SW4

User’s Guide
Version 2.0

N. Anders Petersson
Bjorn Sjogreen

www.geodynamics.org
Disclaimer  This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

Auspices Statement  This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.
4.5.1 Lamb’s problem ........................................ 29

5 Topography .................................................... 33
  5.1 Gaussian hill topography .................................. 34
  5.2 Topography grid file ....................................... 34
  5.3 rfile topography .......................................... 34
  5.4 vfile topography ......................................... 35

6 The material model ....................................... 37
  6.1 The block command ....................................... 38
  6.2 The vfile command ....................................... 39
  6.3 The rfile command ....................................... 41
  6.4 The pfile command ....................................... 43
  6.5 The ifile command ....................................... 45
  6.6 The vimaterial command .................................. 46

7 Mesh refinement ........................................... 47

8 Attenuation .................................................. 51
  8.1 Viscoelastic modeling ..................................... 51

9 Output options ............................................. 54
  9.1 Setting the output directory .............................. 54
  9.2 Time-history at a receiver station: the rec (or sac) command ........................................... 55
    9.2.1 The ASCII text format ................................ 57
    9.2.2 Notes on the rec command ............................. 57
  9.3 2-D cross-sectional data: the image command ............ 57
  9.4 Creating a GMT script with the gmt command ............ 59

10 Examples .................................................... 60
  10.1 The elastic layer over half-space problem: LOH.1 ...... 60
  10.2 The visco-elastic layer over half-space problem: LOH.3 ........................................... 62

11 Keywords in the input file ................................ 65
  11.1 Basic commands ......................................... 66
    11.1.1 fileio [optional] ..................................... 66
    11.1.2 grid [required] ........................................ 66
    11.1.3 time [required] ........................................ 68
    11.1.4 supergrid [optional] .................................... 69
    11.1.5 prefilter [optional] ..................................... 69
  11.2 Sources [required] ....................................... 70
    11.2.1 source ................................................ 70
    11.2.2 rupture .............................................. 72
  11.3 The material model [required] ............................. 72
    11.3.1 attenuation [optional] ................................ 72
    11.3.2 block ................................................. 73
    11.3.3 vfile ................................................ 75
B. Run time and memory requirements

B.1 Run time .................................................. 114
B.2 Memory usage ............................................. 115
Chapter 1

Introduction

SW4 (Seismic Waves, 4th order) is a program for simulating seismic wave propagation on parallel computers. It shares many features with our previous seismic wave propagation code WPP [15]. Both WPP and SW4 solve the seismic wave equations in displacement formulation using a node-based finite difference approach. The numerical method satisfies the principle of summation by parts, which guarantees energy stability of the numerical solution. The major difference between WPP and SW4 lies in the accuracy of the underlying numerical method. SW4 is fourth order accurate in space and time [20], but WPP is only second order accurate. SW4 is therefore significantly more efficient than WPP, because a coarser grid can be used to capture waves with the same frequency content. Compared to a second order accurate method, the advantages of a fourth order method are more pronounced when the solution needs to be more accurate. This is because the error diminishes at a faster rate as the grid size is reduced. A fourth order method is also more efficient when the solution needs to remain accurate for longer times, because the phase error grows at a slower rate in a higher order numerical method [8]. Keeping the phase error small is, for example, important to accurately predict the arrival times of waves that have propagated over many wave lengths. The fourth order method is also significantly more accurate for calculating surface waves, in particular when the ratio between the compressional and shear wave speeds is large, i.e. \( C_p/C_s \gg 1 \), see [9].

SW4 implements substantial capabilities for 3-D seismic modeling, with a free surface condition on the top boundary, absorbing super-grid conditions on the far-field boundaries [17], and an arbitrary number of point force and/or point moment tensor source terms. Each source time function can have one of many predefined analytical time dependencies, or interpolate a user defined discrete time series. SW4 supports a fully 3-D heterogeneous material model that can be specified in several formats. It uses a curvilinear mesh near the free surface to honor the free surface boundary condition on a realistic topography. The curvilinear mesh is automatically generated from the description of the topography. To make SW4 more computationally efficient, the seismic wave equations are discretized on a Cartesian mesh below the curvilinear grid. The Cartesian mesh, which extends to the bottom of the computational domain, is also generated automatically.

SW4 solves the seismic wave equations in Cartesian coordinates. It is therefore appropriate for local and regional simulations, where the curvature of the earth can be neglected. Locations can be specified directly in Cartesian coordinates, or through geographic (latitude, longitude) coordinates. SW4 can be built to use the Proj.4 library [13] for calculating the mapping between geographic and Cartesian coordinates, or use an approximate spheroidal mapping. SW4 can output synthetic seismograms in an ASCII text format, or in the SAC format [7]. It can also present simulation
information as GMT [21] scripts, which can be used to create annotated maps. SW4 can output
the solution, derived quantities of the solution, as well as the material model along 2-D grid planes.
Furthermore, SW4 can output the 3-D volumetric solution, or material model, in a binary file
format.

Cartesian local mesh refinement can be used to make the computational mesh finer near the
free surface, where more resolution often is needed to resolve short wave lengths in the solution,
for example in sedimentary basins. The mesh refinement is performed in the vertical direction
and each Cartesian grid is constructed from user specified refinement levels. In this approach, the
grid size in all three spatial directions is doubled across each mesh refinement interface, leading to
substantial savings in memory and computational effort. The energy conserving mesh refinement
coupling method described in [14], but generalized to fourth order of accuracy, is used to handle
the hanging nodes along the refinement interface.

Visco-elastic behavior can be important when modeling the dissipative nature of realistic ma-
terials, especially for higher frequencies. SW4 uses the rheological model of standard linear solid
(SLS) elements, coupled in parallel. The coefficients in each SLS are determined such that the
resulting quality factors $Q_p$ and $Q_s$, for the attenuation of P- and S-waves, become approximately
constant as function of frequency. These quality factors can vary from grid point to grid point over
the computational domain and are read in the same way as the elastic properties of the material
model. The numerical method for solving the visco-elastic wave equation is based on the technique
described in [16].

While most of the SW4 code is written in C++, almost all numerical computations are imple-
mented in Fortran-77. SW4 uses a distributed memory programming model, implemented with the
C-bindings of the MPI library. Compatible versions of the C++ and Fortran-77 compilers as well
as the MPI library must be available to build the code. We have built and tested SW4 on a variety
of machines, ranging from single processor laptops to large super-computers with $O(100,000)$ cores.

With the exception of some minor details, the syntax of the SW4 command file is the same
as in WPP. Most of the input and output files also use the same formats, but we have taken the
opportunity to improve the image file format, see Chapter [12]. SW4 supports most of the func-
tionality of WPP, version 2.2. Compared to version 1.1 of SW4, the main improvement in version
2.0 is the refinement command for using mesh refinement with hanging nodes. A preliminary
implementation of mesh refinement was first introduced in version 1.18. It has now been further
improved and generalized to support anelastic materials.

The examples subdirectory of the SW4 source distribution contains several examples and vali-
dation tests that are used in this document. Many Matlab/octave scripts are provided in the tools
directory.

1.1 How to cite SW4

The Computational Infrastructure for Geodynamics (CIG) (geodynamics.org) makes the SW4
source code available to you at no cost in hope that the software will benefit your research in
geophysics. A number of individuals have contributed a significant portion of their careers toward
the development of this software. It is essential that you recognize these individuals in the normal
scientific practice by citing the appropriate peer-reviewed papers and making appropriate acknowl-
edgments in talks and publications. The following peer-reviewed papers discuss the numerical
methods that are implemented in SW4:


To cite the SW4 software and manual, use:


1.2 Acknowledgments

The SW4 code was developed under financial support from Lawrence Livermore National Laboratory. The underlying numerical method was developed under financial support from the Office of Science at the U.S. Department of Energy.
Chapter 2

Getting started

2.1 Running \( SW_4 \)

This section assumes that \( SW_4 \) has already been installed on your computer system. We refer to the report by Petersson and Sjogreen [19] for instructions on how to install \( SW_4 \).

\( SW_4 \) can be executed from the command line or from a script. The simulation is specified by the input file, and the name of the input file is given on the command line. The input file is an ASCII text file that contains a number of commands specifying the properties of the simulation, such as the dimensions of the computational domain, grid spacing, the duration of the simulation, the material properties, the source model, as well as the desired output. To improve readability of this document we have used the continuation character “\" to extend long commands to the subsequent line. There is however no support for continuation characters in \( SW_4 \), so each command must be given on one (sometimes long) line in the input file.

Since \( SW_4 \) is a parallel code, it is required to be run under a parallel execution environment such as mpiexec, mpirun, openmpirun, or srun. The \texttt{srun} command is currently always used on the parallel machines at Livermore Computing. It is important to start \( SW_4 \) with the correct parallel execution tool. For example, if you build \( SW_4 \) with the \texttt{openmpi} compilers, you should use the \texttt{openmpirun} environment. Also note that some systems require you to start an \texttt{mpd} daemon before running any parallel programs. Chances are high that somebody else has already figured out how to run parallel programs on your system. If you have problems running \( SW_4 \), ask your local system administrator, or somebody else who has experience running MPI programs on your system.

Throughout this document we use the convention that input files have the file suffix \texttt{.in}. However, \( SW_4 \) will attempt to read any input file, regardless of its extension.

If your system is setup for using \texttt{mpiexec}, the command

\begin{verbatim}
    shell> mpiexec -np 2 sw4 test.in
\end{verbatim}

runs \( SW_4 \) on 2 processes, and tells it to read the input from a file named \texttt{test.in}. If you are using \texttt{mpirun}, you would instead use the command

\begin{verbatim}
    shell> mpirun -np 2 sw4 test.in
\end{verbatim}

Remark: If \( SW_4 \) produces strange looking outputs, for example where the same text is repeated several times (e.g. once per processor), you are probably running \( SW_4 \) under the wrong parallel execution environment. Make sure you are running \( SW_4 \) under an environment that is compatible with the compiler that was used to build \( SW_4 \).
2.1.1 Version information (-v)

Version information for the SW4 executable can be obtained through the \(-v\) flag:

```bash
shell> mpirun optimize/sw4 -v
```

```
sw4 version 2.0
```

This program comes with ABSOLUTELY NO WARRANTY; released under GPL. This is free software, and you are welcome to redistribute it under certain conditions, see LICENSE.txt for more details.

```
Compiled on: Mon Nov 6 09:11:04 PST 2017
By user: petersson1
Machine: fourier.llnl.gov
Compiler: /opt/local/bin/mpicxx
3rd party include dir: /opt/local/lib/proj47/include, and library dir: /opt/local/lib/proj47/
```

Note that the same information is by default printed to standard out (your screen, or log file when running in batch mode) at the beginning of every run.

2.1.2 Running on the parallel machines at Livermore Computing

The srun command is currently used to run parallel jobs on LC machines. For example, the command

```bash
shell> srun -ppdebug -n 32 sw4 xxx.in
```

runs SW4 on 32 processors on the debug partition using xxx.in as the input file. Note that the pdebug partition is intended for shorter jobs. It is subject to both a CPU time limit and a limit on the number of processors per job. Jobs requiring more computer resources must be submitted through the batch system, currently using the msub command. Refer to the Livermore Computing web pages for detailed information (https://computing.llnl.gov).
Chapter 3

Governing equations, coordinate system, and units

$SW_4$ simulates the motion due to a seismic event by solving the elastic or visco-elastic wave equations in displacement formulation. This is a system of linear hyperbolic partial differential equations in second order formulation. By second order formulation, we mean that the partial differential equation contains second derivatives with respect to space and time. The equations are solved in the three-dimensional spatial domain $x \in \Omega$ during the time interval $0 \leq t \leq T$. By default, the motion starts from rest and is driven by a forcing function $F(x,t)$. In the elastic case, the motion is governed by

$$\rho u_{tt} = \nabla \cdot T + F(x,t), \quad x \in \Omega, \ 0 \leq t \leq T, \quad (3.1)$$

$$u(x,0) = 0, \quad u_t(x,0) = 0, \quad x \in \Omega. \quad (3.2)$$

Here, $\rho$ is the density, $u(x,t)$ is the displacement vector, and $T = T(u)$ is the stress tensor. The computational domain $\Omega$ is a box shaped region where one side optionally follows the shape of the topography. By default, a free surface (also called traction-free, or zero normal stress) boundary condition is enforced along the top boundary,

$$T \cdot n = 0, \quad z = \tau(x,y), \quad t \geq 0.$$ 

Here $n$ is the unit normal of the $z = \tau(x,y)$ surface. By default, a super-grid damping layer is used on all other sides of the computational domain.

$SW_4$ uses a right-handed Cartesian coordinate system with the z-direction pointing downwards into the medium, see figure 3.1. $SW_4$ employs MKS (meters-kilograms-seconds) units. All distances (e.g., grid dimensions, spacing, and displacements) are in meters (m), time is in seconds (s), seismic P- and S-wave velocities are in meters per second (m/s), densities are in kilogram per cubic meter (kg/m$^3$), forces are in Newton (N), and seismic moment (torque) is in Newton-meters (Nm). All angles (e.g. latitude, longitude, azimuth, strike, dip and rake) are in degrees.

In $SW_4$ the computational domain is rectangular in the horizontal plane and the vertical extent is defined by the topographic surface

$$z = \tau(x,y),$$

which defines the shape of the free surface. $SW_4$ can also be run with flat topography, in which case $\tau(x,y) = 0$ and the $z$-coordinate equals the depth below the free surface. In the general case,
Figure 3.1: SW4 uses a right handed coordinate system with the z-axis pointing downwards.

the computational domain is given by

$$0 \leq x \leq x_{\text{max}}, \quad 0 \leq y \leq y_{\text{max}}, \quad \tau(x, y) \leq z \leq z_{\text{max}}.$$  \hspace{1cm} (3.3)

The grid command in the input file specifies the extent of the computational domain and the grid size $h$. When topography is enabled, the grid size in the curvilinear grid equals $h$ in the horizontal directions, but varies in the vertical direction to allow the curvilinear grid to follow the shape of the free surface. In this case, the number of grid points in the vertical direction is chosen such that the average of the vertical grid size approximately equals $h$. When mesh refinement is enabled, this is the grid size in the coarsest grid.

The most precise way of specifying the grid is by providing the number of grid points in each direction as well as the grid size,

```
grid nx=301 ny=201 nz=101 h=500.0
```

This command gives a grid with grid size 500 meters, which extends 150 km in $x$, 100 km in $y$ and 50 km in the $z$-direction. Alternatively, the grid can be specified by giving the spatial range in each of the three dimensions and explicitly specifying the grid spacing. For example, the command

```
grid x=30e3 y=20e3 z=10e3 h=500.0
```

results in a grid which spans 30,000 meters in $x$, 20,000 meters in $y$, and 10,000 meters in the $z$-direction. The grid spacing is 500 meters, which is used to compute the number of grid points in each direction: $nx=61$, $ny=41$, and $nz=21$, for a total of 52,521 grid points. Note that the number of grid points in the different directions will be rounded to the nearest integer value according to the pseudo C-code

$$nx = (\text{int})(1.5 + x/h).$$  \hspace{1cm} (3.4)

The extent in the $x$-direction is thereafter adjusted to

$$x = (nx - 1)h.$$  \hspace{1cm} (3.5)

A corresponding procedure is performed in the other coordinate directions.

The third option is to give the spatial range in each of the three dimensions and specify the number of grid points in one direction:
grid x=30000 y=20000 z=10000 nx=100

In this case, the grid spacing is computed as

\[ h = \frac{x}{nx - 1} = 303.03. \]

Note that no rounding needs to take place in this case, since \( h \) is a floating point number. Given this value of \( h \), ny and nz are computed using formulas corresponding to (3.4) giving \( ny=34 \) and \( nz=67 \), for a total of 227,800 grid points. Again, the extents in the \( y \) and \( z \)-directions are adjusted corresponding to (3.5). The syntax for the grid command is given in Section 11.1.2.

The simulation always starts at \( t = 0 \) and runs until \( t = T \), where the user must specify \( T \) with the \texttt{time} command. For example,

\[ \text{time } t=1.6 \]

sets \( T = 1.6 \) seconds. Alternatively, the simulation time interval can be specified as a number of time steps,

\[ \text{time steps}=1200 \]

The end time will in this case be \( T = 1200\Delta t \), where the time step \( \Delta t \) is determined automatically by \texttt{SW4} to satisfy the CFL time step restriction. This calculation is based on the ratio between the grid size and the largest characteristic wave speed, which depends on both the compressional and shear wave speeds.

The simulation start time can be related to a universal time coordinate (UTC). The option \texttt{utcstart} sets the UTC that corresponds to simulation time \( t = 0 \), for example,

\[ \text{time } t=1.6 \text{ utcstart}=01/31/2012:17:34:12.233 \]

The format of the UTC is “month/day/year:hour:minute:second.millisecond”. When the UTC time is set, it is saved in the header of all time series files (see the \texttt{rec} command). Note that the UTC can be very useful for aligning simulated and observed time series.

### 3.1 Geographical coordinates and projections

\texttt{SW4} supports geographical coordinates as an alternative way of specifying spatial locations. The geographic location of the origin of the Cartesian coordinate system (lat\(_0\), lon\(_0\)), see Figure 3.2, is specified in the grid command. If no location is given, the default location is lat\(_0 = 37\) degrees, lon\(_0 = -118\) degrees, and the azimuthal angle of the \( x \)-axis is 135 degrees from North. We remark that latitudes are positive North of the equator and longitude are positive East of the Greenwich prime meridian. The vertical coordinate increases downwards. For the case of general topography, \( z = 0 \) corresponds to mean sea level. When the topography is flat (no \texttt{topography} command in the input file), \( z = 0 \) corresponds to the free surface.

#### 3.1.1 Spheroidal mapping

By default, the latitude (lat) and longitude (lon) are calculated using a spheroidal mapping,

\[ \begin{align*}
\text{lat} & = \text{lat}_0 + \frac{x \cos(\alpha) - y \sin(\alpha)}{M}, \\
\text{lon} & = \text{lon}_0 + \frac{x \sin(\alpha) + y \cos(\alpha)}{M \cos(\phi \pi/180)}.
\end{align*} \tag{3.6} \tag{3.7} \]

13
In this formula, \( \text{lat}, \text{lon}, \text{az}, \text{lat}_0, \) and \( \text{lon}_0 \) are all in degrees, and \( M = 111,319.5 \) meters\(^1\). You can change the location and orientation of the grid by specifying the latitude and longitude of the grid origin, as well as the azimuthal angle between North and the \( x \)-axis. For example:

\[
\text{grid h=500 x=30000 y=20000 z=10000 lat=39 lon=-117 az=150}
\]

sets the origin of the grid to latitude 39 degrees (North), longitude -117 degrees (West), and azimuthal angle 150 degrees.

The default projection is spheroidal as described by equations (3.6)-(3.7). You can change the parameter \( M \) with the \texttt{mlat} keyword in the \texttt{grid} command. By using the \texttt{mlon} keyword, you can also modify the projection by replacing \( M \cos(\phi \pi/180) \) in (3.7) by the constant value \( M_{\text{lon}} \). Using the \texttt{mlon} keyword is only recommended if the computational domain is small and accurate values of both \texttt{mlon} and \texttt{mlat} are available.

### 3.1.2 The Proj.4 library

More accurate projections are available through the Proj4 library (if \textit{SW4} was built with Proj4 support). These projections are enabled by using the \texttt{proj}, or any of the related, keywords in the \texttt{grid} command. For example,

\[
\text{grid h=300 x=40e3 y=43e3 z=40e3 lat=45.01 lon=5.52 az=0 proj=utm ellps=WGS84}
\]

\(^1\)Note that \( M/60 = 1,855.325 \) meters corresponds to one minute of arc of longitude along the Equator on the WGS84 ellipsoid. This distance is also known as a geographical mile and is approximately equal to a Nautical mile (1,852 meters).
sets the origin of the grid to latitude 45.01 degrees (North), longitude 5.52 degrees (East), and azimuthal angle 0. Here we use the UTM projection based on the WGS84 ellipse. Note that the strings “proj=utm” and “ellps=WGS84” are passed directly to the Proj4 library to initialize the projection. Several other options are available, see Section 11.1.2 and the Proj4 documentation [13] for further details.

3.2 Super-grid damping layers

SW4 implements a super-grid modeling technique to reduce artificial reflections from the far-field boundaries [2][17]. In this method, layers are added around the domain of interest, i.e., the domain in which we want to solve the seismic wave equation. In each layer, a stretching function is used to transform the seismic wave equation to mimic a much larger physical domain. The basic idea is to delay artificial reflections from the far-field boundary, because making the physical domain larger means that it will take longer for waves to arrive at the boundary, and then return back into the domain of interest. Inside the layers, the stretching function is combined with a high order artificial dissipation. It damps out waves that have become poorly resolved on the grid because of the stretching function. Note that the artificial dissipation is only added in the layers, and should not affect the accuracy of the solution in the interior of the domain.

The coordinate stretching compresses the solution inside the layers. This corresponds to a slowing down of all traveling waves in the direction normal to each far-field boundary. As a result, the isotropic elastic wave equation becomes anisotropic in the super-grid layers. By using energy estimates, it is possible to prove that the super-grid technique leads to a stable numerical method where the total energy of the solution decreases with time. This estimate is valid for heterogeneous material properties and free surface boundary conditions on one or more sides of the domain, and also extends to anisotropic elastic materials and curvilinear grids. See the papers by Petersson and Sjogreen [17][18] for details.

Super-grid layers are by default added to all sides of the computational domain, except along the free surface, see Figure 3.3. The default thickness of the layers is 30 grid sizes, but the thickness of the super-grid layers can be changed with the supergrid command, see Section 11.1.4. Note that long waves are harder to suppress than short waves, because the artificial damping is less efficient for waves that are well resolved on the grid. The best way of reducing artificial reflections is to make the super-grid layers thicker. Increasing the dissipation coefficient is not recommended as it can make the explicit time stepping unstable. Reducing the thickness of the super-grid layers to below 20 grid sizes can also lead to instabilities, unless the artificial dissipation coefficient is also reduced. The syntax of the supergrid command is described in Section 11.1.4.

For reasons, the super-grid layers are part of the computational domain as specified by the grid command. If, for example, each super-grid layer is 30 grid points wide, the solution should be considered artificial in the first and last 30 points in each horizontal direction, and the bottom 30 points in the vertical direction. Note that the numerical solution within the layers do not approximate the solution of the seismic wave equations. For this reason, it is important to make the computational domain sufficiently large. If the super grid layers are 30 grid points wide, the computational grid must be at least 60 grid points wide in the x and y-directions, and 30 points wide in the Cartesian part of the z-direction. Additional grid points must be added in the interior of the computational domain for the actual seismic modeling. Note that sources and receivers should only be placed in the interior of the computational domain, i.e., the white region of Figure 3.3.
Figure 3.3: *A vertical cross-section through the computational domain with a free surface boundary along the top edge. The original seismic wave equation is solved the white region. The wave speed is reduced in the normal direction of the surrounding super-grid layers, where also a high order damping term is added.*

**Remark:** As a user of $SW_4$, you do not have to worry about the stretching functions or the artificial dissipation in the super-grid layers, because they are set up automatically. Just make sure the computational grid has a sufficient number of grid points to accommodate the layers, and be aware that the seismic wave equation is modified in the layers, which makes that part of the solution artificial.
Chapter 4

Sources, time-functions, and grid size

4.1 Point force and moment tensor sources in SW4

The forcing term \( F \) in equation (3.1) consists of a sum of point forces and point moment tensor source terms. For a point forcing we have

\[
F(x, t) = g(t, t_0, \omega) F_0 \left[ \begin{array}{c} F_x \\ F_y \\ F_z \end{array} \right] \delta(x - x_0),
\]

where \( x_0 = (x_0, y_0, z_0) \) is the location of the point force in space, and \( g(t, t_0, \omega) \) is the time function, with offset time \( t_0 \) and frequency parameter \( \omega \). The source time function can be selected from a set of predefined functions, or by spline interpolation of a user defined discrete time-series. The \( (F_x, F_y, F_z)^T \) vector holds the Cartesian components of the force vector, which is scaled by the force amplitude \( F_0 \).

For a moment tensor source we have

\[
F(x, t) = g(t, t_0, \omega) \mathbf{M} \cdot \nabla \delta(x - x_0), \quad \mathbf{M} = \left[ \begin{array}{ccc} M_{xx} & M_{xy} & M_{xz} \\ M_{xy} & M_{yy} & M_{yz} \\ M_{xz} & M_{yz} & M_{zz} \end{array} \right].
\]

The seismic moment of a moment tensor source is defined by

\[
M_0 = \sqrt{\mathbf{M} : \mathbf{M}} = \left[ (M_{xx}^2 + M_{yy}^2 + M_{zz}^2) + 2(M_{xy}^2 + M_{xz}^2 + M_{yz}^2) \right]^{1/2}.
\]

Note that the moment tensor is always symmetric. A moment source term can alternatively be specified by using \( M_0 \) and the dip, strike, and rake angles, as defined by Aki and Richards [1]. The syntax is described in Section [11.2.1].

