, or line 12 (reload command) needs to be modified with the correct path to the file.

```
1  #!python
2  #!/usr/bin/env python
3
4  import cubit
5  import cubit2dg2d
6  import os
7  import sys
8  from math import *
9  from numpy import *
10
11
12 reload(cubit2dg2d)
13
14 cubit.init([""])
15 cubit.cmd('set duplicate block elements on')
16 command = 'open "testtri.cub"'
17 cubit.cmd(command)
18
19 cubit2dg2d.mesh()
```

Listing A.2: Python script to create input file from cubit model

Modify line 16 by replacing the name "testtri.cub" by the name that has been assigned to the model (name of the .cub file).

A.3 Using a different meshing tool

Currently no other meshing tool is supported by the developers. If a user is not able to use the supported meshing software he/she is asked to create the relevant input files from the output of his/her own meshing software.

Appendix B

Adding a custom source time function

It may become necessary for certain users to add a source time function of their own. This chapter describes, which section of the source code and what entries in the parameter files need to be changed.

B.1 Source code

There are a number of portions of the source code that need to be changed. It is not simply done by adding the function itself, but also modifying the related error messages, so the new function is not rejected. First, the new source time function needs to be added as an additional function in the module "stfMod.f90". It is recommended that the basic structure of the function is similar to the already existing ones. As a basic example the function describing a Ricker-wavelet is shown in code listing B.1

```
1 (sftMod)
2 function stfRicker(t,f0,t0,factor)
3     implicit none
4     real(kind=custom_real) :: stfRicker
5     real(kind=custom_real) :: f0,t0,factor,t
6     real(kind=custom_real) :: aval
7     aval = pi*pi*f0*f0
8     stfRicker =  factor * (1.-2.*aval*(t-t0)**2.) * exp(-aval*(t-t0)**2.)
9 end function stfRicker
```

Listing B.1: Code for the Ricker wavelet.

As described in the section 3.1.2, each source time function is called by a specific number. If an additional function is added, that number increases. There are a number of sections in the code where that number influences the calculations. The first occasion is in the subroutine *initSource* in the module *sourceReceiverMod*. The relevant code is displayed in code listing B.2. Edit the call to the if-condition (line 114) to accept larger values. Edit the error-message (line 116) accordingly.
The next occasions are in the module *timeloopMod*. Here, the source code has to be changed at two locations. First between lines 498 and 548 (see code listing B.3). A new statement to the if call starting at line 503 needs to be added in the same way as the previous ones so to make the new stf available for further calculations. The number in the last segment (line 518) needs to be increased as well.
The final piece of code that needs editing is between lines 671 and 691 in *timeloopMod* (code listing B.4. In much the same way as in the previous code segment of *timeloopMod*, the if statement starting at line 672 needs to be expanded to include the new function in the same way as it is designed for the included source time functions. At this point, no error message has to be updated.

```
113 call readIntPar(this%srcstf(isrc), "stf", filename, pos, errmsg)
114 if( this%srcstf(isrc) < 1 .or. this%srcstf(isrc) > 4) then
115     !this message needs to be adjusted if new wavelets are added to the
        ↪ Program
116     call add(errmsg, 2, "Parameter to select the source time function is
        ↪ out of range. Select either 1, 2, 3 or 4.", myname, filename)
117 end if
118 if (this%srcstf(isrc) == 4) then
119     call readStringPar(this%extwavelet(isrc), "extwavelet", filename, pos,
        ↪ errmsg)
120     inquire(file=trim(this%extwavelet(isrc)), exist=file_exists)
121     if (.not. file_exists) then
122         call add(errmsg, 2, "File does not exist!", myname, filename)
123     end if
124 end if
```