The total seismic moment \( \sum M_0 \) [Nm] is related to the moment magnitude by the formula

\[
M_W = \frac{2}{3} \left[ \log_{10} \left( \sum M_0 \right) - 9.1 \right],
\]

where the summation \( \sum M_0 \) is done over all moment sources. After parsing all source commands in an input file, SW4 outputs the moment magnitude using this formula. This information is given right before the time-stepping is started and looks like this:
Total seismic moment \((M_0)\): \(1.7162 \times 10^{17}\ Nm\)
Moment magnitude \((M_w)\): \(5.42305\)
Number of sources 542

Note that the calculation of the total seismic moment and magnitude only take the moment tensor sources into account, i.e., ignores all point forces.

For moment tensor sources, the function \(g(t)\) is called the moment history time function, while its time derivative \(g'(t)\) is known as the moment rate time function. \(SW4\) calculates the displacements of the motion corresponding to the moment history time function \(g(t)\). Because the material properties are independent of time, the equations solved by \(SW4\) also govern the velocities when the time function is replaced by \(g'(t)\), i.e., the corresponding moment rate time function. For example, if the solution calculated with the GaussianInt time function represents the displacements of the motion, the solution calculated with the Gaussian time function corresponds to the velocities of the same motion. Hence, if you are primarily interested in calculating velocities, you can reduce the amount of post processing by using the corresponding moment rate time function in the source term(s).

If you are interested in comparing results from \(SW4\) with some other code, keep in mind that many other seismic wave propagation codes are based on the first order velocity-stress formulation of the elastic wave equation. Such codes solve for the velocities of the motion. They use the moment rate time function \((g'(t))\) for moment tensor sources, but the regular time function \((g(t))\) for point forces.

The forcing function in \(SW4\) is specified in the input file using at least one source or rupture command. These options can be combined. The rupture command allows complicated source mechanisms to be described in a separate SRF (Standard Rupture Format) file. It is equivalent to at least one (but often many) source command(s). There needs to be at least one source command in the input file in order for anything to happen during the simulation. Complicated source mechanisms can be described by having many source commands in the input file. An example with one source command is:

```
source x=5000 y=4000 z=600 mxx=1e15 myy=1e15 mzz=1e15 \
    type=RickerInt t0=1 freq=5
```

The above command specifies an isotropic source (explosion) at the location \((5000, 4000, 600)\) with amplitude \(10^{15}\ Nm\), using the RickerInt time function with offset time \(t_0 = 1\ s\) and frequency parameter \(w = 5\ Hz\). The off-diagonal moment tensor elements \((M_{xy}, M_{xz} \text{ and } M_{yz})\) are implicitly set to zero (which is the default value).

Note that it is not necessary to place the sources exactly on grid points. The discretization of the source terms is fourth order accurate for any location within the computational domain, including the free surface. However, unexpected results may be obtained if the sources are located in the super-grid far-field layers.

### 4.2 Predefined time functions

All pre-defined source time functions start from zero \(\lim_{t \to -\infty} g(t, t_0, \omega) = 0\) and tend to a constant terminal value, \(\lim_{t \to \infty} g(t, t_0, \omega) = g_\infty\). In seismic applications, time function that have a non-zero terminal value \((g_\infty \neq 0)\) lead to a non-zero steady-state solution after long times. Such time
functions are used to solve for the displacement of the motion. When \( g_\infty = 0 \), the solution tends to zero for large times. This is expected from the velocities or accelerations of the motion due to a seismic event.

The Gaussian, Dirac, and Triangle functions integrate to one (\( \int_{-\infty}^{\infty} g(t, t_0, \omega) dt = 1 \)), while the Sawtooth, Smoothwave, and Ricker functions integrate to zero and have maximum amplitude one. The RickerInt function is the time-integral of the Ricker function and integrates to zero. The GaussianInt, Brune, BruneSmoothed, and Liu functions tend to one (\( \lim_{t \to \infty} g(t, t_0, \omega) = 1 \)).

The initial conditions are homogeneous when \( \text{SW4} \) is used to calculate the motion due to point force and moment tensor sources. In other words, the initial displacement and velocity are zero. To avoid incompatibilities, the source time functions must also start smoothly. Since the Triangle, Sawtooth, Ramp, Smoothwave, Brune, BruneSmoothed, Liu and VerySmoothBump functions are identically zero for \( t < t_0 \), these time functions must have \( t_0 \geq 0 \). More care is required for the Gaussian, GaussianInt, Ricker, and RickerInt functions, because they are centered around \( t = t_0 \), with exponentially decaying tails for \( t < t_0 \). For these functions, incompatibility problems can only be avoided if \( t_0 \) is positive and of the order \( O(1/\omega) \), where \( \omega \) equals the \texttt{freq} parameter. We recommend choosing \( t_0 \) such that \( g(0, t_0, \omega) \leq 10^{-8} \) for these functions.

4.2.1 Gaussian

\[
g(t, t_0, \omega) = \frac{\omega}{\sqrt{2\pi}} e^{-\omega^2(t-t_0)^2/2}.
\]

Note that the spread of the Gaussian function (often denoted \( \sigma \)) is related to \( \omega \) by \( \sigma = 1/\omega \). A plot of the Gaussian time-function is shown in Figure 4.1.

**Important:** To avoid artifacts from a sudden startup, use \( t_0 \geq 6/\omega \).

4.2.2 GaussianInt (or Erf)

\[
g(t, t_0, \omega) = \frac{\omega}{\sqrt{2\pi}} \int_{-\infty}^{t} e^{-\omega^2(\tau-t_0)^2/2} d\tau.
\]

GaussianInt is the time-integral of the Gaussian. A plot of the GaussianInt time-function is shown in Figure 4.1.

**Important:** To avoid artifacts from a sudden startup, use \( t_0 \geq 6/\omega \).

4.2.3 Ricker

\[
g(t, t_0, \omega) = (2\pi^2 \omega^2 (t-t_0)^2 - 1) e^{-\pi^2 \omega^2 (t-t_0)^2}.
\]

A plot of the Ricker time-function is shown in Figure 4.2

**Important:** To avoid artifacts from a sudden startup, use \( t_0 \geq 1.35/\omega \).
Figure 4.1: Gaussian (left) and GaussianInt (right) with $\omega = \pi$ and $t_0 = 0$.

### 4.2.4 RickerInt

$$g(t, t_0, \omega) = (t - t_0)e^{-\pi^2\omega^2(t-t_0)^2}.$$  

RickerInt is the time integral of the Ricker function, and is proportional to the time-derivative of the Gaussian function. Since the RickerInt function tends to zero for large times, it does not lead to any permanent displacements. A plot of the RickerInt time-function is shown in Figure 4.2.

**Important:** To avoid artifacts from a sudden startup, use $t_0 \geq 1.35/\omega$.

### 4.2.5 Brune

$$g(t, t_0, \omega) = \begin{cases} 0, & t < t_0, \\ 1 - e^{-\omega(t-t_0)}(1 + \omega(t-t_0)), & t \geq t_0. \end{cases}$$

Note that the Brune function only has one continuous derivative. Because its second derivative is discontinuous at $t = t_0$, this function can generate noisy numerical solutions. We recommend filtering all computed time series, or using the `prefilter` command to remove any unresolved motions.

### 4.2.6 BruneSmoothed

The BruneSmoothed function has three continuous derivatives at $t = t_0$, but is otherwise similar to the Brune function,

$$g(t, t_0, \omega) = \begin{cases} 0, & t < t_0, \\ 1 - e^{-\omega(t-t_0)} \left[ 1 + \omega(t-t_0) + \frac{1}{2}(\omega(t-t_0))^2 \\ - \frac{3}{2\tau_0^3}(\omega(t-t_0))^3 + \frac{3}{2\tau_0^6}(\omega(t-t_0))^4 - \frac{1}{2\tau_0^9}(\omega(t-t_0))^5 \right], & 0 < \omega(t-t_0) < \tau_0, \\ 1 - e^{-\omega(t-t_0)}(1 + \omega(t-t_0)), & \omega(t-t_0) > \tau_0. \end{cases}$$

The parameter $\tau_0$ in the above formula is fixed to the value 2.31. Plots of the Brune and BruneSmoothed time-functions are shown in Figure 4.3. Since the BruneSmoothed function has three
continuous derivatives, it generates less high frequency noise than the Brune function and gives better accuracy for a given grid resolution.

4.2.7 Liu

This function was given in the paper by Liu et al., [11]. It is defined by

\[
g(t, t_0, \omega) = \begin{cases} 
0, & t \leq t_0, \\
C \left[ 0.7(t - t_0) + \frac{1.2}{\pi} \tau_1 - \frac{1.2}{\pi} \tau_1 \cos \left( \frac{\pi(t - t_0)}{2\tau_1} \right) \\
- \frac{0.7}{\pi} \tau_1 \sin \left( \frac{\pi(t - t_0)}{\tau_1} \right) \right], & 0 < t \leq \tau_1 + t_0, \\
C \left[ t - t_0 - 0.3\tau_1 + \frac{1.2}{\pi} \tau_1 - \frac{0.7}{\pi} \tau_1 \sin \left( \frac{\pi(t - t_0)}{\tau_1} \right) \\
+ \frac{0.3}{\pi} \tau_2 \sin \left( \frac{\pi(t - t_0 - \tau_1)}{\tau_2} \right) \right], & \tau_1 + t_0 < t \leq 2\tau_1 + t_0, \\
C \left[ 0.3(t - t_0) + 1.1\tau_1 + \frac{1.2}{\pi} \tau_1 \\
+ \frac{0.3}{\pi} \tau_2 \sin \left( \frac{\pi(t - t_0 - \tau_1)}{\tau_2} \right) \right], & 2\tau_1 + t_0 < t \leq \tau + t_0, \\
1, & t > \tau + t_0.
\end{cases}
\]

The parameters are given by \( \tau = 2\pi/\omega, \tau_1 = 0.13\tau, \tau_2 = \tau - \tau_1, \) and \( C = \pi/(1.4\tau_1\pi + 1.2\tau_1 + 0.3\tau_2\pi). \)

The Liu function resembles the Brune function, but the rise is somewhat steeper for small \( t - t_0, \) see Figure 4.4.
Figure 4.3: Brune (left) and BruneSmoothed (right) with $\omega = 2$ and $t_0 = -1$.

Figure 4.4: Liu time function with $\omega = 2$ and $t_0 = 0$. 
Figure 4.5: Triangle (left) and Sawtooth (right) with $\omega = 1$ and $t_0 = 0$.

4.2.8 Triangle

For $t_0 < t < t_0 + 1/\omega$,

$$g(t, t_0, \omega) = \frac{16\omega}{\pi^2} \left[ \sin(\pi\omega(t - t_0)) - \frac{\sin(3\pi\omega(t - t_0))}{9} + \frac{\sin(5\pi\omega(t - t_0))}{25} - \frac{\sin(7\pi\omega(t - t_0))}{49} \right],$$

with $g(t, t_0, \omega) = 0$ elsewhere. A plot of the Triangle time-function is shown in Figure 4.5.

4.2.9 Sawtooth

For $t_0 < t < t_0 + 1/\omega$,

$$g(t, t_0, \omega) = \frac{8}{\pi^2} \left[ \sin(2\pi\omega(t - t_0)) - \frac{\sin(6\pi\omega(t - t_0))}{9} + \frac{\sin(10\pi\omega(t - t_0))}{25} - \frac{\sin(14\pi\omega(t - t_0))}{49} \right],$$

with $g(t, t_0, \omega) = 0$ elsewhere. A plot of the Sawtooth time-function is shown in Figure 4.5.

4.2.10 Ramp

$$g(t, t_0, \omega) = \begin{cases} 0, & t < t_0, \\ 0.5(1 - \cos(\pi(t - t_0)\omega)), & t_0 \leq t \leq t_0 + 1/\omega, \\ 1, & t > t_0 + 1/\omega. \end{cases}$$

A plot of the Ramp time-function is shown in Figure 4.6.
4.2.11 Smoothwave

For $t_0 < t < t_0 + 1/\omega$,

$$g(t, t_0, \omega) = \frac{2187}{8} (\omega(t - t_0))^3 - \frac{10935}{8} (\omega(t - t_0))^4 + \frac{19683}{8} (\omega(t - t_0))^5$$

$$- \frac{15309}{8} (\omega(t - t_0))^6 + \frac{2187}{4} (\omega(t - t_0))^7,$$

with $g(t, t_0, \omega) = 0$ elsewhere. A plot of the Smoothwave time-function is shown in Figure 4.6.

4.2.12 VerySmoothBump

$$g(t, t_0, \omega) = \begin{cases} 
0, & t < t_0, \\
1024 \omega^5(t - t_0)^5(1 - \omega(t - t_0))^5, & t_0 \leq t \leq t_0 + 1/\omega, \\
0, & t > t_0 + 1/\omega.
\end{cases}$$

The VerySmoothBump function satisfies $0 \leq g \leq 1$. It has four continuous derivatives. A plot of the VerySmoothBump time-function is shown in Figure 4.7.

4.2.13 C6SmoothBump

$$g(t, t_0, \omega) = \begin{cases} 
0, & t < t_0, \\
51480 \omega^7(t - t_0)^7(1 - \omega(t - t_0))^7, & t_0 \leq t \leq t_0 + 1/\omega, \\
0, & t > t_0 + 1/\omega.
\end{cases}$$
Figure 4.7: VerySmoothBump (left) with $\omega = 0.5$ and $t_0 = 0$. C6SmoothBump (right) with $\omega = 2$ and $t_0 = 0$.

The C6SmoothBump function has six continuous derivatives and integrates to one. A plot of the C6SmoothBump time-function is shown in Figure 4.7.

4.2.14 GaussianWindow

$$g(t, t_0, \omega) = \sin(\omega t)e^{-\frac{(\omega(t-t_0)/N_c)^2}{2}}$$

A plot of the GaussianWindow time-function with $N_c = 5$ is shown in Figure 4.8. Note that $N_c$ is specified with the `ncyc` keyword, which must be given when this time function is used in the `source` command.

4.2.15 Dirac

The Dirac distribution $\delta(t - \tau_0)$ is not a regular time function because it is zero everywhere, except at $t = \tau_0$, where it is unbounded. The integral of $\delta(t)$ is one, and if $f(t)$ is a continuous function,

$$\int f(t)\delta(t - \tau_0) \, dt = f(\tau_0). \quad (4.1)$$

We discretize the Dirac distribution on a grid $t_n = n\Delta_t$, where $\Delta_t > 0$ is the same time step that is used to solve the elastic wave equation. We obtain the discrete time series $d_n$, $n = 0, 1, 2, \ldots$ by imposing moment conditions such that (4.1) is satisfied for the polynomial functions $f(t) = t^q$, $q = 0, 1, \ldots, Q$, in the sense

$$\Delta_t \sum_{n=0}^{\infty} t_n^q d_n = \tau_0^q, \quad q = 0, 1, \ldots, Q.$$ 

This leads to $Q + 1$ conditions for the coefficients $d_n$. The specification of the grid function is made unique by enforcing $d_n = 0$ except at $Q + 1$ consecutive grid points surrounding $t = \tau_0$. Note that $\tau_0$ is not required to coincide with a time step. This procedure is similar to the spatial discretization of point force and moment tensor sources, see [14] for details.
The discretized Dirac distribution triggers all frequencies on the mesh, including completely unresolved and unphysical motions. The numerical solution is therefore meaningless unless it is filtered to remove the unresolved motions. The filtering can either be done after the simulation is completed, or by using the `prefilter` command. The Dirac time function can also be useful for calculating “numerical” Green’s functions.

### 4.3 Discrete time function

The discrete time function uses a quintic (piecewise 5th order polynomial) spline function to interpolate the discrete function values $g_j = g(\tau_j)$, for $j = 1, 2, \ldots, N_s$, where $N_s \geq 7$. The function values are specified on an equidistant grid in time, $\tau_j = t_0 + (j - 1)\delta t$. The step size, $\delta t > 0$, can be chosen independently of the time step that is used to solve the elastic wave equation. The start time, $t_0$, can also have an arbitrary value. The interpolation procedure results in six polynomial coefficients for each interval $\tau_j \leq t < \tau_{j+1}$. The coefficients are chosen such that $g(t)$ becomes four times continuously differentiable.

It is necessary to evaluate the discrete time function throughout the simulation of the elastic wave equation, i.e., for $0 \leq t \leq T$. If the time series starts at $t_0 = \tau_1 > 0$, the discrete time function is evaluated using the polynomial coefficients corresponding to the first interval $[\tau_1, \tau_2]$ for all $0 \leq t < \tau_1$. Similarly, if the last data point in the time series has $\tau_{N_s} < T$, the coefficients of the last interval are used to evaluate the function for $\tau_{N_s} < t \leq T$. To avoid unexpected results due to extrapolation, we recommend specifying the discrete time function for all $t \in [0, T]$. Furthermore, to avoid incompatibilities between the forcing and the homogeneous initial conditions, we also recommend setting $g_j = 0$ for all $\tau_j \leq 0$.

The file format for a discrete time function is described in Section 12.1.
4.4 What is the frequency content in the time function?

Figure 4.9 displays the absolute values of the Fourier transforms of the functions Gaussian, RickerInt, Ricker, and the time derivative of the Brune function. Inspection of the mathematical definitions of the Gaussian and Brune functions shows that the \( \text{freq} \) parameter specifies the angular frequency for these functions, while it specifies the (regular) frequency for the Ricker and RickerInt functions. More generally, the relation between the fundamental frequency \( f_0 \) and the \( \text{freq} \) parameter is given by

\[
 f_0 = \begin{cases} 
 \text{freq}, & \text{for Ricker, RickerInt, VerySmoothBump, C6SmoothBump,} \\
 \frac{\text{freq}}{2\pi}, & \text{for Liu, Brune, BruneSmoothed, Gaussian, GaussianInt, and GaussianWindow.} 
\end{cases}
\]  

The plots in Figure 4.9 were made with frequency parameter \( \text{freq}=0.25 \) for the Ricker and RickerInt functions and frequency parameter \( \text{freq}=1.5 \) for the Gaussian and \( d/dt(\text{Brune}) \) functions. Hence, \( f_0 \approx 0.25 \) for all functions in Figure 4.9. Note that the Fourier transform of the \( d/dt(\text{Brune}) \) function decays much slower than the other functions for high frequencies. This is due to its lack of smoothness at \( t = t_0 \).

It is the highest significant frequency, \( f_{\text{max}} \), that generates the shortest waves and therefore determines how fine the computational grid must be. For practical purposes \( f_{\text{max}} \) can be defined as the frequency where the amplitude of the Fourier transform falls below 5% of its max value. We have

\[
 f_{\text{max}} \approx \begin{cases} 
 2.5f_0, & \text{Ricker, RickerInt, Gaussian time functions,} \\
 3.0f_0, & \text{C6SmoothBump time function,} \\
 4.0f_0, & \text{d/dt(\text{Brune}) time function.} 
\end{cases}
\]
In other words, simulations using the Brune function are hard to resolve on the grid and need a significantly finer grid than the other time functions to give reliable results. The relation between the fundamental frequency \( f_0 \) and the highest significant frequency \( f_{\text{max}} \) in (4.3) is very important. For time functions that not are listed in that formula, it is possible to estimate \( f_{\text{max}} \) with the help of the matlab/octave scripts `fcnplot.m` and `ftfcnplot.m` in the tools directory.

It is important to remember that the estimated highest frequency content in (4.3) assumes that \( t_0 \) is sufficiently large to avoid unresolved transients due to a sudden start. Some time functions are identically zero for \( t < t_0 \). For those functions all \( t_0 \geq 0 \) give reliable results. For time functions that have an exponential tail for \( t < t_0 \), it is important that this tail is sufficiently small for \( t = 0 \). In particular, we recommend

\[
t_0 \geq 6\sigma, \quad \sigma = \begin{cases} 
1/\text{freq}, & \text{Gaussian and GaussianInt,} \\
2\pi/\text{freq}, & \text{Ricker and RickerInt.}
\end{cases}
\]

We remark that inspecting the Fourier transform of a time function only makes sense for functions that tend to zero for large times. Functions such as GaussianInt, Liu, Brune, and BruneSmoothed tend towards unity for large times. Their zero frequency (DC) component therefore grows linearly with the length of the time interval, \( T \). These functions need to be differentiated before applying the Fourier transform.

### 4.5 How to choose the grid size

The most difficult parameter to choose when preparing the input file is probably the grid size, \( h \). It is extremely important to use a grid size that is sufficiently small, because you must resolve the waves that are generated by the source. On the other hand you don’t want to use an unnecessarily small grid size, because both the execution time and the memory requirements increase rapidly when the mesh is refined.

The number of grid points per shortest wavelength, \( P \), is a normalized measure of how well a solution is resolved on the computational grid. Since the shear waves and surface waves have approximately the same wavelength and propagate at approximately the same speed, we can estimate the shortest wavelength by

\[
L_{\text{min}} = \frac{\min C_s}{f_{\text{max}}}. 
\]

Here \( C_s \) is the shear velocity of the material and \( f_{\text{max}} \) is the largest significant frequency in the time function \( g(t) \), as discussed above. Hence the number of grid points per wavelength equals \( L_{\text{min}}/h \), which is given by

\[
P = \frac{L_{\text{min}}}{h} = \frac{\min C_s}{h f_{\text{max}}}. \quad (4.4)
\]

Note that \( h \) needs to be made smaller to maintain the same value of \( P \) if either \( C_s \) is decreased or if the frequency is increased. In formula (4.4), \( \min C_s \) is found from the material properties and \( h \) is determined by the input grid specification. The frequency spectrum of the solution is determined by the frequency spectrum of the time function and \( f_{\text{max}} \) can be estimated from equation (4.3).
4.5.1 Lamb’s problem

The accuracy of the numerical solution, including the implementation of a point force and the reflection properties of the super-grid far field boundary condition, can be tested by solving Lamb’s problem [10]. This problem simulates the motion due to a vertical point force applied on the free surface of a homogeneous elastic half-space.

We have implemented Mooney’s formulas [12] for solving Lamb’s problem. These formulas give an analytical expression for the Green’s function of the vertical component along the free surface, $z = 0$. This Green’s function is convolved with the source time function to give the displacement as function of time. We use a \texttt{C6SmoothBump} time function and the convolution is performed by numerical quadrature using the Quadpack library. This approach allows the displacement to be evaluated to within 12 decimal places. Lamb’s problem is then solved numerically using SW4 and the error in the vertical component is evaluated along the free surface. To evaluate how fine the grid needs to be, we repeat the test for different grid sizes. See the paper by Petersson and Sjogreen [17] for further details.

In this example, the elastic half-space consists of a homogeneous material with shear wave velocity $C_s = 1000$ m/s, compressional wave velocity $C_p = 1000\sqrt{3}$ m/s, and density $\rho = 1500$ kg/m$^3$. The elastic half-space is truncated to the computational domain $(x,y,z) \in [0,10000] \times [0,10000] \times [0,5000]$.

The source is placed on the free surface in the center point of the horizontal plane: $(5000,5000,0)$. The time dependency of the forcing is a “C6SmoothBump” with $\omega = 1$ Hz, $t_0 = 0$ s and magnitude $10^{13}$ N. The above setup is created with the input file shown below, which can be found in \texttt{examples/lamb/seismic1.in}.

```plaintext
grid nx=151 x=10e3 y=10e3 z=5e3
time t=5.0
supergrid gp=60
block vp=1.7320508076e+03 vs=1000 rho=1500
source type=C6SmoothBump x=5e3 y=5e3 z=0 fz=1e13 freq=1 t0=0
rec x=5e3 y=6e3 z=0 file=v1 sacformat=0 usgsformat=1
```

Because the frequency parameter of the \texttt{C6SmoothBump} time function is $f_0 = 1$ Hz and $C_s = 1000$ m/s, the dominant wave length in the solution is about 1,000 meters. Hence the receiver is located approximately one wave length from the source.

It is well known that the error in numerical solutions of wave propagation problems is dominated by phase errors [8]. It is therefore interesting to investigate how the accuracy in the numerical solution depends on the distance between the source and receiver. For this reason, we expand the above calculation to include a second receiver at distance 10,000 meters from the source. This input file is given in \texttt{examples/lamb/seismic2.in}. In this case, we evaluate the solution of Lamb’s problem using the Fortran program \texttt{src/lamb.onepoint.f}. The vertical displacements and errors are shown in Figure [4.10]. The waveforms are smooth and the problem appears to be well resolved. Because of geometric spreading, the amplitude of the exact solution decreases with the distance from the source. At the same time, the amplitude of the error increases. This is because the numerical solution accumulates discretization errors as it propagates through the computational grid.

We show the errors in the vertical displacement in Table [4.1]. The errors are presented in max norm, normalized by the max norm of the exact solution. The frequency parameter in the
Figure 4.10: Lamb’s problem: Vertical displacement at 1,000 (blue) and 10,000 (red) meters from the source. The green and light blue lines show the corresponding errors in the numerical solution with $h = 200/3$, corresponding to $P = 5$ grid points per shortest wave length. The displacement is offset by 1 for the second recording.

C6SmoothBump time function is $f_0 = 1$ Hz. Following (4.3), we estimate the highest significant frequency to be $f_{\text{max}} = 3.0$ Hz. In this case the formula for the number of grid points per wave length (4.4) becomes

$$P = \frac{1000}{3h}.$$ 

Also note how quickly the total number of grid points ($N_{\text{GP}}$) increases when the grid is refined.

The errors are also plotted in Figure 4.11. The error decreases at a rate approaching $O(h^4)$ for the finest grid, indicating that the numerical method and the discretization of the point force are fourth order accurate. For the same grid size, note that the error is about 7.5 times larger at 10 km from the source, compared to 1 km. To get the same accuracy at 10 km from the source, the grid size must be reduced by a factor of $(7.5)^{1/4} \approx 1.65$, i.e., the number of grid points per wave length must be increased by the same factor.

From this experiment we conclude that the accuracy in $SW_4$ depends on the distance between the source and the receiver, which can be normalized by the dominant wave length in the solution.
Table 4.1: Lamb’s problem: Relative max norm errors in the vertical displacement at 1,000 and 10,000 meters from the source. Here, $h$ is the grid size, corresponding to $P$ grid points per shortest wave length. Also, $N_{GP}$ is the total number of grid points in the 3-D grid.

<table>
<thead>
<tr>
<th>$h$</th>
<th>$P$</th>
<th>$|e_1^{(z)}|<em>\infty/|U_1^{(z)}|</em>\infty$</th>
<th>$|e_{10}^{(z)}|<em>\infty/|U</em>{10}^{(z)}|_\infty$</th>
<th>$N_{GP}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>200/3</td>
<td>5</td>
<td>7.17 \cdot 10^{-2}</td>
<td>4.31 \cdot 10^{-1}</td>
<td>3.0 \cdot 10^7</td>
</tr>
<tr>
<td>50</td>
<td>6.7</td>
<td>3.44 \cdot 10^{-2}</td>
<td>2.43 \cdot 10^{-1}</td>
<td>7.3 \cdot 10^7</td>
</tr>
<tr>
<td>100/3</td>
<td>10</td>
<td>9.31 \cdot 10^{-3}</td>
<td>6.99 \cdot 10^{-2}</td>
<td>2.4 \cdot 10^8</td>
</tr>
<tr>
<td>25</td>
<td>13.3</td>
<td>3.25 \cdot 10^{-3}</td>
<td>2.42 \cdot 10^{-2}</td>
<td>5.8 \cdot 10^8</td>
</tr>
</tbody>
</table>

Figure 4.11: Lamb’s problem: Error in max norm as function of the number of points per wave length at 1,000 meters (blue) and 10,000 meters (green) from the source. The dotted lines indicate the asymptotic decay of the error in a fourth order accurate method.
For most practical purposes the accuracy is acceptable when

\[ 6 \leq P \leq 10, \]

but the exact number depends on the required accuracy. The ratio between the compressional and shear velocity, \( C_p/C_s \), can also have a significant influence on the accuracy of the solution, and the number of grid points per wave length must be increased for materials with large \( C_p/C_s \), see [9].

We finally remark that the best way of checking the accuracy in a numerical simulation is to repeat the calculation on a finer mesh and compare the results. Unfortunately, this approach is seldomly used in realistic situations because the computational cost increases too rapidly as the mesh is refined.
Chapter 5

Topography

The topography command in \textit{SW4} is used to specify the shape of the top surface of the computational domain,

\[ z = \tau(x, y). \]

The topography can currently be described in one of four ways: a Gaussian hill (§5.1), by the elevation on a latitude-longitude grid (§5.2), through an \texttt{efile} database (§5.3), or by using a \texttt{rfile} raster file (§5.4).

A curvilinear grid is automatically constructed between the topography surface and a user specified depth \( z = z_{\text{max}} \). If no topography command is present in the input file, the top surface is taken to be the plane \( z = 0 \), and no curvilinear grid is constructed. When the topography surface \( z = \tau(x, y) \) varies between \( \tau_{\text{min}} \leq z \leq \tau_{\text{max}} \) (\( z \) is positive downwards), the grid generation usually works well if

\[ z_{\text{max}} \geq \tau_{\text{max}} + 3(\tau_{\text{max}} - \tau_{\text{min}}), \quad (5.1) \]

Sometimes it is easier to think in terms of elevation, \( e(x, y) = -\tau(x, y) \), because \( e \) is positive above mean sea level. Another way of stating (5.1) is to set the elevation of the bottom grid line in the curvilinear grid to satisfy

\[ e_{\text{grid}} \leq e_{\text{min}} - 3(e_{\text{max}} - e_{\text{min}}), \quad e_{\text{min}} \leq e(x, y) \leq e_{\text{max}}. \quad (5.2) \]

You then set \( z_{\text{max}} = -e_{\text{grid}} \).

After reading the topography, \textit{SW4} prints out the min and max z-coordinates, as well as the specified value of \( z_{\text{max}} \). For example,

***Topography grid: min z = -1.1443e+03, max z = 1.0929e+03, \]
\texttt{top Cartesian z = 8.000000e+03}

Here, \( \tau_{\text{min}} = -1144.3, \tau_{\text{max}} = 1092.9 \) and \( z_{\text{max}} = 8000 \). We have \( \tau_{\text{max}} + 3(\tau_{\text{max}} - \tau_{\text{min}}) = 7804.5 \), which satisfies (5.1). Alternatively, in terms of elevation, \( e_{\text{max}} = 1144.3 \) and \( e_{\text{min}} = -1092.9 \), so \( e_{\text{min}} - 3(e_{\text{max}} - e_{\text{min}}) = -7804.5 \) and \( e_{\text{grid}} = -8000 \), which satisfies (5.2).

Except for the Gaussian hill topography, the topography surface is smoothed by a Jacobi iteration before the curvilinear grid is generated. The purpose of the smoothing is to ensure that the variations in topography can be resolved on the computational grid. By default, 10 iterations are performed and this gives a satisfactory result in many cases. It is possible to change the number of iteration by using the \texttt{smooth} option in the topography command. You can inspect the result of the smoothing by saving the top grid surface in an image file.
Note that the $z$ coordinate (positive downwards) is saved on a grid image file, while the elevation (positive upwards) of the raw (before smoothing) topography is saved on a topo image file (obtained by specifying `mode=topo` instead of `mode=grid`).

### 5.1 Gaussian hill topography

The simplest type of topography is a Gaussian hill, i.e., a topography described by the Gaussian,

$$
\tau(x, y) = Ae^{-((x-x_c)/L_x)^2-((y-y_c)/L_y)^2}.
$$

The user can place one Gaussian hill at a location specified by $(x_c, y_c)$, in the $(x,y)$-plane. Furthermore, the user can adjust the amplitude of the hill, $A$, as well as its spread in the $x$ and $y$-directions, $L_x$ and $L_y$ respectively. The Gaussian hill topography command has the following syntax:

```
topography input=gaussian zmax=7.5 gaussianAmp=2.4 \n    gaussianXc=3.6 gaussianYc=2.4 \n    gaussianLx=0.25 gaussianLy=0.3
```

Note the `zmax` option, which tells SW4 to extend the curvilinear grid to $z = 7.5$. The most common use of the Gaussian hill topography is for testing, see for example the input scripts in `examples/twilight`:

```
gauss-twi-1.in gauss-twi-2.in gauss-twi-3.in
```

### 5.2 Topography grid file

The topography can be given on a regular lattice in geographical (lat, lon), or Cartesian $(x, y)$ coordinates. This approach works well together with the `block`, `pfile`, and `ifile` material commands. It can also be used together with the `rfile` or `efile` commands, but in those cases it is important that the topography is consistent with the material data, because it is given relative to mean sea level ($z = 0$).

To setup the topography, you can give the command

```
topography input=grid file=grenobleCoarse.topo zmax=3000 order=2
```

The file `grenobleCoarse.topo` holds the elevation (in meters) relative to mean sea level and must conform to the simple ASCII text format described in Section 12.4. In the above case, a curvilinear grid is constructed between the topography surface and $z = 3000$, and the `order=2` option specifies a second order polynomial stretching in the curvilinear mapping function. The topography is shown in Figure 5.1.

### 5.3 efile topography

The Etree databases for the San Francisco bay area and Northern California contain topographic information. You can setup the computational grid to follow this topography by using the following command,

```
topography input=efile file=etree.topo order=2
```

The file `etree.topo` contains the topographic data in Etree format and must conform to the ASCII input format. In the above case, a second order polynomial stretching is used in the curvilinear mapping function.
Figure 5.1: Topography in the vicinity of Grenoble, France.

```
topography input=efile etree=USGSBayAreaVM-08.3.0.etree zmax=8e3 order=3
```

Note that the syntax has been changed from WPP. This style of the topography command tells SW4 to read the topography from the specified Etree file. Figure 5.2 (left) shows the resulting elevation of the free surface. The `efile` keyword can also be called `etree`. The `order=3` option specifies a cubic stretching function in the vertical direction. A higher value makes the curvilinear grid smoother near the bottom, but can cause a larger variation in grid size near the top. The `zmax=8e3` option tells SW4 to put the bottom boundary of the curvilinear grid at \( z = 8000 \).

When the computational domain is larger, it is possible to combine the topographic information from the detailed and regional databases, using the following syntax:

```
topography input=efile etree=USGSBayAreaVM.etree xetree=USGSBayAreaVMExt.etree \ 
zmax=8e3 order=3
```

Here the filenames have been abbreviated to improve readability.

### 5.4 rfile topography

The first block in an `rfile` raster file contains the topographic information. You can setup the computational grid to follow this topography by using the following command,

```
topography input=rfile zmax=2e3 order=3 \ 
file=/Users/petersson1/src/sw4/examples/rfile/berkeley.rfile
```
Here, the topography command tells \textit{SW4} to read the topography from the specified \textit{rfile}. Similar to the \textit{efile} topography, the \texttt{order=3} option specifies the type of stretching to use when making the curvilinear grid, and the \texttt{zmax=2e3} option tells \textit{SW4} to put the bottom boundary of the curvilinear grid at \( z = 2000 \). Figure 5.2 (right) shows the elevation of the resulting topography. In this example, the \textit{rfile} holds the same information as the \textit{efile} (shown on the left side of the same figure). The reason the \textit{efile} topography looks rough (pixelated) is that the grid size of the computational grid is \( h = 20 \) m, while the horizontal resolution of both the \textit{rfile} and the \textit{efile} is 100 m. The reason the \textit{rfile} topography is smoother is because bi-linear interpolation is used to define the topography in between the grid points of the \textit{rfile}, while the current query routines for the \textit{efile} only support a piecewise constant representation.

Note that the topography must be described by the same \textit{rfile} as the material model, see Section 6.3 for further information. The format for the \textit{rfile} is described in Section 12.5.
Chapter 6

The material model

In SW4, an isotropic elastic material model is defined by the grid point values of the density ($\rho$), the compressional velocity ($V_p$), and the shear velocity ($V_s$). The material properties can be specified by the block command (§ 6.1), the efile command (§ 6.2), the rfile command (§ 6.3), the pfile command (§ 6.4), the ifile command (§ 6.5), or by a combination of them. If the same computational grid is used for several simulations, the material model can also be obtained through the vimaterial command, assuming that it was initially saved using the volimage command. See § 6.6 for details.

This chapter only discusses isotropic elastic materials. See Chapter 8 for isotropic visco-elastic material models and Section 11.3.10 for anisotropic elastic models.

SW4 requires that the material model is defined for all interior and boundary grid points in the computational grid, as defined by the grid and optionally the topography commands. Note that SW4 also uses two layers of ghost points, just outside the computational domain. If the material model is not defined for ghost points, its properties are extrapolated from the nearest boundary point. In other words, SW4 does not require the material model to be defined at the ghost points, but will use it if it is provided. This is a change from WPP.

The order within the material commands (block, pfile, rfile, efile, and ifile) does matter (unlike all other commands) in that the priority of the material command increases towards the end of the input file. Hence, a material command in the input file can be completely or partially overridden by subsequent material commands.

In the block, pfile, and ifile commands, material properties are assigned based on the depth below the free surface. This means that the internal material model depends on the topography, but the material properties along the free surface will always be the same, independent of the topographic model. For the rfile and efile commands, material properties are defined as functions of the z-coordinate, i.e., relative to mean sea level ($z = 0$). In this case the topography information is embedded in the material description. The rfile command should always be used with topography from the same rfile. If you combine the efile commands with a planar topography, a linear mapping is constructed before the material properties are assigned. The properties at the free surface are thus mapped to the top grid surface ($z = 0$), and the bottom grid surface (with $z = -z_N$) is assigned material properties for elevation $-z_N$. Elevation values at intermediate grid points follow from the linear mapping.

After reading all material commands in the input file and assigning material properties to the computational grid, SW4 outputs general information about the ranges in the material model. For a purely elastic material, the output looks like
It is always a good idea to check that these numbers are reasonable before proceeding with the simulation. In addition, we recommend inspecting the material model along a few image planes.

Before the simulation is started, SW4 checks that density, $V_p$, and $V_s$ are positive at all grid points. In addition, it is verified that the first Lamé parameter satisfies $\lambda > 0$, that is $V_p/V_s > \sqrt{2}$. If either of these conditions is violated, the program stops with an error message. You can obtain more detailed diagnostics by setting `verbose=3` (or higher) in the `fileio` command.

### 6.1 The block command

The block command can be used to specify material properties in rectangular volumes of the computational domain, either with constant values or linear vertical gradients. By combining the block command with the sub-region options we can define a material model composed of three layers:

```
block vp=4000 vs=3500 rho=2000
block vp=6000 vs=3500 rho=2700 z1=15000
block vp=8000 vs=4500 rho=3300 z1=35000 z2=100000
```

In this case the top layer has a thickness of 15 km, the middle layer 20 km and the lower layer 65 km. Because these block commands do not specify horizontal coordinates, the values extend to the grid boundaries in both horizontal directions. To add a box shaped inclusion of a new material we could add the following line

```
block vp=3000 vs=2000 rho=1000 \
  x1=4000 x2=8000 y1=3000 y2=7000 z1=10000 z2=70000
```

An image of $V_p$ in the plane $x = 50,000$ is shown in Figure 6.1 (left).

Several block commands can be combined to generate more complicated material models, for example

```
block vp=8000 vs=4500 rho=3300 vpgrad=-0.01
block vp=3000 vs=2000 rho=1000 \
  x1=1e4 x2=9e4 y1=1e4 y2=9e4 z1=1e4 z2=9e4 vpgrad=0.02
block vp=4000 vs=2500 rho=2000 \
  x1=15e3 x2=85e3 y1=15e3 y2=85e3 z1=15e3 z2=85e3
block vp=6000 vs=3500 rho=2700 \
  x1=15e3 x2=85e3 y1=15e3 y2=85e3 z1=45e3 z2=55e3
block vp=6000 vs=3500 rho=2700 \
  x1=15e3 x2=85e3 z1=15e3 z2=85e3 y1=38e3 y2=45e3
```

This material is displayed on the right side of Figure 6.1.
6.2 The efile command

The efile command is used to obtain material properties from an etree database file. Etree data bases use an oct-tree data structure, which allows material properties to be represented with finer spatial resolution near the surface. Topography and bathymetry information is included in the data base. Note that the same etree database file can be used independently of the grid size. We currently have access to an etree database file for Northern California and the extended San Francisco bay area. The geographical extent of the etree model is given in Table 6.1, which also is shown on a map in Figure 6.2. This model was developed by the USGS and can currently be downloaded from http://earthquake.usgs.gov/data/3dgeoologic. Be aware that the files are large (8 + 6 Gbyte) and can take a very long time to download.

In the etree database, material properties are stored as functions of geographical coordinates (latitude, longitude, elevation), and SW4 must determine the geographical coordinates before it obtains the material properties from the database. Depending on the options of the grid command, SW4 either uses formulas (3.6)-(3.7), or the proj.4 library, to compute the geographic mapping. Internally to SW4, the cencalvm software library is used to query the etree database, which in turn relies on additional libraries. The description in this section assumes that SW4 has been configured to use the efile command, see [19] for details.
It is important to note the bounds of the geographical region in the database. Assuming the computational domain is contained within the bounds of the database, it is easy to set up the material model in the input file:

```
grid x=100e3 y=100e3 z=40e3 lat=38.0 lon=-121.8 az=144 h=1000
efile etree=/p/lscratchd/andersp/USGSBayAreaVM-08.3.0.etree
```

To verify that the computational domain is inside the etree database, we recommend checking the geographical coordinates on a map during the construction of the input file. We often use the Google Earth program for this purpose. In the case when the computational domain is larger than the region covered by the efile, a block command can be used to assign material properties to grid points outside of the efile region:

```
grid x=300000 y=300000 z=60000 lat=38 lon=-121.5 az=135 nx=100
block vp=8000 vs=4000 rho=1000 rhograd=0.5
efile etree=/p/lscratchd/andersp/USGSBayAreaVM-08.3.0.etree
```

However, sharp jumps in material properties can lead to significant scattering of seismic waves. In some cases, better results can be obtained by reducing the size of the computational domain to match the extent of the etree region.

To enable use of the extended SF model, the extended etree file must also be downloaded and then added to the efile command line (file names have been shortened for improved readability):

```
efile etree=USGSBayAreaVM.etree xetree=USGSBayAreaVMExt.etree
```
6.3 The rfile command

A material raster file (rfile) can be used for storing material properties and topography/bathymetry. It uses a binary, block-structured, data format that allows material properties to be represented with finer spatial resolution near the surface of the earth. Topography/bathymetry information is described by the first block of the raster file. The grid sizes in the rfile are independent of the grid size in the computational grid. The rfile parser in SW4 attempts to determine the byte ordering and automatically swap the bytes if it encounters little-endian ordering on a big-endian machine, or vice versa.

There are two main reasons why the raster file format is ideally suited for large material models with heterogeneous properties. First, the data on an rfile is organized to allow each core to only read the relevant part of the file, thus saving memory. Secondly, the rfile format can be read in parallel (if a parallel file system is available). In this approach, only a subset of the cores (processors) interact with the file system, and the data is propagated to all other cores using calls to the MPI library. The parallel reading capability of the rfile format has been shown to scale well on up to 131,072 cores. In practice, it is the most important advantage over the efile format and allows material models of the same complexity to be read significantly faster.

Material properties in the rfile are stored internally on a block-structured grid, and relies on the proj.4 library to map between Cartesian and geographic coordinates. The rfile command can therefore only be used if SW4 has been linked with the proj.4 library. See [19] for instructions.

It is important to note the bounds of the geographical region in the material model. Assuming that the computational domain is contained within those bounds, and the azimuth of the grid agrees with the rfile, the following command sets up the material model:

```
grid x=12e3 y=12e3 z=5e3 nx=601 lat=37.93 lon=-122.25 az=143.6380 proj=tmerc \
    datum=NAD83 lon_p=-123.0 lat_p=35.0 scale=0.9996
rfile filename=USGSBayAreaVM-08.3.0.rfile directory=/Users/petersson1
```

To verify that the computational domain is inside the rfile, we recommend checking the geographical coordinates on a map during the construction of the input file. We often use the Google Earth program for this purpose. The restriction that the azimuth of the computational grid must agree with the azimuth of the rfile is made to simplify the parallel reading algorithm.

The topography and material properties are evaluated on the computational grid using linear interpolation from the underlying grid in the rfile format. When the computational grid is finer than the grid in the rfile, the interpolation leads to some smoothing, see Figure 6.3. In this case, the rfile was generated by copying the topography and material properties from the efile, using the same resolution in both file formats. A vertical cross-section of the same model is shown in Figure 6.4. The query routines for the efile format define constant material properties in each cell, while the reader of a rfile uses linear interpolation to define properties in between the grid point values. Thus, material discontinuities (such as the vertical fault line) are maintained by the efile command, but become slightly smeared out by the rfile command. On the other hand, the efile interpretation of the material properties on either side of the fault line leads to stair-stepping artifacts, while the linear interpolation of the rfile command results in a much smoother model.
Figure 6.3: The compressional wave speed \((C_P)\) along the topography using the \texttt{rfile} (left) and the \texttt{efile} (right) commands. Here, the computational grid has size \(h = 20\) m, while the horizontal grid sizes of the \texttt{rfile} and \texttt{efile} are both \(hh = 100\) m. Notice the stair-stepping in the right figure.

Figure 6.4: The compressional wave speed in a vertical cross-section, using the \texttt{rfile} (left) and the \texttt{efile} (right) commands. Here, the computational grid has size \(h = 20\) m. Near the surface, the grid sizes of the \texttt{rfile} and \texttt{efile} are both \(hh = 100\) m in the horizontal direction and \(hv = 25\) m in the vertical direction. Below \(z = 400\), the horizontal and vertical grid sizes are doubled.
6.4 The pfile command

The pfile command can be used to assign material properties based on depth profiles. A pfile contains the values of the model features (P-velocity, S-velocity, density, and optionally the attenuation factors $Q_P$ and $Q_S$) as function of depth at points on a regular lattice covering the horizontal extent of the computational domain. The points on the lattice are either defined by their latitude and longitude coordinates, or by the $x$ and $y$-coordinates. The number of grid points in the depth direction needs to be the same for all profiles, but the grid spacing does not need to be uniform and can also be different for each profile. Material discontinuities can be represented by two material values for the same depth value. Material layers, which only occur in a subset of the profiles, can be tapered to have zero thickness in the remaining profiles. This is handled by introducing multiple data points with the same depth and material values in a profile.

The lattice of the pfile does not need to have any relation to the computational mesh used in SW4 and is often much coarser. The material properties in the computational mesh are assigned values using Gaussian averaging between the nearest $N_G \times N_G$ profiles in the latitude-longitude plane and linear interpolation in the depth direction. Let the grid point have longitude $\theta$, latitude $\phi$ and depth $d$. Material properties are first linearly interpolated in the depth direction along each profile and then averaged in the latitude-longitude plane. The number of points in the Gaussian averaging, $N_G$, is assigned by the user in the pfile command. For example, the following line in the input file makes SW4 read a pfile named material.ppmod:

```
pfile filename=material.ppmod vsmin=1000 vpmin=1732 smoothingsize=4
```

The optional $vsmin$ and $vpmin$ keywords are used to assign minimum threshold values for the P- and S-velocities, respectively. Here, smoothingsize=4 means that $N_G = 4$ in the Gaussian averaging. A larger value of $N_G$ ($\geq 5$) is particularly useful to avoid staircasing imprints when the computational grid is much finer than the pfile lattice, see Figure 6.5. The smoothingsize keyword can be assigned any number greater than or equal to one.

When $N_G$ is odd, the Gaussian averaging starts by finding the closest grid point on the latitude-longitude lattice, $(\phi_i, \theta_j)$. The material property $c (\rho, V_p, V_s, Q_s, Q_p)$ is assigned by the formula

$$c(\phi, \theta) = \sum_{m=-W}^{i+W} \sum_{n=-W}^{j+W} \frac{c_{m,n} \omega_{m,n}}{\sum_{m=-W}^{i+W} \sum_{n=-W}^{j+W} \omega_{m,n}}, \quad W = \frac{N_G - 1}{2},$$

(6.1)

where the weights are given by

$$\omega_{m,n} = e^{-[(\phi_m-\phi)^2+(\theta_n-\theta)^2]/\alpha^2}, \quad \alpha = \frac{N_G \Delta}{2\sqrt{-\log 10^{-6}}},$$

where the parameter $\Delta$ is given in header of the pfile (on line 2). It should approximately equal the grid size in latitude and longitude (which do not have to be equal). This choice of $\alpha$ makes the weights $\omega_{m,n} < 10^{-6}$ for points that are further from $(\phi_m, \theta_n)$ than $N_G \Delta/2$, which justifies the truncation of the series in (6.1). A similar procedure is used for even values of $N_G$, but in this case the averaging formula (6.1) is centered around the nearest cell center on the latitude-longitude lattice.

By default the pfile data are assumed to be given in latitude-longitude coordinates. It is also possible to read pfiles where the data is given on a Cartesian grid in $(x, y)$-coordinates. To indicate that the pfile contains data on a Cartesian grid, use the style option as in the following example:
Figure 6.5: The smoothing size parameter can be used to average out imprinting from the horizontal lattice in a coarse pile material model. Here we show $V_P$ in the plane $x = 3,500$ as function of $y$ and $-z$. In this case, smoothing size $= 1$ in the top left, smoothing size $= 3$ in the top right, smoothing size $= 4$ in the bottom left, and smoothing size $= 5$ in the bottom right plot.
Data files for the pfile command are written in an ASCII text format. See Section 12.3 for a description of both the latitude-longitude pfile and the Cartesian pfile grid formats.