Listing B.2: Code snippet from *initSource*.

```
498 ! choose source time function
499 if (mesh%has_src) then
500     do i=1,src%nsrc
501         do it=1,par%nt
502             time = (float(it)-1.)*dt
503             if (src%srcstf(i) == 1) then !GAUSS
504                 plotstf(it,1,i) = time
505                 plotDiffstf(it,1,i) = time
506                 plotstf(it,2,i) = -stfGauss(time,src%srcf0(i),t0(i)+src%
                        ↪ delay(i),src%srcfactor(i))
507                 plotDiffstf(it,2,i) = -stfDiffGauss(time,src%srcf0(i),t0(i)
                        ↪ +src%delay(i),src%srcfactor(i))
508             else if (src%srcstf(i) == 2) then !RICKER
509                 plotstf(it,1,i) = time
510                 plotDiffstf(it,1,i) = time
511                 plotstf(it,2,i) = -stfRicker(time,src%srcf0(i),t0(i)+src%
                        ↪ delay(i),src%srcfactor(i))
512                 plotDiffstf(it,2,i) = -stfDiffRicker(time,src%srcf0(i),t0(i
                        ↪ )+src%delay(i),src%srcfactor(i))
513             else if (src%srcstf(i) == 3) then !SIN^3
514                 plotstf(it,1,i) = time
515                 plotDiffstf(it,1,i) = time
516                 plotstf(it,2,i) = -stfSin3(time,src%srcf0(i),t0(i)+src%
                        ↪ delay(i),src%srcfactor(i))
517                 plotDiffstf(it,2,i) = -stfDiffSin3(time,src%srcf0(i),t0(i)+
                        ↪ src%delay(i),src%srcfactor(i))
518             else if (src%srcstf(i) > 4) then
519                 call add(errmsg, 2, "Chose a valid source time function.
                        ↪ Available functions are listed in the 'source'
                        ↪ parameter file.", myname, "data/source")
520                 call print(errmsg)
521                 call stop_mpi()
522             end if
523         end do
524         if (src%srcstf(i) == 4) then
525             filename=src%extwavelet(i)
526             call stfExternal(plotstf(:,2,i),plotstf(:,1,i),dt,par%nt,.true
                    ↪ .,6,3,trim(filename),0, errmsg)
527             call stfExternal(plotDiffstf(:,2,i),plotDiffstf(:,1,i),dt,par%
                    ↪ nt,.true.,6,3,trim(filename),1, errmsg)
528         end if
529         [...]
530     end do
531 end if
```

Listing B.3: Code snippet from *timeloopMod* (1).

```
671 do i=1, src%nsrc
672     if (src%srcstf(i) == 1) then
673         stf_val = -stfGauss(time,src%srcf0(i),t0(i)+src%delay(i),src%
                ↪ srcfactor(i))
674         stf_val_diff = -stfDiffGauss(time,src%srcf0(i),t0(i)+src%delay(i),
                ↪ src%srcfactor(i))
675     else if (src%srcstf(i) == 2) then !RICKER
676         stf_val = -stfRicker(time,src%srcf0(i),t0(i)+src%delay(i),src%
                ↪ srcfactor(i))
677         stf_val_diff = -stfDiffRicker(time,src%srcf0(i),t0(i)+src%delay(i),
                ↪ src%srcfactor(i))
678     else if (src%srcstf(i) == 3) then !Sin^3
679         stf_val = -stfSin3(time,src%srcf0(i),t0(i)+src%delay(i),src%
                ↪ srcfactor(i))
680         stf_val_diff = -stfDiffSin3(time,src%srcf0(i),t0(i)+src%delay(i),
                ↪ src%srcfactor(i))
681     end if
682     if (src%srcstf(i) == 4) then
683         if (it<par%nt) then
684             stf_val=(1.-rk_time)*plotstf(it,2,i)+rk_time*plotstf(it+1,2,i)
685             stf_val_diff=(1.-rk_time)*plotDiffstf(it,2,i)+rk_time*
                    ↪ plotDiffstf(it+1,2,i)
686         else
687             stf_val=(1.+rk_time)*plotstf(it,2,i)-rk_time*plotstf(it-1,2,i)
688             stf_val_diff=(1.+rk_time)*plotDiffstf(it,2,i)-rk_time*
                    ↪ plotDiffstf(it-1,2,i)
689         endif
690     end if
691 enddo
```

Listing B.4: Code snippet from *timeloopMod* (2).