6.5 The ifile command

The ifile command reads a file holding the depth to material interface surfaces. The material properties between each pair of material surfaces must be defined by the material command. The depth must be non-negative. Zero depth corresponds to the topography. Material surfaces are specified on a regular lattice in geographic coordinates. The unit for depth is meters, while latitude and longitude are in degrees. The ifile command may be combined with other material specifications and it is not necessary that the lattice in geographical coordinates covers the horizontal extent of the computational domain.

Let \( N_{mat} \geq 1 \) material surfaces be known at longitudes
\[ \phi_i, \quad i = 1, 2, \ldots, N_{lon}, \]
and latitudes
\[ \theta_j, \quad j = 1, 2, \ldots, N_{lat}, \]
Note that the latitudes and the longitudes must either be strictly increasing or strictly decreasing, but the step size may vary. Also note that the lattice points are independent of those in the topography command.

The material surfaces should be given on the regular lattice
\[ d_{q,i,j} = \text{depth to surface number } q \text{ at longitude } \phi_i, \text{ latitude } \theta_j. \]
The material surfaces correspond to material properties in the following way. At longitude \( \phi_i \), latitude \( \theta_j \) material number 1 (as defined by the material command) occupies depths \( 0 \leq d \leq d_{1,i,j} \). Material number 2 occupies depths \( d_{1,i,j} \leq d \leq d_{2,i,j} \), and so on. If \( d_{1,i,j} = 0 \), material number 1 is not used. Similarly, material number \( k > 1 \) is not used if \( d_{k-1,i,j} = d_{k,i,j} \). Material properties are only defined for depths down to the last surface, i.e.,
\[ 0 \leq d \leq d_{N_{mat}, i,j}. \]
If the computational domain extends below the last material surface, it is necessary to use other commands to define the material properties in those regions.

The material properties can have a constant, linear, quadratic, and square root dependence of depth. For example, the most general dependence for density is
\[ \rho(d) = \rho_{k,0} + \rho_{k,1}d + \rho_{k,2}d^2 + \rho_{k,1/2}\sqrt{d}, \quad d_{k-1,i,j} \leq d < d_{k,i,j}. \]
Bi-linear interpolation in longitude and latitude is used to define the material surfaces in between the data points. Note that only constant values are supported for the quality factors (\( Q_P \) and \( Q_S \)) within each material.

The ifile file format is described in Section 12.4.
6.6 The \texttt{vimaterial} command

The \texttt{vimaterial} command is intended to speed up the reading of large material models for cases when many simulations use the same computational grid. In order for this command to work, the \texttt{grid} and \texttt{topography} commands must be identical between runs. An initial run must construct the material model using the commands described in the previous sections (block, ef ile, pf ile, if ile). The initial run must also save the material model using the \texttt{volimage} command. Each of these commands saves one scalar property per file, so three files must be saved for an elastic material. In subsequent simulations the \texttt{vimaterial} command can replace the initial material model. This approach works the best on machines with a fast (parallel) file system. Also note that the \texttt{volimage} files can easily become very large, because the material properties are saved at each grid point.

For example, if the material model and the topography are given in the etree format, the initial run reads this information and saves it on 5 separate \texttt{volimage} files,

\begin{verbatim}
fileio pfs=1 path=MaterialModel
grid x=275e3 y=120e3 z=40e3 h=400 lon=-122.688 lat=39.009 az=142.25
topography input=ef ile file=ef ile/USGSBayAreaVM-08.3.0.etree zmax=8e3 order=3
attenuation nmech=3 phasefreq=1 maxfreq=10

ef ile query=MAXRES etree=ef ile/USGSBayAreaVM-08.3.0.etree

volimage mode=rho file=Lp cycle=0 precision=double
volimage mode=mu file=Lp cycle=0 precision=double
volimage mode=lambda file=Lp cycle=0 precision=double
volimage mode=qp file=Lp cycle=0 precision=double
volimage mode=qs file=Lp cycle=0 precision=double
\end{verbatim}

The \texttt{volimage} files are written in the \texttt{MaterialModel} subdirectory, as specified by the \texttt{path} option in the \texttt{fileio} command.

Subsequent runs, that use identical grid and topography commands, can read the material model directly from the \texttt{volimage} files,

\begin{verbatim}
fileio pfs=1 path=OtherRun
grid x=275e3 y=120e3 z=40e3 h=400 lon=-122.688 lat=39.009 az=142.25
topography input=ef ile file=ef ile/USGSBayAreaVM-08.3.0.etree zmax=8e3 order=3
attenuation nmech=3 phasefreq=1 maxfreq=10

vimaterial path=MaterialModel rho=Lp.cycle=0.rho.3Dimg mu=Lp.cycle=0.mu.3Dimg \ lambda=Lp.cycle=0.lambda.3Dimg qs=Lp.cycle=0.qs.3Dimg \ qp=Lp.cycle=0.qp.3Dimg
\end{verbatim}

Note that a path can be specified in the \texttt{vimaterial} command, and that the \texttt{fileio} command now writes the results to a different directory.
Chapter 7

Mesh refinement

The refinement command in $SW_4$ enables the user to locally refine the computational mesh in areas where finer resolution is needed, i.e., where the wave speed is small. In order to maintain a constant resolution in terms of the number of grid points per wavelength for a given frequency (see Equation (4.4)), the grid size should be adjusted such that the ratio $V_s/h$ becomes approximately constant over the computational domain. In $SW_4$, we use a composite grid approach consisting of a set of structured component grids with hanging nodes on the grid refinement interfaces. This allows the grid resolution to follow the main variations in wave speed, and gives ideal wave propagation properties in each component grid. To assure stability of the numerical scheme, an energy conserving coupling approach is used to couple the solution across grid refinement interfaces.

When using mesh refinement, the extent of the computational domain is determined by the grid command, which also specifies the grid size in the coarsest component grid,

```plaintext
grid h=2000 x=40000 y=40000 z=40000
```

The two refinement commands

```plaintext
refinement zmax=23000
refinement zmax=6000
```

specify two mesh refinement interfaces: $z_1 = 23000$, and $z_2 = 6000$. As a result, the composite grid contains three component grids, where the coarsest component has grid size $h = 2000$ and covers the bottom of the computational domain: $z_1 \leq z \leq 40000$. Next refinement grid has half the grid size ($h = 1000$) and covers $z_1 \leq z \leq z_2$. The grid size in the third component is another factor of two smaller ($h = 500$) and covers the top of the computational domain: $z_2 \leq z \leq 0$. The composite grid is shown in Figure [7.1], where the grid is plotted in the vertical $x = 20000$ plane. Note that refinement grids are aligned in the sense that every second grid point coincides with a grid point in the next coarser grid.

The summation by parts finite difference boundary stencil is used at the interfaces between the component grids. This stencil is applied at six points near the boundary, and is eight points wide, which leads to a minimum requirement on the number of grid points in the $z$-direction. On component grids where both the upper and lower boundaries are interfaces to another grid, or has a free surface boundary condition, there has to be at least 12 grid points in the $z$-direction. On grids where only one of the upper or lower boundaries is an interface to another grid, or has a free surface boundary condition, eight grid points are required in the $z$-direction. $SW_4$ will stop with an error message if the number of grid points in the $z$-direction is below either of these limits.
Figure 7.1: Composite grid with two mesh refinement interfaces and three Cartesian component grids.

Normally, the eight grid point limit would only be tested at the bottom of the computational domain, which by default uses a supergrid damping layer. Note that the supergrid layer is 30 grid points thick (by default), and must be fully contained in the bottom grid.

Mesh refinement can also be used together with topography. Here we use an example from a simulation of the 2007 Alum Rock earthquake. The composite grid is setup with the commands

```plaintext
grid x=100e3 y=100e3 z=40e3 lat=38.0 lon=-121.8 az=143.638 h=2000
refinement zmax=23e3
refinement zmax=12e3
topography input=rfile order=2 file=USGSBayAreaVM-08.3.0.rfile
rfile filename=USGSBayAreaVM-08.3.0.rfile
```

Here the base grid, which always is Cartesian, has grid size $h = 2000$ and covers $23000 \leq z \leq 40000$. Next Cartesian grid has half the grid size ($h = 1000$) and covers $12000 \leq z \leq 23000$. The grid size in the finest Cartesian component grid is reduced by another factor of two, which gives $h = 500$. This component extends to the bottom of the curvilinear grid, i.e., $12000 \leq z \leq 6000$. The vertical extent of the curvilinear grid is specified by the $z_{\text{max}}=6000$ option in the topography command, i.e., the curvilinear grid covers the domain between $z = 6000$ and the topography surface, $z = \tau(x,y)$. In the horizontal directions, the grid size in the curvilinear grid is the same as in the finest Cartesian grid. The number of grid points in the vertical direction is choosen such that the average vertical grid size is the same as the grid size in the horizontal directions. A portion of the computational grid is shown in the vertical cross section $x = 50000$, see Figure 7.2.

After constructing the computational grid, $SW4$ outputs information about the number of grid points in each component grid. For the above example, we get
Figure 7.2: A composite grid with two mesh refinement interfaces and topography. In this case there are three Cartesian components and one curvilinear grid following a non-planar topography.

Global grid sizes (without ghost points)

<table>
<thead>
<tr>
<th>Grid</th>
<th>h</th>
<th>Nx</th>
<th>Ny</th>
<th>Nz</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2000</td>
<td>51</td>
<td>51</td>
<td>10</td>
<td>26010</td>
</tr>
<tr>
<td>1</td>
<td>1000</td>
<td>101</td>
<td>101</td>
<td>12</td>
<td>122412</td>
</tr>
<tr>
<td>2</td>
<td>500</td>
<td>201</td>
<td>201</td>
<td>13</td>
<td>525213</td>
</tr>
<tr>
<td>3</td>
<td>500</td>
<td>201</td>
<td>201</td>
<td>12</td>
<td>484812</td>
</tr>
</tbody>
</table>

Total number of grid points (without ghost points): 1.15845e+06

Note that most grid points are in grids number 2 and 3, with grid size $h = 500$. Further down in the output file, $SW_4$, provides information about the resolution in terms of grid points per wave length:

***** PPW = minVs/h/maxFrequency *******

g=0, h=2.000000e+03, minVs/h=1.81531 (Cartesian)
g=1, h=1.000000e+03, minVs/h=3.29057 (Cartesian)
g=2, h=5.000000e+02, minVs/h=5.52953 (Cartesian)
g=3, h=5.000000e+02, minVs/h=0.16 (curvilinear)

As is common in seismic applications, the material velocities are the lowest near the free surface, i.e., in grid number 3 in this case. From this information, we can estimate the highest frequency that can be reliably propagated on this mesh. To get $P = 8$ grid points per shortest wave length, we can use a source time function with maximum frequency

$$f_{max} = \min \frac{V_S}{hP} = \frac{0.16}{8} = 0.02 \text{ Hz}.$$
In this case the grid was made very coarse to make Figure 7.2 clearer. The grid size need to be decreased for practical computations.
Chapter 8

Attenuation

8.1 Viscoelastic modeling

SW4 implements a linear viscoelastic material model by superimposing \( n \) standard linear solid (SLS) mechanisms, leading to the governing equations

\[
\rho \frac{\partial^2 u}{\partial t^2} = L(\lambda_0, \mu_0) u - \sum_{\nu=1}^{n} L(\lambda_\nu, \mu_\nu) \bar{u}^{(\nu)} + F, \quad x \in \Omega, \ t \geq 0, \tag{8.1}
\]

where the spatial operator is

\[
L(\lambda, \mu) u =: \nabla(\lambda(\nabla \cdot u)) + \nabla \cdot (2\mu D(u)), \quad D(u) = \frac{1}{2} (\nabla u + \nabla u^T). \tag{8.2}
\]

The memory variables, \( \bar{u}^{(\nu)} \), in (8.1) are governed by the differential equations

\[
\frac{1}{\omega_\nu} \frac{\partial \bar{u}^{(\nu)}}{\partial t} + \bar{u}^{(\nu)} = u, \quad x \in \Omega, \ t \geq 0, \tag{8.3}
\]

for \( \nu = 1, 2, \ldots, n \), where \( \omega_\nu > 0 \) are the relaxation frequencies. For more details on visco-elastic modeling and the numerical method used by SW4, we refer to the paper by Petersson and Sjogreen [16] and the references therein.

There are three components in each of the vector variables \( u \) and \( \bar{u}^{(\nu)} \), \( \nu = 1, 2, \ldots, n \), resulting in \( 3 + 3n \) differential equations for as many dependent variables. Hence, compared to the purely elastic case, visco-elastic modeling will require more memory and more CPU-time.

The material parameters \( \mu_\nu \) and \( \lambda_\nu \), as well as the relaxation frequencies \( \omega_\nu \), are determined by Emmerich and Korn’s least-squares procedure. In this approach, the material parameters are selected such that the quality factors \( Q_S \) and \( Q_P \) become close to constant over a frequency range

\[
\omega_{\text{min}} \leq \omega \leq \omega_{\text{max}}.
\]

Because the computational cost of viscoelastic modeling increases with the number of mechanisms, \( n \), it is desirable to use the smallest value of \( n \) that gives acceptable accuracy in the approximation of \( Q(\omega) \). In Figure 8.1, we present \( Q(\omega) \) when the material coefficients are chosen to approximate \( Q = 100 \) in the frequency band \( \omega \in [1, 100] \). Clearly, \( n = 2 \) provides inadequate modeling of a constant \( Q \) over two decades in frequency, but \( n = 3 \) gives a much better approximation. Increasing
Figure 8.1: Actual quality factor $Q(\omega)$ approximating $Q_0 = 100$ in the frequency band $\tilde{\omega} \in [1, 100]$, for different numbers of viscoelastic mechanisms.

$n$ further only leads to small improvements. It is interesting to note that in all models, $Q(\omega)$ grows rapidly for $\omega > \omega_{\text{max}}$. Hence the viscoelastic model does not provide significant damping of higher (poorly resolved) frequencies in the numerical solution, and does not act as an artificial dissipation.

Wave propagation in visco-elastic materials is dispersive, i.e., the phase velocity of a wave depends on its frequency. Figure 8.2 illustrates that the frequency dependence on the phase velocity becomes more pronounced when $Q$ gets smaller. Also note that the phase velocity grows approximately linearly on a logarithmic scale in $\omega$, throughout the frequency band $[\omega_{\text{min}}, \omega_{\text{max}}]$. Outside this band, the phase velocity tends to constant values. Due to the dispersive nature of visco-elastic materials, it is necessary to specify the reference frequency, $\omega_r$, at which the phase velocities are specified.

In SW4, visco-elastic modeling is enabled by the `attenuation` command. For example, the command

```
attenuation nmech=3 phasefreq=2.5 maxfreq=10
```

enables visco-elastic modeling with three SLS mechanisms, tuned for the max frequency $f_{\text{max}} = 10$ Hz, such that the phase velocities are valid at the reference frequency $f_r = 2.5$ Hz. For simplicity, the lower frequency in the modeling is always two orders of magnitude smaller than the max frequency,

$$f_{\text{min}} = \frac{f_{\text{max}}}{100}.$$  

Instead of using `maxfreq`, the upper frequency limit can alternatively be specified through the `minppw` option. In this case the material model is first evaluated to find $\min V_S/h$. The upper
Figure 8.2: Relative phase velocity over the frequency band $\omega \in [1, 100]$. Here, $n = 3$, and the different colors correspond to different values of $Q$.

The frequency limit is then calculated through the relation $P = \min V_s/(hf)$, i.e.,

$$f_{max} = \frac{1}{P_{min}} \min \frac{V_s}{h}, \quad P_{min} = \text{minppw}.$$  

The syntax for using $\text{minppw}$ is given by

```
attenuation nmech=3 phasefreq=2.5 minppw=5
```

This procedure results in the same visco-elastic model as by replacing the $\text{minppw}$ keyword by $\text{maxfreq}=f_{max}$.

After the input file has been parsed, $SW4$ outputs basic information about the attenuation modeling:

```plaintext
*** Attenuation parameters calculated for 3 mechanisms, 
max_freq=2.000000e+00 [Hz], min_freq=2.000000e-02 [Hz], 
velo_freq=1.000000e+00 [Hz]
```

Note that $\text{max_freq} = \omega_{max}/(2\pi)$, etc.
Chapter 9

Output options

9.1 Setting the output directory

The fileio command is used to specify by which method and in which directory SW4 should write its output files. If the specified directory does not exist, SW4 attempts to create it for you. The fileio command may also be used to set the level of diagnostic messages (verbose) and how often the time step information is printed. For example, the command

```
fileio path=sw4_dir verbose=1 printcycle=10
```

causes all output files to be written to the directory "./sw4_dir". The path may be absolute or relative to the working directory. The `verbose=1` option enables some extra diagnostic messages to be printed to standard output. The default value is 0. A higher value gives more details, but values exceeding 2 give a lot of information and are mostly intended for debugging purposes. The `printcycle=10` instructs SW4 to output time step information every 10 time steps, instead of the default, which is every 1000 time steps.

Serial and Parallel file systems  Some parallel machines have a dedicated parallel file system that allows many processors to simultaneously write to the same file. These file systems are often mounted under a particular sub-directory. By default, SW4 assumes that the file system is serial, which means that only one processor is allowed to write to the same file at the same time. If you have access to a parallel file system, the I/O performance of SW4 can sometimes be significantly improved by allowing several processors to simultaneously write to the same file. You enable this feature by using the `pfs=1` option,

```
fileio pfs=1 nwriters=16 path=/p/1scratcha/my_output_directory
```

Note that the parallel file systems is often only accessible from certain directories. Setting `pfs=1` without re-directing the output to such a directory may cause SW4 to either crash or hang. The number of processes (cores) that write to disk can be changed with the `nwriters` option. By default, `nwriters=8`. There is a trade-off when selecting `nwriters`. If `nwriters` is too small, SW4 will not make optimal use of the parallel file system, and some extra overhead is incurred by passing parts of the data to the writing processors. If `nwriters` is too large, the I/O operation is chopped up in too many small pieces, which will reduce the efficiency. To obtain good I/O performance, the striping of the parallel file system needs be set appropriately. In our experience `nwriters` should
be at least as large as the stripe count. On the Lustre file system, used at Livermore Computing, the lfs command can be used to set the striping. The striping is set on directories, not on files. For example to set the stripe count to 30 on the directory /p/lscratchd/user/sw4runs, give the command

```
lfs setstripe -s 8M -c 30 /p/lscratchd/user/sw4runs
```

This also sets the stripe size to 8 M byte. The stripe count, 30, will apply to all files that are subsequently written to the directory. Increasing the stripe count from the default value (currently 2 at LC) to somewhere around 30–60 can make a big difference when writing large 3D data files, such as produced by the volimage command. If only image slices and receiver data are output, the stripe count is less significant.

### 9.2 Time-history at a receiver station: the rec (or sac) command

**SW4** can save the time-history of the solution at a receiver station that is located anywhere in the computational domain. The basic command looks like this:

```
rec x=100e3 y=50e3 z=0 file=sta1
```

For backwards compatibility with **WPP**, the rec command can also be called sac. The above command makes **SW4** save the three components of the solution at the grid point. The solution is saved at the grid point which is the closest to the specified (x, y, z) location.

By default, **SW4** saves the data using the binary Seismic Analysis Code (SAC) format, see [7]. Since each SAC file contains one component of the solution, the default rec command results in three files:

```
sta1.x sta1.y sta1.z
```

The x,y,z files hold the corresponding solution component. Note that the orientation of the z-component is positive downwards.

The location of the receiver station can alternatively be given in geographical (latitude, longitude, depth) coordinates. Information about the event location, date, time, and station name is saved in the header of the SAC file. The event location is taken as the hypocenter, i.e., the location of the source with the earliest initiation time. The file name is used as the default station name, but can be modified with the sta option. The date and time are by default set to be the starting time of the simulation. That datum can be changed by using the utcstart option in the time command,

```
time t=10 utcstart=01/04/2012:17:34:45.2343
```

```
rec lat=38.25 lon=-122.20 depth=0 file=sta1 sta=EKM
```

Note that the depth option specifies the depth of the receiver relative to the topography. To place a receiver at elevation e relative to mean sea level (e is negative below sea level) you use the option z=-e. Station that are placed above the topography will be ignored and do not generate any data.

By default, SAC files are written to disk every 1000 time steps, and at the end of the simulation. We can change this frequency by using the writeEvery option. For example, to write the SAC file every 100 time steps, you would say
rec lat=38.25 lon=-122.20 depth=0 file=sta1 writeEvery=100

By default, SW4 outputs the three components of the solution $u(x, t) = (u_x, u_y, u_z)^T$ at each time step. Here, this quantity is called the displacement. However, it is important to note that physical meaning of the solution depends on the source time function. For example, if the displacement corresponds to a GaussianInt source time function, the solution would hold the corresponding velocity if the time function was changed to a Gaussian.

The components that are saved by the rec command can be rotated to the East, North, and vertical (positive up) directions by using the nsew option,

rec lat=38.25 lon=-122.20 depth=0 file=sta1 nsew=1

Here, the angle between North and the $x$-axis is determined by the azimuth (az=...) option in the grid command:

grid x=100e3 y=50e3 z=30e3 lat=37.5 lon=-122.0 az=135

By default, the rec command outputs the three components of the solution (the displacement). The rec command can also output the time derivative of the solution (the velocity),

rec lat=38.25 lon=-122.20 depth=0 file=sta1 variables=velocity nsew=1

As indicated here, the variables option can be combined with nsew. To remind the user of what quantities are saved in a SAC file, we modify the file name extensions according to the following table:

<table>
<thead>
<tr>
<th>variables</th>
<th>nsew=0</th>
<th>nsew=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>displacement</td>
<td>.x, .y, .z</td>
<td>.e, .n, .u</td>
</tr>
<tr>
<td>velocity</td>
<td>.xv, .yv, .zv</td>
<td>.ev, .nv, .uv</td>
</tr>
</tbody>
</table>

It is also possible to save the divergence, curl, or strain of the solution. The variables option governs this behavior. For example,

rec lat=38.25 lon=-122.20 depth=0 file=sta1 variables=div

outputs a single file named sta1.div containing the divergence of the solution, which is independent of the orientation of the components. Similarly,

rec lat=38.25 lon=-122.20 depth=0 file=sta1 variables=curl

outputs the Cartesian components of the curl in the files sta1.curlx, sta1.cury, and sta1.curlz. Furthermore, SW4 can output the components of the symmetric strain tensor,

rec lat=38.25 lon=-122.20 depth=0 file=sta1 variables=strains

In this case, six files are generated: sta1.xx, sta1.yy, sta1.zz, sta1.xy, sta1.xz, and sta1.yz. It is also possible to output the non-symmetric displacement gradient by,

rec lat=38.25 lon=-122.20 depth=0 file=sta1 variables=displacementgradient

which outputs nine components named sta1.duxdx to sta1.duzdz for the components $\partial u(x)/\partial x$ to $\partial u(z)/\partial z$.

For the curl, the strain, and the displacement gradient only nsew=0 has been implemented. That is, we always output the Cartesian components of these quantities.
9.2.1 The ASCII text format

SW4 can also output receiver time-histories on an ASCII text format,

```
rec lat=38.25 lon=-122.20 depth=0 file=sta1 sacformat=0 usgsformat=1
```

The ASCII text file holds all components (usually three, but only one when variables=div, and six when variables=strains) in a single file named sta1.txt. When the usgsformat=1 option is used, the file gets extension .txt independently of the nsew and variables options. Instead, the header of the file is modified to reflect its content. Note that you need to give the sacformat=0 option unless you want the solution to be output in both formats.

9.2.2 Notes on the rec command

- The files are treated in the same way on parallel and serial file systems, because the data for each recording station originates from one processor (core) and is always written by that processor only.
- The binary SAC format is described in Section 12.6.
- The ASCII text format is very simple and is outlined in its header.
- The binary SAC files can be read by the SAC program. We also provide a matlab/octave script in tools/readsac.m.
- The ASCII text file format can be read by the Matlab/Octave script in tools/readusgs.m.

9.3 2-D cross-sectional data: the image command

The image command saves two-dimensional horizontal or vertical cross-sectional data at a specified time level. It can be used for visualizing the solution, making the images for a movie, or checking material properties. Each image file contains a scalar field as function of the spatial coordinates in the cross-sectional plane. The scalar field can be either a component of the solution, a derived quantity of the solution, a material property, or a grid coordinate. All in all, SW4 can output twenty-nine different image quantities, plus six additional image types for testing. See Section 11.5.2 for details.

The cross-sectional plane is specified by a Cartesian coordinate \((x, y, \text{or} \ z)\). The image can be written at a specific time step or at a specified time. Images can also be output at a fixed frequency, either specified by a time step interval or a time interval.

For example, the command

```
image mode=ux y=500 file=picturefile cycle=1
```

tells SW4 to output the \(x\)-component of the displacement (the solution) along the vertical \(y = 500\) plane. The data is written to a file named picturefile.cycle=1.y=500.ux.sw4img after the first time step \((cycle=1)\). The example

```
image mode=div x=1000 file=picturefile cycleInterval=100
```

outputs the divergence of the solution field in the \(yz\)-plane at the grid surface closest to \(x = 1000\). The data is written to the files
With this setup, one image file is output every 100 time steps.

Note that the divergence of the solution field does not contain shear (S) waves and the rotation (curl) of the solution field does not contain compressional (P) waves. These options can therefore be used to distinguish between P- and S-waves in the solution.

The hvelmax and vvelmax modes store the maximum in time of the horizontal and vertical velocity components, respectively. As these names indicate, it is assumed that the sources in SW4 are set up for calculating displacements. The horizontal velocity is defined as \( \max(|u_N|, |u_E|) \), where \( u_N \) and \( u_E \) are the displacement components in the North and East directions, respectively. The vertical velocity is \( |w_t| \), where \( w \) is the displacement component in the z-direction. For these modes, the cycleInterval or timeInterval options only determine how often the maxima are written to disk; the actual accumulation of the maximum is performed after each time step.

When SW4 is run in parallel, the data that gets saved on an image file originates from all processors that are intersected by the image plane. For horizontal image planes, this means all processors. To improve the I/O performance, image data is first communicated to a number of dedicated image writing processors. By default, 8 processors write each image file to disk (or all processors if SW4 is run on fewer than 8). This number can be changed using the fileio command,

```
fileio nwriters=4
```

The above command tells SW4 to use 4 processors to write each image file. For simulations that use very large number of grid points and many processors, care must be taken to make sure that enough memory is available to buffer the image data before it is written to disk.

Notes on the image command:

- By default, single precision data is saved. Double precision data can be saved by using the `precision=double` option.
- When topography is used, an image plane along the free surface is specified by the `z=0` option.
- A `mode=topo z=0` image holds the elevation (negative z-coordinate) of the raw topography. It can only be written when topography is used.
- A `mode=grid z=0` image holds the z-coordinate (negative elevation) of the grid along the free surface, which is the actual shape of the upper surface of the computational domain.
- When topography is used, vertical image planes intersect both component grids in the composite grid. In this case, cross-sectional data from both component grids are stored on the image file.
- When attenuation is enabled, the P- and S-phase velocities depend on frequency. The values saved on image files correspond to the zero frequency limit.
- The images files are written in a binary format, see Section 12.7 for details.
- We provide matlab/octave scripts for reading image files in the tools directory. The basic function is called `readimage.m`.  

58
Figure 9.1: Location of the source and stations for the Barnwell simulation. This figure was generated using the GMT command, see Section 11.5.4 for details.

9.4 Creating a GMT script with the gmt command

The Generic Mapping Toolkit (GMT) [21] is a suite of image generation programs for geophysical applications. These programs can be used to make postscript plots like Figure 9.1. In the example shown here, topography information is included as well as information on the general setup of the simulation. Note that the gmt command in SW4 causes an ASCII text file to be generated. This file contains a UNIX C-shell script with commands for the gmt programs, holding general information about the run such as geometric coordinates of the computational domain as well as locations of sources and receivers. There are many options for these programs, and they might need to be fine-tuned to suit the needs of a particular application, see the GMT documentation for details.

To have SW4 generate a GMT shell script file, you give the command

\begin{verbatim}
gmt file=bolinas.gmt
\end{verbatim}
Chapter 10

Examples

In addition to the layer over half-space problems discussed here, there are additional, more realistic, examples in the sw4/examples directory. Those cases illustrate the set up of heterogeneous material models and also visco-elastic attenuation.

10.1 The elastic layer over half-space problem: LOH.1

The LOH.1 problem, defined in the input script examples/scuc/LOH.1-h50.in, has a layered material model where the top 1000 meters \((z \in [0, 1000])\) has different properties than the rest of the domain. The computational domain is taken to be \((x,y,z) \in [0,30000]^2 \times [0,17000]\). The grid size is chosen to be \(h = 50 m\) and the material properties in the different layers are described by

\[
\begin{align*}
\text{grid} & \quad h=50 \quad x=30000 \quad y=30000 \quad z=17000 \\
\text{block} & \quad vp=4000 \quad vs=2000 \quad rho=2600 \\
\text{block} & \quad vp=6000 \quad vs=3464 \quad rho=2700 \quad z1=1000 \\
\text{block} & \quad vp=4630.76 \quad vs=2437.56 \quad rho=2650 \quad z1=999 \quad z2=1001
\end{align*}
\]

The last block command defines the properties right at the interface, using a harmonic average for the Lamé parameters \((\mu, \lambda)\), and an arithmetic average for density. The problem is driven by a single point moment source, positioned in the lower half-space. The time function in this problem is a Gaussian (if setup as in the input file, the Gaussian source is equivalent to using a Brune time function followed by a post processing deconvolution step, as is described in [4]). The advantage of using the Gaussian is that no post processing is necessary, and the Gaussian function produces less high wave number waves, which are poorly resolved on the computational mesh. Note that \(freq=16.6667\) corresponds to the spread \(\sigma = 0.06 \ (freq = 1/\sigma)\) in the Gaussian time function. The command lines to setup the source and the time duration of the simulation are:

\[
\begin{align*}
time & \quad t=9 \\
source & \quad x=15000 \quad y=15000 \quad z=2000 \quad mxy=1e18 \quad t0=0.36 \quad freq=16.6667 \\
& \quad \quad \quad \quad \quad \quad \text{type=Gaussian}
\end{align*}
\]

We use the fileio command to save all output files in the sub-directory LOH1-h50. If this directory does not exist, SW4 will attempt to create it for you.

fileio path=LOH1-h50
The magnitude of the solution along the free surface \((z = 0)\) is saved on an image file every 0.5 seconds:

\[
\text{image mode}=\text{mag} \ z=0 \ \text{file}=\text{surf} \ \text{timeInterval}=0.5
\]

In addition, the solution is recorded along a line of receivers on the free surface:

\[
\begin{align*}
\text{rec} \ x &= 15600 \ y=15800 \ z=0 \ \text{file}=\text{sta01} \ \text{usgsformat}=1 \\
\text{rec} \ x &= 16200 \ y=16600 \ z=0 \ \text{file}=\text{sta02} \ \text{usgsformat}=1 \\
\text{rec} \ x &= 16800 \ y=17400 \ z=0 \ \text{file}=\text{sta03} \ \text{usgsformat}=1 \\
\text{rec} \ x &= 17400 \ y=18200 \ z=0 \ \text{file}=\text{sta04} \ \text{usgsformat}=1 \\
\text{rec} \ x &= 18000 \ y=19000 \ z=0 \ \text{file}=\text{sta05} \ \text{usgsformat}=1 \\
\text{rec} \ x &= 18600 \ y=19800 \ z=0 \ \text{file}=\text{sta06} \ \text{usgsformat}=1 \\
\text{rec} \ x &= 19200 \ y=20600 \ z=0 \ \text{file}=\text{sta07} \ \text{usgsformat}=1 \\
\text{rec} \ x &= 19800 \ y=21400 \ z=0 \ \text{file}=\text{sta08} \ \text{usgsformat}=1 \\
\text{rec} \ x &= 20400 \ y=22200 \ z=0 \ \text{file}=\text{sta09} \ \text{usgsformat}=1 \\
\text{rec} \ x &= 21000 \ y=23000 \ z=0 \ \text{file}=\text{sta10} \ \text{usgsformat}=1
\end{align*}
\]

The velocity time histories for station 10 are shown in Figure 10.1 together with a semi-analytical solution. In Matlab or octave, the semi-analytical solution can be generated by the script in `tools/loh1exact.m`. The syntax is

\[
\text{octave:4> } [t \ ra \ tr \ ve]=\text{loh1exact}(0.06);
\]

As is customary in seismology, the velocity components have been rotated to polar components, with the origin at the source. The vertical component \((ve)\) is positive downwards. The \text{rec} command outputs the \(u_x, u_y\) and \(u_z\)-components of the velocity. These components are rotated to radial and transverse components using the transformations,

\[
u_{rad} = 0.6u_x + 0.8u_y, \quad u_{tran} = -0.8u_x + 0.6u_y.
\]

The vertical component is stored in \(u_z\) (positive downwards). We conclude that most features in the solution are very well captured on the grid with \(h = 50\). The numerical solution becomes almost identical with the semi-analytical solution when the grid is refined to \(h = 25\) (experiment not shown).

By using formulas (4.4)-(4.3), we can calculate the number of points per wave length for this simulation. Since we are using a Gaussian time-function, the center frequency is \(f_0 = 1/(2\pi\sigma) \approx 2.6526\) and we estimate the highest significant frequency to be \(f_{max} \approx 2.5f_0 = 6.6315\) Hz. The material model has \(\min V_s = 2000\) m/s. The grid has size \(h = 50\) m, which results in

\[
P = \frac{2000}{50 \cdot 6.6315} \approx 6.03.
\]

From our previous discussion (see Section 4.5), 6 points per wave length is on the low side, but visual inspection of Figure 10.1 indicates very good agreement of the wave forms. The finer grid with \(h = 25\) gives \(P \approx 12.1\). For the purpose of most engineering calculations, we suggest using in between \(P = 6\) and \(P = 12\) grid points per shortest wave length.
Figure 10.1: LOH.1: The radial (top), transverse (middle) and vertical (bottom) velocities for receiver number 10. Here the numerical solutions are plotted in red ($h = 50$) and the semi-analytical solution is plotted in black.

10.2 The visco-elastic layer over half-space problem: LOH.3

The LOH.3 benchmark problem adds effects of anelastic attenuation to the LOH.1 problem. It is defined in the input script examples/scec/LOH.3-h50.in. The input file is very similar to the LOH.1 case. The visco-elastic modeling is enabled by the attenuation command,

```
attenuation phasefreq=2.5 nmech=3 maxfreq=15
```

In this case, three standard linear solid mechanisms are used ($nmech=3$) to give a material with approximately constant quality factors in the frequency band $0.15 \leq f \leq 15$ Hz. Note that only the upper frequency limit needs to be specified ($maxfreq=15$); the lower limit is always 100 times smaller (and can not be specified). Since the visco-elastic material is dispersive, we use the `phasefreq=2.5` option to specify at what frequency the compressional and shear speeds should apply. In the description of the LOH.3 problem this frequency is given as 2.5 Hz, see Day et al. [5].

Apart from the density and the material velocities, the material model must include the quality
factors for the attenuation of compressional \( (Q_P) \) and shear waves \( (Q_S) \). For this problem, we use the block commands

\[
\begin{align*}
\text{block vs}=3464 & \quad \text{vp}=6000 & \quad \text{rho}=2700 & \quad Qs=69.3 & \quad Qp=155.9 \\
\text{block vs}=2000 & \quad \text{vp}=4000 & \quad \text{rho}=2600 & \quad z2=1000 & \quad Qs=40 & \quad Qp=120 \\
\text{block vs}=2437.6 & \quad \text{vp}=4630.8 & \quad \text{rho}=2650 & \quad z1=999 & \quad z2=1001
\end{align*}
\]

Similar to the LOH.1 test case, the source is of point moment-tensor type with a Gaussian time-function. However, note that the Gaussian has spread \( \sigma = 0.05 \) for LOH.3. This corresponds to center angular frequency \( \text{freq} = 1/\sigma = 20 \text{ rad/s} \) and center frequency \( f_0 = 20/(2\pi) \approx 3.18 \text{ Hz} \). The source is specified by the command

\[
\text{source x}=15000 \quad y=15000 \quad z=2000 \quad mxy=1e18 \quad t0=0.3 \quad \text{freq}=20 \ \text{type=Gaussian}
\]

Again, artifacts from a sudden startup are avoided by taking the center time to be \( t0=6\sigma = 0.3 \). The solution is recorded in the same array of receivers as before. In this case the semi-analytical solution can be created by the Matlab/Octave script \( \text{tools/loh3exact.m} \). The syntax is

\[
\text{octave:15> [t ra tr ve]=loh3exact(0.05)}; \\
\text{octave:23> [t1 ux uy uz]=readusgs("sta10.txt");}
\]

The components are rotated to polar components using the Octave commands,

\[
\text{octave:25> ur = 0.6*ux + 0.8*uy}; \\
\text{octave:26> ut = -0.8*ux + 0.6*uy};
\]

We can compare the radial component of the semi-analytical and numerical solutions by giving the commands

\[
\text{octave:33> plot(t,ra,"k",t1,ur,"r");axis([0 9 -1.5 1.5])}
\]

All three components of the solution at station 10 are shown in Figure 10.2. Note that the wave forms are very similar to LOH.1, but the amplitudes are slightly smaller. As for LOH.1, we can estimate the resolution in terms of the number of grid points per shortest significant wave length. In this case the center frequency is \( f_0 \approx 3.18 \text{ Hz} \) and we estimate the upper power frequency to be \( f_{max} \approx 2.5f_0 = 7.95 \text{ Hz} \). The material model has \( \min V_s = 2000 \text{ m/s} \) where the grid size is \( h = 50 \text{ m} \), and we arrive at

\[
P = \frac{2000}{50\cdot7.95} \approx 5.03.
\]

From our previous discussion (see Section 4.5), 5 points per wave length can only be expected to give marginal accuracy, but visual inspection of Figure 10.2 still indicates rather good agreement of the wave forms. Note that the visco-elastic dissipation damps out higher frequencies faster than lower frequencies. Station 10 is 10 km away from the source, and the dominant wave length is approximately \( 2000/3.18 \approx 628.9 \text{ meters} \). Hence, by the time the solution reaches station 10, it has propagated about 16 wave lengths. The higher frequency components of the solution are therefore less prominent compared to the purely elastic LOH.1 problem. This can also be seen by comparing the amplitudes in Figures 10.2 and 10.1.
Figure 10.2: LOH.3: The radial (top), transverse (middle) and vertical (bottom) velocities at receiver number 10. Here the numerical solution with grid size $h = 50$ is plotted in red and the semi-analytical solution is plotted in black.
Chapter 11

Keywords in the input file

The syntax of the input file is

command1 parameter1=value1 parameter2=value2 ... parameterN=valueN
# comments are disregarded
command2 parameter1=value1 parameter2=value2 ... parameterM=valueM
...

Each command starts at the beginning of the line and ends at the end of the same line. Blank and comment lines are disregarded. A comment is a line starting with a # character. The order of the parameters within each command makes no difference. The material commands (block, ifile, pfile, and efile) are applied in the order they appear. The ordering of all other commands is inconsequential. Note that the entire input file is read before the simulation starts.

Parameter values are either integers (-2,0,5,...), real numbers (20.5, -0.05, 3.4e4), or strings (earthquake, my-favorite-simulation). Note that there must be no spaces around the = signs and strings are given without quotation marks and must not contain spaces. Depending on the specific command, some parameter values are required to fall within specified ranges.

A brief description of all commands is given in the following sections. The commands marked as [required] must be present in all $SW4$ input files, while those marked as [optional] are just that. Other commands, such as those specifying the material model can be given by a combination of different commands (block, pfile, efile, or ifile). Note that some required commands must occur exactly once (grid, time). Some optional commands should not occur more than once (topography, fileio, prefilter, globalmaterial, gmt, developer). Any number of output commands can be given (rec, image, volimage). Unless $SW4$ is run in one of its test modes (twilight, testlamb, testpointsource, testrayleigh, testenergy), at least one source must be specified, and the material must be specified by at least one of the (block, pfile, ifile, efile) commands. Also note that the test modes are mutually exclusive. Not all of these rules are currently enforced by the parser, but $SW4$ can give unexpected behavior if they are violated.

Note that the same command parser is used for $SW4$ and its companion code $SW4opt$, which calculates source parameters from seismic observations. Here we only document the commands and options that are relevant for $SW4$. 

65
11.1 Basic commands

11.1.1 fileio [optional]

The `fileio` command is used for specifying output directories, setting the amount of information output by SW4, the output frequency during the time-stepping, as well as enabling fast I/O for parallel file system. See §9.1 for more information.

Syntax:

```
fileio path=... verbose=... printcycle=... pfs=... nwriters=...
```

Required parameters:

None

<table>
<thead>
<tr>
<th>fileio command parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Option</strong></td>
</tr>
<tr>
<td>path</td>
</tr>
<tr>
<td>verbose</td>
</tr>
<tr>
<td>printcycle</td>
</tr>
<tr>
<td>pfs</td>
</tr>
<tr>
<td>nwriters</td>
</tr>
</tbody>
</table>

11.1.2 grid [required]

Syntax:

```
grid nx=... ny=... nz=... x=... y=... z=... h=... lat=... lon=... az=... mlat=... mlon=... proj=... ellps=... datum=... scale=... lat_p=... lon_p=...```

Required parameters:

See below.

The grid command specifies the extent of the computational domain and the grid size in the base grid. Optionally the grid command also specifies the latitude and longitude of the origin and the azimuth angle between North and the x-axis. A number of different projections can be specified.

There are three basic ways of specifying the extent of the computational domain and the grid size:

- number of grid points in all three directions and the grid size: `nx=... ny=... nz=... h=...`
- lengths in all three directions and the grid size: `x=... y=... z=... h=...`
- lengths in all three directions and the number of grid points in one direction (the x-direction in this example): \( x=... \ y=... \ z=... \ nx=... \) 

It is not allowed to over-specify the grid size. For example, if \( x=... \) is given, you can not specify both \( h=... \) and \( nx=... \). Similarly, it is not allowed to over-specify the extent of the computational domain. For example, when \( h=... \) is given, you can not prescribe both \( y=... \) and \( ny=... \).

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>physical dimension of grid in the x-direction</td>
<td>real</td>
<td>m</td>
<td>none</td>
</tr>
<tr>
<td>( y )</td>
<td>physical dimension of grid in the y-direction</td>
<td>real</td>
<td>m</td>
<td>none</td>
</tr>
<tr>
<td>( z )</td>
<td>physical dimension of grid in the z-direction</td>
<td>real</td>
<td>m</td>
<td>none</td>
</tr>
<tr>
<td>( h )</td>
<td>grid spacing</td>
<td>real</td>
<td>m</td>
<td>none</td>
</tr>
<tr>
<td>( nx )</td>
<td>number of grid points in the x-direction</td>
<td>int</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>( ny )</td>
<td>number of grid points in the y-direction</td>
<td>int</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>( nz )</td>
<td>number of grid points in the z-direction</td>
<td>int</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>

The default projection is spheroidal as described by equations (3.6)-(3.7). You can change the parameter \( M \) with the \texttt{mlat} keyword. By using the \texttt{mlon} keyword, you modify the projection by replacing \( M \cos(\phi \pi/180) \) in (3.7) by the constant \( M_{lon} \).

More accurate projections are available through the Proj4 library (if \textit{SW4} was built with Proj4 support). These projections are enabled by using one of the keywords \texttt{proj}, \texttt{ellps}, \texttt{datum}, \texttt{scale}, \texttt{lat.p}, or \texttt{lon.p}. The values assigned to these keywords are used to generate a string which is passed directly to the \texttt{pj_init_plus} routine in the Proj4 library. For example, the \texttt{grid} command

```
grid x=12e3 y=12e3 z=5e3 nx=601 lat=37.93 lon=-122.25 az=143.6380 proj=tmerc \\
    datum=NAD83 lon_p=-123.0 lat_p=35.0 scale=0.9996
```

results in the string

```
pstr = "+units=m +proj=tmerc +datum=NAD83 +lon_0=-123.0 +lat_0=35.0 +scale=0.9996"
```

Note that the values of \texttt{lat.p} and \texttt{lon.p} are passed as arguments to \texttt{lat.0} and \texttt{lon.0}, respectively. See the Proj4 documentation for further guidance.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>az</td>
<td>clockwise angle from North to the x-axis</td>
<td>real</td>
<td>degrees</td>
<td>135.0</td>
</tr>
<tr>
<td>lat</td>
<td>latitude geographical coordinate of the origin</td>
<td>real</td>
<td>degrees</td>
<td>37.0</td>
</tr>
<tr>
<td>lon</td>
<td>longitude geographical coordinate of the origin</td>
<td>real</td>
<td>degrees</td>
<td>-118.0</td>
</tr>
<tr>
<td>mlat</td>
<td>meters per degree of latitude (spheroidal projection)</td>
<td>real</td>
<td>meters</td>
<td>111,319.5</td>
</tr>
<tr>
<td>mlon</td>
<td>meters per degree of longitude (spheroidal projection)</td>
<td>real</td>
<td>meters</td>
<td>None</td>
</tr>
<tr>
<td>proj</td>
<td>name of projection (see proj4 documentation)</td>
<td>string</td>
<td>None</td>
<td>utm</td>
</tr>
<tr>
<td>ellps</td>
<td>name of ellipse (see proj4 documentation)</td>
<td>string</td>
<td>None</td>
<td>WGS84</td>
</tr>
<tr>
<td>datum</td>
<td>datum of projection (e.g. NAD83)</td>
<td>string</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>lon_p</td>
<td>central meridian of projection</td>
<td>string</td>
<td>degrees</td>
<td>lon</td>
</tr>
<tr>
<td>lat_p</td>
<td>latitude of projection origin</td>
<td>string</td>
<td>degrees</td>
<td>lat</td>
</tr>
<tr>
<td>scale</td>
<td>scale factor for central meridian</td>
<td>string</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

Note: The default projection, given by equations (3.6)-(3.7), is used if neither of the proj, ellps, datum, scale, lat_p, or lon_p keywords are specified. The default value for lat_p and lon_p are taken from the geographic coordinates of the grid origin (\lon and \lat, respectively). The default value for ellps is only used in the case that datum is not specified.

11.1.3 time [required]

Syntax:
```
time t=... steps=... utcstart=...
```

Required parameters:
t or steps

The time command specifies the duration of the simulation. You can either specify the final time in seconds by using t, or specify the number of time-steps with steps. The size of the time step is computed internally by SW4. You may not over specify the duration of the simulation, i.e., you can not give both t=... and steps=....

The optional utcstart keyword is used to assign the Universal Time Coordinate (UTC) corresponding to simulation time t = 0. The format of the UTC time is a string (without quotation) “month/day/year:hour:minute:second.millisecond”. For example, the 17th hour, 34th minute, 12th second and 233th millisecond of January 31, year 2012, is encoded as utcstart=01/31/2012:17:34:12.233. When the UTC time is set, all time-series (saved with the rec command) are time stamped with that datum. The UTC time stamp is essential for correctly aligning observed data when solving the inverse problem with SW4opt.

Note that the prefilter command does not modify the start time. This is a change from WPP. See §11.1.5 for a discussion.
### time command parameters

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>duration of simulation</td>
<td>real</td>
<td>s</td>
<td>none</td>
</tr>
<tr>
<td>steps</td>
<td>number of cycles (time-steps) to advance</td>
<td>int</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>utcstart</td>
<td>month/day/year:hour:minute:second.millisecond</td>
<td>string</td>
<td>datum</td>
<td>SW4 start time</td>
</tr>
</tbody>
</table>

### 11.1.4 supergrid [optional]

**Syntax:**

supergrid gp=... dc=... width=...

**Required parameters:**

None

Note that the keywords are different from WPP. Also note that increasing dc may lead to instabilities.

### supergrid command parameters

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp</td>
<td>Number of grid points in the supergrid layer</td>
<td>int</td>
<td>none</td>
<td>30</td>
</tr>
<tr>
<td>width</td>
<td>Physical width of the supergrid layer</td>
<td>float</td>
<td>meters</td>
<td>none</td>
</tr>
<tr>
<td>dc</td>
<td>Damping coefficient in supergrid region</td>
<td>real</td>
<td>none</td>
<td>0.02</td>
</tr>
</tbody>
</table>

The parameters gp and width are mutually exclusive, setting both will lead to an error.

### 11.1.5 prefilter [optional]

**Syntax:**

prefilter fc1=... fc2=... type=... passes=... order=...

**Required parameters:**

None

The prefilter command is used to filter all source time functions before the simulation starts. This approach gives the same result as filtering all time-series after the simulation is completed. Hence, the prefilter command can be used to reduce the amount of post processing. The command is particularly useful in combination with the Dirac source time function, which triggers all frequencies on the grid. The prefilter option is also useful for removing unphysical modes from image files, e.g. max velocities or displacements. The prefilter command modifies the time functions in all source commands using a discrete Butterworth filter. Lowpass and bandpass filters are supported, with orders between 1 and 10. Only the fc2 frequency is used for lowpass filters, while it is assumed that fc1<fc2 for bandpass filters. The filtering is either forwards in time (passes=1), or forwards and backwards (passes=2). For passes=1, the filter is causal but gives the filtered signal a phase shift. When passes=2 the filter has zero phase shift, but is acausal. In this case, the filtered signal is only exponentially small as t → −∞. In order to avoid unphysical oscillations...
due to an abrupt start, an estimate of the length of this tail is calculated. A warning message is printed to stdout if the time shift \((t0)\) in the source is smaller than this value.

<table>
<thead>
<tr>
<th>Prefilter command parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Option</strong></td>
</tr>
<tr>
<td>fc1</td>
</tr>
<tr>
<td>fc2</td>
</tr>
<tr>
<td>type</td>
</tr>
<tr>
<td>passes</td>
</tr>
<tr>
<td>order</td>
</tr>
</tbody>
</table>

11.2 Sources [required]

11.2.1 source

**Syntax:**

source x=... y=... z=... lat=... lon=... depth=... topodepth=... m0=... mxx=... mxy=... mxz=... myy=... myz=... mzz=... f0=... fx=... fy=... fz=... rake=... strike=... dip=... t0=... freq=... type=... ncyc=... dfile=...

**Required parameters:**

See below.

There can be multiple source commands in an input file. Each source command either sets up a point force or a point moment tensor source and should follow the following rules:

- The location of the source must be specified by either Cartesian \((x, y, z)\) or geographical \((\text{lat}, \text{lon}, \text{depth} \text{ or topodepth})\) coordinates. The depth below mean sealevel \((z = 0)\) is specified with \(z\), while \text{depth} or \text{topodepth} specifies the depth below the topography.

- Select a point force or a point moment tensor source:
  - Point force: give at least one component of the force vector \((fx, fy, fz)\) and optionally the amplitude \(f0\).
  - A point moment tensor source can be specified in one of two ways:
    1. Seismic moment \(m0\), and double couple focal mechanism, \text{strike/dip/rake} angles (as defined in Aki and Richards [1]).
    2. At least one component of the moment tensor \((mxx, mxy, etc.)\) and optionally a scaling factor \(m0\).

- Specify a pre-defined source time function (with the \text{type} keyword), or give the file name for a discrete time function (using the \text{dfile} keyword).

Note that all pre-defined time functions use the \(t0\) keyword, and all functions except \text{Dirac} also use the \text{freq} keyword. The only pre-defined time function that uses the \text{ncyc} keyword is \text{GaussianWindow}.
source command parameters (part 1)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x position of the source (within domain)</td>
<td>real</td>
<td>m</td>
<td>none</td>
</tr>
<tr>
<td>y</td>
<td>y position of the source (within domain)</td>
<td>real</td>
<td>m</td>
<td>none</td>
</tr>
<tr>
<td>z</td>
<td>z position of the source (within domain)</td>
<td>real</td>
<td>m</td>
<td>none</td>
</tr>
<tr>
<td>depth</td>
<td>depth of the source ($\geq 0$)</td>
<td>real</td>
<td>m</td>
<td>none</td>
</tr>
<tr>
<td>topodepth</td>
<td>(same as depth)</td>
<td>real</td>
<td>m</td>
<td>none</td>
</tr>
<tr>
<td>lat</td>
<td>latitude geographical coordinate of the source</td>
<td>real</td>
<td>degrees</td>
<td>none</td>
</tr>
<tr>
<td>lon</td>
<td>longitude geographical coordinate of the source</td>
<td>real</td>
<td>degrees</td>
<td>none</td>
</tr>
<tr>
<td>t0</td>
<td>offset in time ($\geq 0$)</td>
<td>real</td>
<td>s</td>
<td>0.0</td>
</tr>
<tr>
<td>freq</td>
<td>frequency ($&gt;0$) (not used for Dirac)</td>
<td>real</td>
<td>Hz or rad/s</td>
<td>1.0</td>
</tr>
<tr>
<td>type</td>
<td>Name of source time function</td>
<td>string</td>
<td>none</td>
<td>RickerInt</td>
</tr>
<tr>
<td>nycrc</td>
<td>Number of cycles (must be specified for the GaussianWindow function)</td>
<td>int</td>
<td>none</td>
<td>0</td>
</tr>
<tr>
<td>dfile</td>
<td>File name for discrete time function</td>
<td>string</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>

The **type** keyword specifies the source time function. It can have the following values: GaussianInt, Erf, Gaussian, RickerInt, Ricker, Ramp, Triangle, Sawtooth, Smoothwave, VerySmoothBump, Brune, BruneSmoothed, GaussianWindow, Liu, Dirac, and C6SmoothBump. The functions are described in §4.2.

source command parameters (point moment tensor)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>m0</td>
<td>moment amplitude</td>
<td>real</td>
<td>Nm</td>
<td>1.0</td>
</tr>
<tr>
<td>mxx</td>
<td>xx-component of the moment tensor</td>
<td>real</td>
<td>Nm</td>
<td>0.0</td>
</tr>
<tr>
<td>myy</td>
<td>yy-component of the moment tensor</td>
<td>real</td>
<td>Nm</td>
<td>0.0</td>
</tr>
<tr>
<td>mzz</td>
<td>zz-component of the moment tensor</td>
<td>real</td>
<td>Nm</td>
<td>0.0</td>
</tr>
<tr>
<td>mxy</td>
<td>xy-component of the moment tensor</td>
<td>real</td>
<td>Nm</td>
<td>0.0</td>
</tr>
<tr>
<td>mxz</td>
<td>xz-component of the moment tensor</td>
<td>real</td>
<td>Nm</td>
<td>0.0</td>
</tr>
<tr>
<td>myz</td>
<td>yz-component of the moment tensor</td>
<td>real</td>
<td>Nm</td>
<td>0.0</td>
</tr>
<tr>
<td>strike</td>
<td>strike angle (see Aki and Richards)</td>
<td>real</td>
<td>degrees</td>
<td>none</td>
</tr>
<tr>
<td>dip</td>
<td>dip angle</td>
<td>real</td>
<td>degrees</td>
<td>none</td>
</tr>
<tr>
<td>rake</td>
<td>rake angle</td>
<td>real</td>
<td>degrees</td>
<td>none</td>
</tr>
</tbody>
</table>
source command parameters (point force)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>f0</td>
<td>point force amplitude</td>
<td>real</td>
<td>N</td>
<td>1.0</td>
</tr>
<tr>
<td>fx</td>
<td>forcing function in the x direction</td>
<td>real</td>
<td>N</td>
<td>0.0</td>
</tr>
<tr>
<td>fy</td>
<td>forcing function in the y direction</td>
<td>real</td>
<td>N</td>
<td>0.0</td>
</tr>
<tr>
<td>fz</td>
<td>forcing function in the z direction</td>
<td>real</td>
<td>N</td>
<td>0.0</td>
</tr>
</tbody>
</table>

11.2.2 rupture

Syntax:
rupture file=...

Required parameters:
name of SRF file.

The rupture command is used to describe complex time-dependent rupture mechanisms over a fault surface. The rupture command reads a SRF (Standard Rupture Format) file and generates a kinematic rupture model consisting of a set of moment tensor source terms. The local shear modulus of the material is taken into account to calculate the strength of each source terms such that the prescribed amount of slip is achieved on each sub-fault of the SRF file.

The SRF file format is described on the SCEC wiki webpage:
http://scec.usc.edu/scecpedia/Standard_Rupture_Format

11.3 The material model [required]

It is required that the material model is defined in the entire computational domain. The material properties are extrapolated to the ghost points if they are not covered by the material model (this is a change from WPP). The material commands block, ifile, efile, and pfile, are applied in the same order as they are given. Hence, it is possible to overwrite the properties specified by a material command given earlier in the file. This can be particularly useful when using the block command. Finally, the properties of the optional globalmaterial command are enforced after all other material commands have been applied. Also note that the input file is scanned for the attenuation command before any other commands are parsed. This command may be located anywhere in the input file.

11.3.1 attenuation [optional]

The attenuation command is used to enable visco-elastic modeling as described in Section 8. The visco-elastic model is defined through the quality factors $Q_P$ and $Q_S$. Similar to the elastic properties, the quality factors may vary from point to point throughout the computational domain. If visco-elastic modeling is enabled, the $Q_P$ and $Q_S$ factors must be specified as part of every material command described below. When visco-elastic modeling is not enabled, the $Q_P$ and $Q_S$ factors are not required in the material commands, and are ignored if present.

Syntax:
attenuation phasefreq=... nmech=... maxfreq=... minppw=... qmultiplier=...

Required parameters:
None

Note: you may not specify both maxfreq and minppw.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>phasefreq</td>
<td>The frequency (&gt; 0) at which $C_S$ and $C_P$ are</td>
<td>real</td>
<td>Hz</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>specified</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nmech</td>
<td>Number of SLS mechanisms to approximate constant</td>
<td>int</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>$Q_P$ and $Q_S$ (between 0 and 8)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxfreq</td>
<td>The upper frequency limit (&gt; 0) for approximating</td>
<td>real</td>
<td>Hz</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>constant $Q_P$ and $Q_S$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>minppw</td>
<td>Calculate the upper frequency limit based on</td>
<td>real</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>this number of grid points per shortest wave</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>length (&gt; 0)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>qmultiplier</td>
<td>A number that will multiply both $Q_P$ and $Q_S$</td>
<td>real</td>
<td>&gt;0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

If you specify the minppw option, the upper frequency limit is calculated based on the relation

$$P = \min C_S/(hf)$$

i.e.,

$$f_{max} = \frac{1}{P_{min}} \min \frac{C_S}{h}, \quad f_{max} = \text{maxfreq}, \quad P_{min} = \text{minppw}.$$  

The qmultiplier value will multiply $Q_P$ and $Q_S$ at all grid points, before the computation starts,

$$Q_P := \text{qmultiplier} \times Q_P \quad Q_S := \text{qmultiplier} \times Q_S.$$  

This option gives the user an easy way to modify the quality factors, even if they are given on material file formats that are hard to access manually. Use qmultiplier with caution. If the quality factors are small (i.e., $\text{qmultiplier}<<1$), the equations will become ill-posed. SW4 will detect too small quality factors and terminate before the time stepping starts.

As a computationally inexpensive alternative to the visco-elastic modeling described in Section 8 it is possible to set nmech=0. In this case, the attenuation modeling is performed without memory variables. After each time step, the solution field at each grid point $x_{i,j,k}$ is simply multiplied by the factor

$$e^{-\pi f_c \delta t / Q},$$

where $\delta t$ is the time step, $f_c$ is the center frequency, and $Q = Q_S(x_{i,j,k})$. The center frequency is specified by the maxfreq keyword. Note that the attenuation factor $Q_P$ is not used in this case, but it must still be specified.

11.3.2 block

Syntax:

block vp=... vs=... rho=... qp=... qs=... vpgrad=... vsgrad=... 
| rhograd=... absdepth=... x1=... x2=... y1=... y2=... z1=... z2=... |
Required parameters:
vp, vs, rho (qp and qs with attenuation)

The block command specifies material properties that are constant or vary linearly with depth. By default, the material properties apply to the entire computational domain. By using the optional parameters \( x1=... \), \( x2=... \), etc., the material properties are only assigned in parts of the computational domain. When used together with the topography command, the \( \text{absdepth} \) flag determines how the \( z \)-coordinates are used. If \( \text{absdepth}=0 \) (default) \( z1=... \) and \( z2=... \) specify depths below the free surface. If \( \text{absdepth}=1, z1=... \) and \( z2=... \) bound the \( z \)-coordinate of the material block.

The gradient parameters \( \text{vpgrad}, \text{vsgrad}, \text{rhograd} \) specify linear variations in the \( z \)-direction (downward). The units for \( \text{vpgrad} \) and \( \text{vsgrad} \) are 1/seconds, which can be interpreted as m/s per m, or km/s per km. The linear variation is relative to the properties at the free surface \( (z=0 \text{ or depth}=0 \text{ with topography}) \), e.g.,

\[
C_p(z) = \text{vp} + z \text{vpgrad}.
\]

Note that when \( \text{vpgrad} \) is specified together with \( z1 = z_1, C_p(z_1) = \text{vp} + z_1 \text{vpgrad} \). Hence, the material properties at the top of the block \( (z = z_1) \) can be very different from \( \text{vp} \) when \( z_1 \text{vpgrad} \) is large.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{vp} )</td>
<td>P-wave velocity</td>
<td>real</td>
<td>m/s</td>
<td>none</td>
</tr>
<tr>
<td>( \text{vs} )</td>
<td>S-wave velocity</td>
<td>real</td>
<td>m/s</td>
<td>none</td>
</tr>
<tr>
<td>( \text{rho} )</td>
<td>density</td>
<td>real</td>
<td>kg/m(^3)</td>
<td>none</td>
</tr>
<tr>
<td>( \text{vpgrad} )</td>
<td>vertical gradient for ( \text{vp} )</td>
<td>real</td>
<td>m/s/m</td>
<td>none</td>
</tr>
<tr>
<td>( \text{vsgrad} )</td>
<td>vertical gradient for ( \text{vs} )</td>
<td>real</td>
<td>m/s/m</td>
<td>none</td>
</tr>
<tr>
<td>( \text{rhograd} )</td>
<td>vertical gradient for ( \text{rho} )</td>
<td>real</td>
<td>kg/m(^4)</td>
<td>none</td>
</tr>
<tr>
<td>( \text{qp} ) (or ( \text{Qp} ))</td>
<td>P-wave quality factor</td>
<td>real</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>( \text{qs} ) (or ( \text{Qs} ))</td>
<td>S-wave quality factor</td>
<td>real</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x1 )</td>
<td>minimum x-dim for the box shaped sub-region</td>
<td>real</td>
<td>m</td>
<td>-max x</td>
</tr>
<tr>
<td>( x2 )</td>
<td>maximum x-dim for the box shaped sub-region</td>
<td>real</td>
<td>m</td>
<td>2 max x</td>
</tr>
<tr>
<td>( y1 )</td>
<td>minimum y-dim for the box shaped sub-region</td>
<td>real</td>
<td>m</td>
<td>-max y</td>
</tr>
<tr>
<td>( y2 )</td>
<td>maximum y-dim for the box shaped sub-region</td>
<td>real</td>
<td>m</td>
<td>2 max y</td>
</tr>
<tr>
<td>( z1 )</td>
<td>minimum z-dim for the box shaped sub-region</td>
<td>real</td>
<td>m</td>
<td>-max z</td>
</tr>
<tr>
<td>( z2 )</td>
<td>maximum z-dim for the box shaped sub-region</td>
<td>real</td>
<td>m</td>
<td>2 max z</td>
</tr>
<tr>
<td>( \text{absdepth} )</td>
<td>( z1 ) and ( z2 ) relative to topography (0), or absolute ( z )-coordinate (1)</td>
<td>int</td>
<td>none</td>
<td>0</td>
</tr>
</tbody>
</table>

74
11.3.3 efile

Syntax:
```
efile etree=... xetree=... logfile=... query=... access=... resolution=...
```

Required parameters:
```
etree
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>etree</td>
<td>full path to the etree database file</td>
<td>string</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>xetree</td>
<td>full path to the extended etree database file</td>
<td>string</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>logfile</td>
<td>name of log file</td>
<td>string</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>type of query to perform</td>
<td>string</td>
<td>none</td>
<td>MAXRES</td>
</tr>
<tr>
<td>resolution</td>
<td>average properties over this distance (for</td>
<td>real</td>
<td>m</td>
<td>h</td>
</tr>
<tr>
<td></td>
<td>query=FIXEDRES)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>access</td>
<td>can be set to parallel or serial</td>
<td>string</td>
<td>none</td>
<td>parallel</td>
</tr>
</tbody>
</table>

The query option can be set to one of the following:

<table>
<thead>
<tr>
<th>query option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXRES</td>
<td>Sample the data at the maximum available resolution in the database. This is the default query type.</td>
</tr>
<tr>
<td>FIXEDRES</td>
<td>Average the material properties at the requested resolution, which is specified with the resolution keyword. The default resolution is the grid spacing, $h$.</td>
</tr>
</tbody>
</table>

For example, to set the data to be sampled at 1 km resolution:

```
efile query=FIXEDRES resolution=1000 etree=USGS-SF1906.etree
```

Note: the logfile option can be used to track if any grid points were outside the etree database domain, or if any grid points were located in the air.

11.3.4 pfile

Syntax:
```
pfile filename=... directory=... smoothingsize=... vpmin=... vsmin=... rhomin=... flatten=... style=...
```

Required parameters:
```
filename
```
<table>
<thead>
<tr>
<th>pfile command parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
</tr>
<tr>
<td>filename</td>
</tr>
<tr>
<td>directory</td>
</tr>
<tr>
<td>smoothingsize</td>
</tr>
<tr>
<td>vpmin</td>
</tr>
<tr>
<td>vsmin</td>
</tr>
<tr>
<td>rhomin</td>
</tr>
<tr>
<td>flatten</td>
</tr>
<tr>
<td>style</td>
</tr>
</tbody>
</table>

11.3.5 rfile
Syntax:
rfile filename=... directory=...
Required parameters:
filename

<table>
<thead>
<tr>
<th>rfile command parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
</tr>
<tr>
<td>filename</td>
</tr>
<tr>
<td>directory</td>
</tr>
</tbody>
</table>

Note that only relative paths are currently supported in the filename keyword. Use the directory keyword to specify absolute paths.

11.3.6 ifile
Syntax:
ifile filename=... input=...
Required parameters:
filename

The ifile command specifies the depth of material surfaces as function of longitude and latitude, and must be used in conjunction with the material command. The file format is described in Section 12.4.

<table>
<thead>
<tr>
<th>ifile command parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
</tr>
<tr>
<td>filename</td>
</tr>
<tr>
<td>input</td>
</tr>
</tbody>
</table>
If input=cartesian, the file is assumed to give the material surfaces as function of the $x$- and $y$-coordinates. If input=geographic, they are functions of latitude and longitude. See Section 12.4 for details.

11.3.7 material

Syntax:

\[
\text{material id=}... \quad \text{vp=}... \quad \text{vs=}... \quad \text{rho=}... \quad \text{vpgrad=}... \quad \text{vsgrad=}... \quad \text{rhograd=}... \\
\text{vp2=}... \quad \text{vs2=}... \quad \text{rho2=}... \quad \text{vpsqrt=}... \quad \text{vssqrt=}... \quad \text{rhosqrt=}... \quad \text{qp=}... \quad \text{qs=}...
\]

Required parameters:

id, vp, vs, rho

The material command is used to define material properties together with the ifile command, see Section 12.4 for the file format.

<table>
<thead>
<tr>
<th>material command parameters (constants)</th>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>material ID number $&gt; 0$</td>
<td>int</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>vp</td>
<td>P-wave velocity</td>
<td>real</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>vs</td>
<td>S-wave velocity</td>
<td>real</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>rho</td>
<td>Density</td>
<td>real</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>qp or Qp</td>
<td>P-wave quality factor</td>
<td>None</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>qs or Qs</td>
<td>S-wave quality factor</td>
<td>None</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>material command parameters (gradients)</th>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>vpgrad</td>
<td>P-velocity gradient</td>
<td>real</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>vsgrad</td>
<td>S-velocity gradient</td>
<td>real</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>rhograd</td>
<td>Density gradient</td>
<td>real</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>material command parameters (higher order)</th>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>vp2</td>
<td>P-velocity quadratic coefficient</td>
<td>real</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>vs2</td>
<td>S-velocity quadratic coefficient</td>
<td>real</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>rho2</td>
<td>Density quadratic coefficient</td>
<td>real</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>vpsqrt</td>
<td>P-velocity $\sqrt{z}$ coefficient</td>
<td>real</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>vssqrt</td>
<td>S-velocity $\sqrt{z}$ coefficient</td>
<td>real</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>rhosqrt</td>
<td>Density $\sqrt{z}$ coefficient</td>
<td>real</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>
11.3.8 globalmaterial [optional]

Syntax:
globalmaterial vpmin=... vsmin=...

Required parameters:
None

The **globalmaterial** command is used to put threshold values on the P- and S-velocities in the material model. These thresholds are enforced after the material properties have been assigned to all grid points.

<table>
<thead>
<tr>
<th>globalmaterial command parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
</tr>
<tr>
<td>vpmin</td>
</tr>
<tr>
<td>vsmin</td>
</tr>
</tbody>
</table>

11.3.9 randomize [optional]

Syntax:
randomize amplitude=... lengthscale=... lengthscalez=... gradient=... sdthreshold=... seed1=... seed2=... seed3=...

Required parameters:
None

The **randomize** command makes a random perturbation of the material velocities $C_p$ and $C_s$, according to

$$C_p := (1 + A(z) \times \theta)C_p \quad C_s := (1 + A(z) \times \theta)C_s$$

where $\theta$ is a random field with correlation lengths specified by the **lengthscale** and **lengthscalez** options. The range of $\theta$ is limited to $-n\sigma \leq \theta \leq n\sigma$, where $\sigma$ is the standard deviation of $\theta$ and $n$ is set by the parameter **sdthreshold**. By default $n = 3$. This is large enough to avoid visible “clipping” effects. To enforce $-1 \leq \theta \leq 1$, set $n = 1.732$. $C_p$ and $C_s$ use the same random field, hence the ratio $C_p/C_s$ is unaffected by the perturbation. The density is not perturbed. The perturbation amplitude is allowed to vary with the z-coordinate as

$$A(z) = A_0 + g \times z,$$

where $A_0$ is specified by the **amplitude** option, and $g$ is specified by the **gradient** option. The random field, $\theta$ is generated by smoothing a completely uncorrelated field of uniformly distributed pseudo-random numbers. The pseudo-random number generator uses three seed variables. Varying the seeds will give a different perturbations, but the random perturbation is not affected by the number of processors.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>amplitude</td>
<td>Amplitude of random perturbation ($A_0$)</td>
<td>real</td>
<td>None</td>
<td>0.1</td>
</tr>
<tr>
<td>lengthscale</td>
<td>Horizontal correlation length</td>
<td>real</td>
<td>m</td>
<td>100</td>
</tr>
<tr>
<td>lengthscalez</td>
<td>Vertical correlation length</td>
<td>real</td>
<td>m</td>
<td>lengthscale</td>
</tr>
<tr>
<td>sdthreshold</td>
<td>Limits range of $\theta$ to $\pm sdthreshold$ $\sigma$</td>
<td>real</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>seed1</td>
<td>Random number generator seed</td>
<td>int</td>
<td>None</td>
<td>1234</td>
</tr>
<tr>
<td>seed2</td>
<td>Random number generator seed</td>
<td>int</td>
<td>None</td>
<td>5678</td>
</tr>
<tr>
<td>seed3</td>
<td>Random number generator seed</td>
<td>int</td>
<td>None</td>
<td>9876</td>
</tr>
<tr>
<td>gradient</td>
<td>Amplitude gradient in the depth direction</td>
<td>real</td>
<td>1/m</td>
<td>0</td>
</tr>
</tbody>
</table>

### 11.3.10 anisotropy [optional]

**Syntax:**

anisotropy

**Required parameters:**

None

The *anisotropy* command enables modeling of general anisotropic elastic materials described by a 21 parameter stiffness matrix $C$. The $6 \times 6$ stiffness matrix is symmetric and must be positive definite. Currently, the anisotropic model can *not* be used together with attenuation. Furthermore, the material model can only be specified with the *ablock* command, see Section 11.3.11. Material commands for the isotropic model will be ignored when the *anisotropy* command is used.

### 11.3.11 ablock [optional]

**Syntax:**

```plaintext```
ablock x1=... x2=... y1=... y2=... z1=... z2=... rho=... rhograd=...
c11=... c12=... c13=... c14=... c15=... c16=...
c22=... c23=... c24=... c25=... c26=...
c33=... c34=... c35=... c36=...
c44=... c45=... c46=...
c55=... c56=...
c66=...
cgrad11=... cgrad12=... cgrad13=... cgrad14=... cgrad15=... cgrad16=...
cgrad22=... cgrad23=... cgrad24=... cgrad25=... cgrad26=...
cgrad33=... cgrad34=... cgrad35=... cgrad36=...
cgrad44=... cgrad45=... cgrad46=...
cgrad55=... cgrad56=...
cgrad66=...
```

**Required parameters:**

rho, and a sufficient number of cij to make the $C$ matrix positive definite.
The stiffness matrix defines the relation between stresses and strains using Voigt notation, see for example Carcione [3] for details.

11.4 Topography command [optional]

11.4.1 topography [optional]

Syntax:
topography input=... file=... resolution=... zmax=... order=... smooth=... gaussianAmp=... gaussianXc=... gaussianYc=... gaussianLx=... gaussianLy=...

Required parameters:
input, zmax, file (except when input=gaussian)

Also see discussion below.

The topography command specifies the shape of the free surface boundary, and optionally allows the polynomial order of the grid mapping to be adjusted. The topography is given as elevation (in meters) relative to mean sea level, i.e., positive above sea level and negative below sea level. The curvilinear grid is located between the topography and \( z = z_{\text{max}} \) (recall that \( z \) is directed downwards). If the elevation \( e \) of the topography ranges between \( e_{\text{min}} \leq e \leq e_{\text{max}} \), we recommend using \( z_{\text{max}} \geq -e_{\text{min}} + 2|e_{\text{max}} - e_{\text{min}}| \).

There are five ways of specifying the topography:

- **input=geographic** Read the topography from a file, where the coordinates are given as a function of geographic coordinates (latitude and longitude). The file name must be specified by the file=... keyword. The format for this file is described in Section 12.2.

- **input=cartesian** Read the topography as function of Cartesian coordinates. The file name must be specified by the file=... keyword. The format for this file is described in Section 12.2.

- **input=rfile** Read the topography from a binary raster file. The name of the raster file must be specified by the file=... keyword. The format of this file is described in Section 12.5.

- **input=efile** Read the topography from the Etree database. The database file name must be specified by the file=... keyword. The spatial resolution for querying the Etree database may be specified by the resolution=... keyword.

- **input=gaussian** Build an analytical topography in the shape of a Gaussian hill. The amplitude is specified by gaussianAmp=..., the hill is centered at gaussianXc=..., gaussianYc=..., and the half width of the hill in the \( x \) and \( y \)-directions are specified by gaussianLx=..., and gaussianLy=.... Note that this topography is not smoothed, i.e., the smooth keyword is not used in this case.
### topography command parameters (basic)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>Type of input: geographic (or grid), cartesian, efile or gaussian</td>
<td>string</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>file</td>
<td>File name if input=geographic or input=cartesian</td>
<td>string</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>etree</td>
<td>File name of detailed database if input=efile (same as file)</td>
<td>string</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>xetree</td>
<td>File name of extended etree database if input=efile</td>
<td>string</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>resolution</td>
<td>Resolution for querying the efile if input=efile</td>
<td>real</td>
<td>meters</td>
<td>none</td>
</tr>
<tr>
<td>zmax</td>
<td>z coordinate of the interface between Cartesian and curvilinear grid</td>
<td>real</td>
<td>m</td>
<td>0</td>
</tr>
<tr>
<td>order</td>
<td>Interpolation order (2-6)</td>
<td>int</td>
<td>none</td>
<td>4</td>
</tr>
<tr>
<td>smooth</td>
<td>Number of smoothing iterations of topography grid surface</td>
<td>int</td>
<td>none</td>
<td>10</td>
</tr>
</tbody>
</table>

### topography command parameters (Gaussian Hill)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>gaussianAmp</td>
<td>Amplitude for a Gaussian hill topography</td>
<td>real</td>
<td>meters</td>
<td>0.05</td>
</tr>
<tr>
<td>gaussianXc</td>
<td>x-coordinate of center for a Gaussian Hill</td>
<td>real</td>
<td>meters</td>
<td>0.5</td>
</tr>
<tr>
<td>gaussianYc</td>
<td>y-coordinate of center for a Gaussian Hill</td>
<td>real</td>
<td>meters</td>
<td>0.5</td>
</tr>
<tr>
<td>gaussianLx</td>
<td>Width of the Gaussian hill in the x-direction</td>
<td>real</td>
<td>meters</td>
<td>0.15</td>
</tr>
<tr>
<td>gaussianLy</td>
<td>Width of the Gaussian hill in the y-direction</td>
<td>real</td>
<td>meters</td>
<td>0.15</td>
</tr>
</tbody>
</table>

### 11.4.2 refinement [optional]

Each `refinement` command corresponds to a mesh refinement patch for $z \leq z_{\text{max}}$. The grid size in each refinement patch is half of the next coarser grid size. The grid size in the coarsest grid is prescribed by the `grid` command.

**Syntax:**

`refinement zmax=...`

**Required parameters:**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Unit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>zmax</td>
<td>maximum z-coordinate for the refinement region</td>
<td>real</td>
<td>m</td>
<td>None</td>
</tr>
</tbody>
</table>
11.5 Output commands [optional]

The output commands enable results from the simulation to be saved on file. The rec command saves a time series of the solution at a recording station, which can be read by the SAC program [7] or the readsac.m Matlab script in the tools directory. The image command is used to save a two-dimensional cross-section of the solution, the material properties, or the grid. The image files can be read by the readimage.m Matlab script in the tools directory. The volimage command is used to save three-dimensional volumetric data of the solution, derived quantities of the solution, or the material model. These files are written in a binary format. The readimage3d.m Matlab script in the tools director can read the three-dimensional data into Matlab or Octave. The gmt command outputs a shell script file containing the location of all rec stations and the epicenter, i.e., the location of the source command with the earliest start time. This shell script file can be used for further postprocessing by programs in the GMT suite [21].

11.5.1 rec (or sac) [optional]

The rec command is used to save the time history of the solution at a fixed location in space, see §9.2 for examples. For backwards compatibility with WPP, this command can also be called sac. However, some options of the sac command in WPP have changed names and others are no longer supported.

Syntax:
rec x=... y=... z=... lat=... lon=... depth=... topodepth=... sta=...
file=... writeEvery=... nsew=... usgsformat=... sacformat=... variables=...

Required parameters:
Location of the receiver in Cartesian or geographical coordinates.

The file format is described in Section 12.6

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x position of the receiver</td>
<td>real</td>
<td>m</td>
<td>0.0</td>
</tr>
<tr>
<td>y</td>
<td>y position of the receiver</td>
<td>real</td>
<td>m</td>
<td>0.0</td>
</tr>
<tr>
<td>z</td>
<td>z position of the receiver</td>
<td>real</td>
<td>m</td>
<td>0.0</td>
</tr>
<tr>
<td>lat</td>
<td>latitude geographical coordinate of the receiver</td>
<td>real</td>
<td>degrees</td>
<td>none</td>
</tr>
<tr>
<td>lon</td>
<td>longitude geographical coordinate of the receiver</td>
<td>real</td>
<td>degrees</td>
<td>none</td>
</tr>
<tr>
<td>depth</td>
<td>depth of the receiver (below topography)</td>
<td>real</td>
<td>m</td>
<td>none</td>
</tr>
<tr>
<td>topodepth</td>
<td>depth of the receiver (same as depth)</td>
<td>real</td>
<td>m</td>
<td>none</td>
</tr>
</tbody>
</table>
### sac command parameters (part 2)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>file name</td>
<td>string</td>
<td>none</td>
<td>station</td>
</tr>
<tr>
<td>sta</td>
<td>name of the station</td>
<td>string</td>
<td>none</td>
<td>(same as file)</td>
</tr>
<tr>
<td>writeEvery</td>
<td>cycle interval to write the file to disk</td>
<td>int</td>
<td>none</td>
<td>1000</td>
</tr>
<tr>
<td>usgsformat</td>
<td>output all components in an ASCII text file</td>
<td>int</td>
<td>none</td>
<td>0</td>
</tr>
<tr>
<td>sacformat</td>
<td>output each component in a SAC file</td>
<td>int</td>
<td>none</td>
<td>1</td>
</tr>
<tr>
<td>nsew</td>
<td>output ((x,y,z))-components ((0)) or East, North, and vertical ((-z)) components ((1))</td>
<td>int</td>
<td>none</td>
<td>0</td>
</tr>
<tr>
<td>variables</td>
<td>displacement, velocity, div, curl, strains, or displacementgradient</td>
<td>string</td>
<td>none</td>
<td>displacement</td>
</tr>
</tbody>
</table>

#### 11.5.2 image [optional]

**Syntax:**

```plaintext
image x=... y=... z=... time=... timeInterval=... cycle=... cycleInterval=... file=... mode=... precision=...
```

**Required parameters:**
- Location of the image plane \((x, y, \text{or } z)\)
- Time for output \((\text{time, timeInterval, cycle, or cycleInterval})\)

**Notes:**
- `mode=topo` can only be used when the `topography` command is used.
- \(z=0\) corresponds to the free surface when `topography` is used. It is not possible to output a plane with \(z=\text{const.}\) that cuts through the curvilinear grid. When `topography` is used, any such \(z\)-plane will be output on the free surface.
- The error in the solution can only be calculated in testing mode, i.e., while using `twilight`, `testlamb`, or `testpointsource`.

The image file format is described in Section [12.7](#).

### image command parameters (part 1)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x location of image plane ((\geq 0))</td>
<td>real</td>
<td>m</td>
<td>none</td>
</tr>
<tr>
<td>y</td>
<td>y location of image plane ((\geq 0))</td>
<td>real</td>
<td>m</td>
<td>none</td>
</tr>
<tr>
<td>z</td>
<td>z location of image plane ((\geq 0))</td>
<td>real</td>
<td>m</td>
<td>none</td>
</tr>
</tbody>
</table>
image command parameters (part 2)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>Time-level for outputting image (closest time step) ((\geq 0))</td>
<td>real</td>
<td>s</td>
<td>none</td>
</tr>
<tr>
<td>timeInterval</td>
<td>Time-level interval for outputting a series of images ((&gt; 0))</td>
<td>real</td>
<td>s</td>
<td>none</td>
</tr>
<tr>
<td>cycle</td>
<td>Time-step cycle to output image ((\geq 0))</td>
<td>int</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>cycleInterval</td>
<td>Time-step cycle interval to output a series of images ((\geq 1))</td>
<td>int</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>file</td>
<td>File name header of image</td>
<td>string</td>
<td>none</td>
<td>image</td>
</tr>
<tr>
<td>precision</td>
<td>Floating point precision for saving data ((\text{float or double}))</td>
<td>string</td>
<td>none</td>
<td>float</td>
</tr>
<tr>
<td>mode</td>
<td>The field to be saved</td>
<td>string</td>
<td>none</td>
<td>rho</td>
</tr>
</tbody>
</table>

**mode** can take one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lat</td>
<td>latitude ((\text{in degrees}))</td>
</tr>
<tr>
<td>lon</td>
<td>longitude ((\text{in degrees}))</td>
</tr>
<tr>
<td>topo</td>
<td>elevation of topography (\text{only available with topography})</td>
</tr>
<tr>
<td>grid</td>
<td>grid coordinates in the plane of visualization ((\text{e.g. y-z plane if x=const}))</td>
</tr>
<tr>
<td>gridx</td>
<td>grid \text{x}-coordinates in the plane of visualization</td>
</tr>
<tr>
<td>gridy</td>
<td>grid \text{y}-coordinates in the plane of visualization</td>
</tr>
<tr>
<td>gridz</td>
<td>grid \text{z}-coordinates in the plane of visualization</td>
</tr>
</tbody>
</table>

**mode options (material)**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rho</td>
<td>Density</td>
</tr>
<tr>
<td>lambda</td>
<td>1st Lamé parameter</td>
</tr>
<tr>
<td>mu</td>
<td>2nd Lamé parameter ((\text{shear modulus}))</td>
</tr>
<tr>
<td>p</td>
<td>Compressional wave speed</td>
</tr>
<tr>
<td>s</td>
<td>Shear wave speed</td>
</tr>
<tr>
<td>qp</td>
<td>(Q_P) quality factor</td>
</tr>
<tr>
<td>qs</td>
<td>(Q_S) quality factor</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>ux</td>
<td>displacement in the x-direction</td>
</tr>
<tr>
<td>uy</td>
<td>displacement in the y-direction</td>
</tr>
<tr>
<td>uz</td>
<td>displacement in the z-direction</td>
</tr>
<tr>
<td>div</td>
<td>divergence of (\mathbf{u}) (displacement)</td>
</tr>
<tr>
<td>curl</td>
<td>magnitude of the rotation of (\mathbf{u})</td>
</tr>
<tr>
<td>mag</td>
<td>magnitude of (\mathbf{u})</td>
</tr>
<tr>
<td>hmag</td>
<td>magnitude of ((u^{(x)}, u^{(y)})) (horizontal components)</td>
</tr>
<tr>
<td>hmax</td>
<td>maximum in time of magnitude of ((u^{(x)}, u^{(y)}))</td>
</tr>
<tr>
<td>vmax</td>
<td>maximum in time of (</td>
</tr>
<tr>
<td>divdudt (veldiv)</td>
<td>divergence of (\mathbf{u}_t) (velocity)</td>
</tr>
<tr>
<td>curldudt (velcurl)</td>
<td>magnitude of the rotation of (\mathbf{u}_t)</td>
</tr>
<tr>
<td>magdudt (velmag)</td>
<td>magnitude of (\mathbf{u}_t)</td>
</tr>
<tr>
<td>hmagdudt (hvelmag)</td>
<td>magnitude of ((u_t^{(x)}, u_t^{(y)}))</td>
</tr>
<tr>
<td>hmaxdudt (hvelmax)</td>
<td>maximum in time of magnitude of ((u_t^{(x)}, u_t^{(y)}))</td>
</tr>
<tr>
<td>vmaxdudt (vvelmax)</td>
<td>maximum in time of (</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uxexact</td>
<td>(x)-component of exact solution</td>
</tr>
<tr>
<td>uyexact</td>
<td>(y)-component of exact solution</td>
</tr>
<tr>
<td>uzexact</td>
<td>(z)-component of exact solution</td>
</tr>
<tr>
<td>uxerr</td>
<td>(x)-component of error (difference between computed and exact solution)</td>
</tr>
<tr>
<td>uyerr</td>
<td>(y)-component of error (difference between computed and exact solution)</td>
</tr>
<tr>
<td>uzerr</td>
<td>(z)-component of error (difference between computed and exact solution)</td>
</tr>
</tbody>
</table>

11.5.3 volimage [optional]

Syntax:
```
volimage file=... mode=... precision=... time=... timeInterval=... cycle=...
cycleInterval=... startTime=...
```

Required parameter:
Time for output: (time, timeInterval, cycle, or cycleInterval).

The volimage command in SW4 allows you to save 3D volumetric data on file. These files can be read by the Matlab/Octave script in tools/readimage3d.m. You may also want to use the
open source post processor *VisIt*, to read these files. Be aware that these files can be *very* large. The output occurs at certain time levels during the simulation. The time levels are controlled by the parameters `time=...`, `timeInterval=...`, `cycle=...`, or `cycleInterval=...`, which have the same meaning as in the image command. In addition, the `startTime=...` option can be used in conjunction with `cycleInterval` or `timeInterval` to only output data after a specified time level in the simulation. The options `file=...`, `mode=...`, and `precision=...` have the same meaning as the corresponding parameters in the image command. The set of possible modes, which is different from the image command, is given in the table below. The `volimage` command produces files with extension `.3D.mode.sw4img`, where `mode` is one of `ux`, `uy`, `uz`, `rho`, `lambda`, `mu`, `p`, `s`, `qp`, `qs`.

When topography is present, i.e., the top grid is curvilinear, each file also includes the z-coordinates of the curvilinear grid. Note that the grid coordinates are only saved when the input file contains a `topography` command.

The file format is described in Section [12.8](#).

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Units</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>file</code></td>
<td>file name header of image</td>
<td>string</td>
<td>none</td>
<td>volimage</td>
</tr>
<tr>
<td><code>mode</code></td>
<td>specifies which field is written to the image file</td>
<td>string</td>
<td>none</td>
<td>rho</td>
</tr>
<tr>
<td><code>precision</code></td>
<td>precision of image data on file (float/double)</td>
<td>string</td>
<td>none</td>
<td>float</td>
</tr>
<tr>
<td><code>time</code></td>
<td>simulation time to output image, will be closest depending on dt taken</td>
<td>real</td>
<td>sec.</td>
<td>none</td>
</tr>
<tr>
<td><code>timeInterval</code></td>
<td>simulation time interval to output series of images</td>
<td>real</td>
<td>sec.</td>
<td>none</td>
</tr>
<tr>
<td><code>cycle</code></td>
<td>time-step cycle to output image</td>
<td>int</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td><code>cycleInterval</code></td>
<td>time-step cycle interval to output a series of images</td>
<td>int</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td><code>startTime</code></td>
<td>only output data after this time level (only used with <code>cycleInterval</code> or <code>timeInterval</code>)</td>
<td>real</td>
<td>s</td>
<td>-999.9</td>
</tr>
</tbody>
</table>

The `mode` keyword can have the following values:
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ux</td>
<td>x-components of the solution (displacement)</td>
</tr>
<tr>
<td>uy</td>
<td>y-components of the solution (displacement)</td>
</tr>
<tr>
<td>uz</td>
<td>z-components of the solution (displacement)</td>
</tr>
<tr>
<td>rho</td>
<td>density of material</td>
</tr>
<tr>
<td>p</td>
<td>$C_p$ material velocity</td>
</tr>
<tr>
<td>s</td>
<td>$C_s$ material velocity</td>
</tr>
<tr>
<td>lambda</td>
<td>First Lamé parameter</td>
</tr>
<tr>
<td>mu</td>
<td>Second Lamé parameter</td>
</tr>
<tr>
<td>qp</td>
<td>$Q_p$ material attenuation factor</td>
</tr>
<tr>
<td>qs</td>
<td>$Q_s$ material attenuation factor</td>
</tr>
</tbody>
</table>

### 11.5.4 gmt [optional]

**Syntax:**
gmt file=...

**Required parameters:**
None.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>name of output file for gmt c-shell commands</td>
<td>string</td>
<td>sw4.gmt.csh</td>
</tr>
</tbody>
</table>

### 11.6 SW4 testing commands [optional]

#### 11.6.1 twilight

The **twilight** command runs $SW^4$ in a testing mode where forcing functions are constructed to create a known smooth analytical solution, see Appendix A.1 for details.

**Syntax:**
twilight errorlog=... omega=... c=... phase=... momega=... mphase=... amprho=... ampmu=... amplambda=...  

**Required parameters:**
None
### twilight command parameters

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorlog</td>
<td>Outputs error log in file twilight_errors.dat</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>omega</td>
<td>Wave number in exact solution</td>
<td>real</td>
<td>1.0</td>
</tr>
<tr>
<td>c</td>
<td>Phase speed in exact solution</td>
<td>real</td>
<td>1.3</td>
</tr>
<tr>
<td>phase</td>
<td>Solution phase coefficient</td>
<td>real</td>
<td>0.0</td>
</tr>
<tr>
<td>momega</td>
<td>Wave number in material</td>
<td>real</td>
<td>1.0</td>
</tr>
<tr>
<td>mphase</td>
<td>Material phase coefficient</td>
<td>real</td>
<td>0.4</td>
</tr>
<tr>
<td>amrho</td>
<td>Density amplitude</td>
<td>real</td>
<td>1.0</td>
</tr>
<tr>
<td>ampmu</td>
<td>Material $\mu$ amplitude</td>
<td>real</td>
<td>1.0</td>
</tr>
<tr>
<td>amplambda</td>
<td>Material $\lambda$ amplitude</td>
<td>real</td>
<td>1.0</td>
</tr>
</tbody>
</table>

---

11.6.2 testlamb

The testlamb command solves Lamb’s problem, i.e., the displacement due to a vertical point forcing on a flat free surface, see Appendix A.2 for details.

**Syntax:**

testlamb x=... y=... cp=... rho=... fz=...

**Required parameters:**
Location of the forcing $(x, y)$.

### testlamb command parameters

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>$x$-coordinate of point source</td>
<td>real</td>
<td>0.0</td>
</tr>
<tr>
<td>y</td>
<td>$y$-coordinate of point source</td>
<td>real</td>
<td>0.0</td>
</tr>
<tr>
<td>cp</td>
<td>P-wave velocity</td>
<td>real</td>
<td>1.0</td>
</tr>
<tr>
<td>rho</td>
<td>Density</td>
<td>real</td>
<td>1.0</td>
</tr>
<tr>
<td>fz</td>
<td>Magnitude of the forcing</td>
<td>real</td>
<td>1.0</td>
</tr>
</tbody>
</table>

---

11.6.3 testpointsource

The testpointsource command calculates the displacement due to a point source in a homogeneous whole space, and computes the error. The source properties are specified by a regular source command. However, there must be exactly one source command and, the time function must be one of the types VerySmoothBump, C6SmoothBump, SmoothWave, or Gaussian.

Before the solution has reached the a boundary or the super-grid layers, this test evaluates the discretization of the source term. If supergrid layers are used on all six boundaries, this test can also be used to evaluate the accuracy of the supergrid far field layers.

By setting diractest=1, SW4 checks the moment conditions of the source discretization. See the source code for further details.
Syntax:

testpointsource cp=... cs=... rho=... diractest=...

Required parameters:
None

<table>
<thead>
<tr>
<th>testpointsource command parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
</tr>
<tr>
<td>cp</td>
</tr>
<tr>
<td>cs</td>
</tr>
<tr>
<td>rho</td>
</tr>
<tr>
<td>diractest</td>
</tr>
</tbody>
</table>

11.6.4 testrayleigh

The testrayleigh command runs a surface wave through the domain with periodic boundary conditions in the horizontal plane. The file RayleighErr.txt with information about the error in the computation after each time step is produced. The file has four columns where the first column is the time, the second column is the maximum norm of the error, the third column is the $L^2$ norm of the error, and the fourth column is the maximum norm of the solution.

Syntax:

testrayleigh cp=... cs=... rho=... nwl=...

Required parameters:
None

<table>
<thead>
<tr>
<th>testrayleigh command parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
</tr>
<tr>
<td>cp</td>
</tr>
<tr>
<td>cs</td>
</tr>
<tr>
<td>rho</td>
</tr>
<tr>
<td>nwl</td>
</tr>
</tbody>
</table>

11.6.5 testenergy

The numerical scheme used by SW4 can be proven to be energy conserving, see [20]. The testenergy command verifies the implementation of the scheme by checking the energy conservation. Testenergy sets the material speeds, $C_P, C_S$, and the density to be completely random (but positive). The initial data is also set to a random field. By default the boundary conditions are periodic in the $x$- and $y$-directions, with free-surface conditions at $z = 0$ and homogeneous Dirichlet conditions at $z = z_{max}$. The boundary conditions can be changed by the keyword described in Section 11.7.1. The file energy.log is produced, containing the energy after each time step.

Syntax:
testenergy command parameters

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpcsratio</td>
<td>P-wave velocity to S-wave velocity ratio</td>
<td>real</td>
<td>$\sqrt{3}$</td>
</tr>
<tr>
<td>seed</td>
<td>Pseudo-random number generator seed</td>
<td>int</td>
<td>2934839</td>
</tr>
<tr>
<td>writeEvery</td>
<td>frequency for saving log file to disk</td>
<td>int</td>
<td>1000</td>
</tr>
<tr>
<td>filename</td>
<td>name of log file</td>
<td>string</td>
<td>energy.log</td>
</tr>
</tbody>
</table>

11.7 Advanced simulation controls [optional]

WARNING! The commands in this section are only intended for advanced users who are intimately familiar with the inner workings of SW4. These commands might lead to unexpected side effects. Only the source code gives a complete description of what these commands really do.

11.7.1 boundary_conditions [optional]

Syntax:
boundary_conditions lx=... hx=... ly=... hy=... lz=... hz=...

Required parameters:
None

Note that the boundary condition values are different in SW4 and WPP. The stress-free boundary condition can only be used on the upper ($z = 0$) and lower ($z = z_{max}$) sides.

<table>
<thead>
<tr>
<th>Boundary conditions parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>lx</td>
</tr>
<tr>
<td>hx</td>
</tr>
<tr>
<td>ly</td>
</tr>
<tr>
<td>hy</td>
</tr>
<tr>
<td>lz</td>
</tr>
<tr>
<td>hz</td>
</tr>
</tbody>
</table>
### boundary condition values

<table>
<thead>
<tr>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Stress-free boundary</td>
</tr>
<tr>
<td>1</td>
<td>Dirichlet boundary</td>
</tr>
<tr>
<td>2</td>
<td>Supergrid boundary</td>
</tr>
<tr>
<td>3</td>
<td>Periodic boundary</td>
</tr>
</tbody>
</table>

#### 11.7.2 developer [optional]

Warning: you need to be intimately familiar with the inner workings of SW4 to use this command. Look in the source code to get a full understanding of what this command really does.

**Syntax:**

developer cfl=... checkfornan=...

**Required parameters:**

None

By setting the checkfornan keyword to **on** or **yes**, SW4 scans the solution arrays for floating point exceptions and other errors resulting in NaN (Not a Number). The check is performed after each time step.

### developer parameters

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfl</td>
<td>CFL number ($&gt;0$)</td>
<td>real</td>
<td>1.3</td>
</tr>
<tr>
<td>checkfornan</td>
<td>Scan solution arrays for NaN</td>
<td>string</td>
<td>off</td>
</tr>
</tbody>
</table>
Chapter 12

File formats

12.1 Discrete time function

The discrete time function interpolates values on a uniform grid in time, \( \tau_j = t_0 + (j - 1)\delta_t \), \( j = 1, 2, \ldots, N_d \). The file is formatted. The first line of the file contains the reference time \( t_0 \), time step \( \delta_t \), and number of data points \( N_d \). The subsequent \( N_d \) lines in the file should contain the function values \( g_j = g(\tau_j) \). The file should follow the following format:

<table>
<thead>
<tr>
<th>Line</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( t_0 ) (real)</td>
<td>( \delta_t ) (real)</td>
<td>( N_d ) (integer)</td>
</tr>
<tr>
<td>2</td>
<td>( g_1 ) (real)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( g_2 ) (real)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( N_d + 1 )</td>
<td>( g_{N_d} ) (real)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The time step must be positive, \( \delta_t > 0 \), and at least seven data points must be given, \( N_d \geq 7 \).

12.2 Topography

Topography is specified as elevation above mean sea level on a regular lattice in the horizontal plane. There are two variants of the topography format: geographic or Cartesian. By default, topography is specified as function of geographic coordinates in the horizontal plane. Alternatively, the lattice can be specified in Cartesian coordinates. In both cases, the unit for elevation is meters and the topography file must cover the entire horizontal extent of the computational domain.

12.2.1 Topography on a geographic lattice

Latitude and longitude should be given in degrees. Let the elevation be known at longitudes \( \phi_i, \quad i = 1, 2, \ldots, N_{lon}, \)

and latitudes \( \theta_j, \quad j = 1, 2, \ldots, N_{lat}, \)
Note that the latitudes and the longitudes must either be strictly increasing or strictly decreasing, but the step size may vary.

The elevation should be given on the regular lattice

\[ e_{i,j} = \text{elevation at longitude } \phi_i, \text{ latitude } \theta_j. \]

Bi-cubic interpolation is used to define the elevation in between the lattice points.

The topography file should be an ASCII text file with the following format. The first line of the file holds the number of longitude and latitude data points:

\[ N_{\text{lon}} \quad N_{\text{lat}} \]

On subsequent lines, longitude, latitude and elevation values are given in column first ordering:

\[ \begin{array}{ccc}
\phi_1 & \theta_1 & e_{1,1} \\
\phi_2 & \theta_1 & e_{2,1} \\
\vdots & \vdots & \vdots \\
\phi_{N_{\text{lon}}} & \theta_1 & e_{N_{\text{lon}},1} \\
\vdots & \vdots & \vdots \\
\phi_1 & \theta_{N_{\text{lat}}} & e_{1,N_{\text{lat}}} \\
\phi_2 & \theta_{N_{\text{lat}}} & e_{2,N_{\text{lat}}} \\
\vdots & \vdots & \vdots \\
\phi_{N_{\text{lon}}} & \theta_{N_{\text{lat}}} & e_{N_{\text{lon}},N_{\text{lat}}} 
\end{array} \]

### 12.2.2 Topography on a Cartesian lattice

Cartesian coordinates should be given in meters (\(m\)). Let the elevation be known at \(x\)-coordinates

\[ x_i, \quad i = 1, 2, \ldots, N_x, \]

and \(y\)-coordinates

\[ y_j, \quad j = 1, 2, \ldots, N_y, \]

Note that the coordinate vectors must either be strictly increasing or strictly decreasing, but the step size may vary. Also note that the step size can be different from the step size in the computational grid. To guarantee that the topography grid covers the entire horizontal extent of the computational domain, we require

\[
\min_i x_i \leq 0, \quad \min_j y_j \leq 0, \quad \max_i x_i \geq x_{\text{max}}, \quad \max_j y_j \geq y_{\text{max}},
\]

where \(x_{\text{max}}\) and \(y_{\text{max}}\) are defined by Equation (3.3). Bi-cubic interpolation is used to define the elevation in between the lattice points.

The elevation should be given on the regular lattice

\[ e_{i,j} = \text{elevation at Cartesian coordinate } (x, y) = (x_i, y_j). \]
The topography file should be an ASCII text file with the following format. The first line of the file holds the number of data points in each direction:

\[ Nx \quad Ny \]

On subsequent lines, \(x\), \(y\) and elevation values are given in column first ordering:

\[
\begin{align*}
&x_1 \quad y_1 \quad \varepsilon_{1,1} \\
&x_2 \quad y_1 \quad \varepsilon_{2,1} \\
&\vdots \quad \vdots \quad \vdots \\
&x_{Nx} \quad y_1 \quad \varepsilon_{Nx,1} \\
&\vdots \quad \vdots \quad \vdots \\
&x_1 \quad y_{Ny} \quad \varepsilon_{1,Ny} \\
&x_2 \quad y_{Ny} \quad \varepsilon_{2,Ny} \\
&\vdots \quad \vdots \quad \vdots \\
&x_{Nx} \quad y_{Ny} \quad \varepsilon_{Nx,Ny}
\end{align*}
\]

12.3 pfile

There are two variants of the pfile format: geographic or Cartesian. By default, geographic coordinates are used to specify the location of the depth profiles in the horizontal plane. Alternatively, the lattice can be specified in Cartesian coordinates. Note that different units are used in the two cases. Pfiles are ASCII text formatted.

12.3.1 pfile on a geographic lattice

The header has 7 lines and follows the following format:

<table>
<thead>
<tr>
<th>Line</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(\Delta) [deg] (real)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(N_{lat}) (integer)</td>
<td>(Lat_{min}) [deg] (real)</td>
<td>(Lat_{max}) [deg] (real)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(N_{lon}) (integer)</td>
<td>(Lon_{min}) [deg] (real)</td>
<td>(Lon_{max}) [deg] (real)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>(N_{dep}) (integer)</td>
<td>(d_{min}) [km] (real)</td>
<td>(d_{max}) [km] (real)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>(I_{sed}) (integer)</td>
<td>(I_{MoHo}) (integer)</td>
<td>(I_{410}) (integer)</td>
<td>(I_{660}) (integer)</td>
</tr>
<tr>
<td>7</td>
<td>(Q)-available? (logical)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The first line holds the optional name of the material model. Line 2 contains the parameter \(\Delta\), which is used to average the material properties according to equation (6.1). Lines 3 and 4 contain the number of lattice points as well as the starting and ending angles in the latitude and longitude direction, respectively. Line 5 contains the number of depth values in each profile, followed by the minimum and maximum depth measured in km. Line 6 supplies optional information about the
The index of some material discontinuities in each depth profile. Give -99 if not known. Note that the index for each discontinuity (sediment, MoHo, 410, 660) indicates the row number within each profile, for the material property just above the discontinuity. Hence, the subsequent entry in each profile should have the same depth value and contain the material property just below the same discontinuity. Line 7 should contain the single letter 'T' or 't' if the subsequent data contains quality factors ($Q_P$ and $Q_S$); otherwise it should contain the single letter 'F' or 'f'. The presence of quality factors may alternatively be indicated by using the strings '.TRUE.', '.true.', '.FALSE.', or '.false.'.

The first seven lines of a pfile can look like this:

Caucasus
0.25
7 38.00 39.50
19 44.50 49.00
30 0.00 161.00
-99 -99 -99 -99
TRUE.

The header is directly followed by $N_{lat} \times N_{lon}$ depth profiles, ordered such that the longitude varies the fastest, that is, according to the pseudo-code:

   for ($Lat_i = Lat_{min}; Lat_i <= Lat_{max}; Lat_i = \Delta_{lat}$)
       for ($Lon_j = Lon_{min}; Lon_j <= Lon_{max}; Lon_j = \Delta_{lon}$)
           (save depth profile for $Lat_i$, $Lon_j$)
       end
   end

Here, $\Delta_{lat} = (Lat_{max} - Lat_{min})/(N_{lat} - 1)$ and $\Delta_{lon} = (Lon_{max} - Lon_{min})/(N_{lon} - 1)$. In general, $\Delta_{lat} \neq \Delta_{lon} \neq \Delta$.

The first line of each depth profile holds the latitude and longitude (in degrees as real numbers), and the number of depth values, which must equal $N_{dep}$. For example a depth profile for latitude 33.108, longitude -115.66, with $N_{dep} = 19$ points in the depth direction starts with the line

33.108 -115.66 19

The subsequent $N_{dep}$ lines have the following format:

<table>
<thead>
<tr>
<th>Index (int)</th>
<th>depth [km]</th>
<th>$C_p$ [km/s]</th>
<th>$C_s$ [km/s]</th>
<th>$\rho$ [g/cm$^3$]</th>
<th>$Q_P$</th>
<th>$Q_S$</th>
</tr>
</thead>
</table>

Note that $Q_P$ and $Q_S$ should only be present when indicated so by the $Q$-availability flag on line 7 of the header. Also note that the units are different than in other parts of SW4. In particular, $C_P$ and $C_S$ should be given in km/s = 1000 m/s, and density ($\rho$) should be given in g/cm$^3$ = 1000 kg/m$^3$.

### 12.3.2 pfile on a Cartesian lattice

The header of the Cartesian grid pfile format consists of seven lines with the following information:
Table 1: Structure of the Material Surface File

<table>
<thead>
<tr>
<th>Line</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Name (string)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(h \text{ [m]}) (real)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(N_x) (integer)</td>
<td>(x_{\text{min}} \text{ [m]}) (real)</td>
<td>(x_{\text{max}} \text{ [m]}) (real)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(N_y) (integer)</td>
<td>(y_{\text{min}} \text{ [m]}) (real)</td>
<td>(y_{\text{max}} \text{ [m]}) (real)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>(N_{dep}) (integer)</td>
<td>(d_{\text{min}} \text{ [m]}) (real)</td>
<td>(d_{\text{max}} \text{ [m]}) (real)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>(I_{\text{sed}}) (integer)</td>
<td>(I_{\text{MoHo}}) (integer)</td>
<td>(I_{410}) (integer)</td>
<td>(I_{660}) (integer)</td>
</tr>
<tr>
<td>7</td>
<td>Q-available? (logical)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This is essentially the same header as for the geographic coordinate format, with the only difference that information on lines 2, 3, and 4 is different. The spacing, \(h\), of the grid of depth profiles is given on line 2. The number of depth profiles in the \(x\)-direction \(N_x\) with minimum and maximum coordinate values are given on line 3. The same quantities for the \(y\)-direction are given on line 4. Note that all distances, including the depth information on line 5, must be given in meters (\([\text{m}]\)).

The header is directly followed by \(N_x \times N_y\) depth profiles, ordered such that the \(x\)-coordinate varies the fastest, that is, according to the pseudo-code:

\[
\begin{align*}
&\text{for } (y = y_{\text{min}}; y <= y_{\text{max}}; y+ = h) \\
&\quad \text{for } (x = x_{\text{min}}; x <= x_{\text{max}}; x+ = h) \\
&\quad \quad \text{(save depth profile for } x, y) \\
&\quad \text{end} \\
&\text{end}
\end{align*}
\]

The first line of each depth profile holds the \(x\)-coordinate and the \(y\)-coordinate (in meters as real numbers), and the number of depth values, which must equal \(N_{dep}\). For example a depth profile for \(x = 100.4 \text{ m}\) and \(y = 30.6 \text{ m}\), with \(N_{dep} = 19\) points in the depth direction starts with the line

100.4 30.6 19

The subsequent \(N_{dep}\) lines have the following format:

<table>
<thead>
<tr>
<th>Index (int)</th>
<th>depth [m]</th>
<th>(C_p) [m/s]</th>
<th>(C_s) [m/s]</th>
<th>(\rho) [kg/m(^3)]</th>
<th>(Q_P)</th>
<th>(Q_S)</th>
</tr>
</thead>
</table>

\(Q_P\) and \(Q_S\) can be left out when indicated not present by the Q-availability flag on line 7 of the header. Note that, unlike the pfils on a geographic lattice, the units should here be the standard MKS units, which normally are used in \(SW4\).

12.4 ifile

The material surface file (ifile) should be an ASCII text file with the following format. We start by assuming the material surfaces are given as function of geographic coordinates (input=geographic). Modifications for the Cartesian case are described at the end of this section.

The first line of the file holds the number of longitude and latitude data points, as well as the number of material surfaces:

\(N_{lon} \ N_{lat} \ N_{mat}\)
On subsequent lines, longitude, latitude and $N_{\text{mat}}$ surface depth values are given in column first ordering:

\[
\begin{align*}
\text{Lon}_1 & \quad \text{Lat}_1 & \quad d_{1,1,1} & \quad \ldots & \quad d_{N_{\text{mat}},1,1} \\
\text{Lon}_2 & \quad \text{Lat}_1 & \quad d_{1,2,1} & \quad \ldots & \quad d_{N_{\text{mat}},2,1} \\
\vdots & \quad \vdots & \quad \vdots & \quad \quad \vdots & \quad \vdots \\
\text{Lon}_{N_{\text{lon}}} & \quad \text{Lat}_1 & \quad d_{1,N_{\text{lon}},1} & \quad \ldots & \quad d_{N_{\text{mat}},N_{\text{lon}},1} \\
\vdots & \quad \vdots & \quad \vdots & \quad \quad \vdots & \quad \vdots \\
\text{Lon}_1 & \quad \text{Lat}_{N_{\text{lat}}} & \quad d_{1,1,N_{\text{lat}}} & \quad \ldots & \quad d_{N_{\text{mat}},1,N_{\text{lat}}} \\
\text{Lon}_2 & \quad \text{Lat}_{N_{\text{lat}}} & \quad d_{1,2,N_{\text{lat}}} & \quad \ldots & \quad d_{N_{\text{mat}},2,N_{\text{lat}}} \\
\vdots & \quad \vdots & \quad \vdots & \quad \quad \vdots & \quad \vdots \\
\text{Lon}_{N_{\text{lon}}} & \quad \text{Lat}_{N_{\text{lat}}} & \quad d_{1,N_{\text{lon}},N_{\text{lat}}} & \quad \ldots & \quad d_{N_{\text{mat}},N_{\text{lon}},N_{\text{lat}}} 
\end{align*}
\]

It is required that $0 \leq d_{q,i,j} \leq d_{q+1,i,j}$.

### 12.4.1 Cartesian ifile

When the material surfaces are given as function of Cartesian coordinates (input=cartesian), the first line of the file holds the number of data points in the “x” and “y” directions, as well as the number of material surfaces:

\[
N_x \quad N_y \quad N_{\text{mat}}
\]

The subsequent lines have essentially the same format as in the geographic case. In the above description, simply substitute “longitude” by “x” and “latitude” by “y”.

### 12.5 rfile

The rfile starts with a header followed by a data section. The header has two parts. The first part contains 5 integers (4 bytes each), 3 doubles (8 bytes each) and a character array with a variable number of elements (1 byte each):

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Name</th>
<th>Type</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>magic</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>prec</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>att</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>az</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>20</td>
<td>lon0</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>28</td>
<td>lat0</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>36</td>
<td>mlen</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>40</td>
<td>merestr</td>
<td>char</td>
<td>mlen</td>
</tr>
<tr>
<td>40+mlen</td>
<td>$N_b$</td>
<td>int</td>
<td>4</td>
</tr>
</tbody>
</table>
Here, magic = 1 is used to determine the byte ordering on the file, i.e., if it was written by a machine using big- or little-endianess. prec is the number of bytes per entry in the data section, i.e., 4 for single precision and 8 for double precision. The flag att can currently be 0 or 1, and indicates whether the visco-elastic attenuation parameters $Q_P$ and $Q_S$ are included in the data section. The grid azimuth $az$ gives the angle [deg] between North and the positive $x$-axis. The geographical coordinates of the origin of the data in the horizontal plane is ($lon0$, $lat0$). The number of characters in the string mercstr is $mlen$, and the number of blocks (patches) in the data section is given by $N_b$.

The second part of the header contains 40 bytes of grid size information for each of the $N_b$ grid patches. The information is saved in the following format:

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Name</th>
<th>Type</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$o_0 + 40(b - 1)$</td>
<td>$hh_b$</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>$o_0 + 8 + 40(b - 1)$</td>
<td>$hv_b$</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>$o_0 + 16 + 40(b - 1)$</td>
<td>$z0_b$</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>$o_0 + 24 + 40(b - 1)$</td>
<td>$nc_b$</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>$o_0 + 28 + 40(b - 1)$</td>
<td>$ni_b$</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>$o_0 + 32 + 40(b - 1)$</td>
<td>$nj_b$</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>$o_0 + 36 + 40(b - 1)$</td>
<td>$nk_b$</td>
<td>int</td>
<td>4</td>
</tr>
</tbody>
</table>

Here, $o_0 = 44 + mlen$, is the size (in bytes) of the first header section. $hh_b$ and $hv_b$ are the grid sizes in the horizontal and vertical directions, respectively. $z0_b$ is the base $z$-level for block $b$. The number of components in block $b$ is $nc_b$. Block number $b$ has $ni_b \times nj_b \times nk_b$ grid points in the $(i, j, k)$ directions, respectively. Note that the first block is assumed to hold the elevation of the topography/bathymetry, so $z0_1$ is not used.

The Cartesian coordinates $(x_i, y_j, z_k)$ of grid point $(i, j, k)$ in patch $b$ satisfy

$$x_i = hh_b(i - 1), \quad y_j = hh_b(j - 1), \quad z_k = z0_b + (k - 1)hv_b.$$  

It is important to notice that the vertical coordinate $z_k$ is interpreted as the depth below mean sea level, i.e., it has the same meaning as the vertical coordinate of the computational mesh. This means that some grid points in an rfile will be located above the topography/bathymetry.

The data section holds the content of the four-dimensional arrays $a_b(nc_b, ni_b, nj_b, nk_b)$ for each grid patch $1 \leq b \leq N_b$. Depending on the value of prec, each element is saved as a 4 byte float, or an 8 byte double. To enable better parallel reading performance, the grid points are traversed in “C”-order. The data blocks can be read by the pseudo-C code

```c
for (b = 1; b \leq N_b; b++)
    for (i = 1; i \leq ni_b; i++)
        for (j = 1; j \leq nj_b; j++)
            for (k = 1; k \leq nk_b; k++)
                for (c = 1; c \leq nc_b; c++)
                    read $a_b(c, i, j, k)$
```

98
The elevation of the topography/bathymetry is always stored in the first block. Note that it is defined to be positive above mean sea level. For $b \geq 2$, the material properties are stored in the following order,

$$ a_b(c, i, j, k) = \begin{cases} 
\rho, & c = 1, \\
V_P, & c = 2, \\
V_S, & c = 3, \\
Q_P, & c = 4 \text{ (if } \text{att} = 1), \\
Q_S, & c = 5 \text{ (if } \text{att} = 1), 
\end{cases} \quad b \geq 2. $$

If a grid point is above the topography (i.e., in the air), you should set $\rho = V_P = V_S = -999$, and $Q_P = Q_S = -999$ (if $\text{att} = 1$). If a grid point is above the bathymetry (i.e., in water), you should set $V_S = -999$, but give physical values to $V_P$ and $\rho$. Note that wave propagation in water is currently not modeled by SW4 so those values are not currently used. However, setting correct density and compressional wave speeds in water will allow the same material model to be used when such modeling becomes available. It is also consistent with how water is handled in USGS’s etree model of the San Francisco bay area.

A Matlab/octave reader of material models in the rfile format, called readmat.m, is provided in the tools directory. A sample rfile model is included in the examples/rfile directory.

Restrictions and assumptions.

- The azimuth in the rfile must agree with the azimuth given in the grid command.

- The origin of the rfile does not have to agree with the origin of the computational grid. In fact, the setup of a simulation is less prone to errors if the rfile domain is at least a few percent larger than the extent of the computational domain.

- The first block holds the elevation of the topography/bathymetry, so $n_{c1} = 1$. The following blocks must have either 3 or 5 components, i.e.,

$$ n_{cb} = \begin{cases} 
1, & b = 1, \\
3, & \text{att} = 0 \text{ and } b \geq 2, \\
5, & \text{att} = 1 \text{ and } b \geq 2. 
\end{cases} $$

- Because the topography/bathymetry is a function of the horizontal coordinates, the first block must have $n_{k1} = 1$.

- All patches must have the same horizontal extent, i.e.,

$$(ni_b - 1)hh_b = \text{const.}, \quad (nj_b - 1)hh_b = \text{const.}, \quad b \geq 1,$$

and the z-coordinate of the last grid point in one block must match the first one of the next block,

$$z_{0b} + (nk_b - 1)hv_b = z_{0b+1}, \quad b = 2, 3, \ldots, N_b - 1.$$ 

However, neither $z_{01}$ nor $hv_1$ are used when parsing the rfile.

- It is assumed that the horizontal grid size for the topography/bathymetry is the same as the first material block,

$$hh_1 = hh_2, \quad ni_1 = ni_2, \quad nj_1 = nj_2.$$
12.6 sac

SAC files hold the time history of one component of the solution at a fixed point in space. A detailed description of the SAC format can be found at [http://www.iris.edu/manuals/sac/manual.html](http://www.iris.edu/manuals/sac/manual.html), and we refer to that web page for a detailed description of the file format.

We provide a simplified Matlab/octave reader of SAC files called `readsac.m` in the tools directory. Note that only a subset of the header information is parsed by this reader.

12.7 image

**Important note:** The image file format used by SW4 is different from that used by WPP. To emphasize the change in format, all SW4 image files have extension `.sw4img`. A reader for image files is provided in the matlab/octave function `readimage.m` in the tools directory.

Images files hold cross-sectional data on a composite grid and are written in a binary format. The image file starts with a header followed by a data section, which contains a two-dimensional grid function for each grid patch in the composite grid. The header has two parts. The first part contains 5 integers (4 bytes each), 2 doubles (8 bytes each) and a character array with 25 elements (1 byte each), i.e., 61 bytes of data:

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Name</th>
<th>Type</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>prec</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>N_p</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>t</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>plane</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>ξ</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>28</td>
<td>mode</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>32</td>
<td>grdinfo</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>36</td>
<td>creation-time</td>
<td>char[25]</td>
<td>25</td>
</tr>
</tbody>
</table>

Here, `prec` is the number of bytes per entry in the data section, i.e., 4 for single precision and 8 for double precision. The positive number of data patches is stored in `N_p`, and `t ≥ 0` is the simulation time at which the image was saved. The `plane` variable indicates the orientation of the plane. It is 0 for a constant `x`-plane, 1 for a constant `y`-plane, and 2 for a constant `z`-plane. The coordinate value is stored in `ξ`. For example, if `plane`=0, the data is saved along `x = ξ`. The type of data is saved in `mode`. The following 32 different types of data can be saved on an image file:

<table>
<thead>
<tr>
<th>mode</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td><code>u(x)</code></td>
<td><code>u(y)</code></td>
<td><code>u(z)</code></td>
<td><code>ρ</code></td>
<td><code>λ</code></td>
<td><code>μ</code></td>
<td><code>C_p</code></td>
<td><code>C_s</code></td>
<td><code>u_{exx}</code></td>
<td><code>u_{ey}</code></td>
<td><code>div(u)</code></td>
<td>`</td>
<td>curl(u)</td>
</tr>
<tr>
<td>mode</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td><code>div(u)</code></td>
<td>`</td>
<td>curl(u)</td>
<td>`</td>
<td><code>lat</code></td>
<td><code>lon</code></td>
<td><code>topo</code></td>
<td><code>x</code></td>
<td><code>y</code></td>
<td><code>z</code></td>
<td><code>u_{error}</code></td>
<td><code>u_{error}</code></td>
<td><code>u_{error}</code></td>
</tr>
<tr>
<td>mode</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>`</td>
<td>u_t</td>
<td>`</td>
<td>`</td>
<td>u_l</td>
<td>`</td>
<td><code>max_t (u_{lxxx})^2 + (u_{lyy})^2</code></td>
<td><code>max_t (u_{lxx})^2 + (u_{lyy})^2</code></td>
<td>`max_t</td>
<td>u_{lzz}</td>
<td>`</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Here, $|a|$ denotes the magnitude of the vector $a \in \mathbb{R}^3$ and $u_{ex}$ denotes the exact solution, which also is needed to calculate the error in the solution, $u_{error}$. Note that image modes 9, 10, 11, 22, 23, and 24, only are available when the exact solution is known, i.e., when SW4 is run in one of its test modes. Also note that image modes 19-21 store the coordinates of the grid in the data section of the image file. These modes provide an alternative way of saving the grid when it is curvilinear.

The `grdinfo` variable can have values 0 or 1. No grid information is saved on the image file if `grdinfo=0`. In this case all data patches are on a Cartesian grid, which can be reconstructed from the information in the header of the image file (see below). If `grdinfo=1`, the z-coordinates of the curvilinear grid are saved in an additional grid patch, following the last data patch. The final field in the first part of the header is a character array with 25 elements. It holds the creation time of the image file, as obtained from the C++ function `localtime()`. On a Mac running OSX, the time has the format “Thu Feb 28 14:24:07 2013”. Exactly 25 characters are saved and the string is truncated if it is longer than that.

The second part of the header contains grid size information (32 bytes) for each of the $N_p$ data patches. The information is saved in the following format:

<table>
<thead>
<tr>
<th>Offset (bytes)</th>
<th>Name</th>
<th>Type</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>61 + 32(p − 1)</td>
<td>$h_p$</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>69 + 32(p − 1)</td>
<td>$zmin_p$</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>77 + 32(p − 1)</td>
<td>$ib_p$</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>81 + 32(p − 1)</td>
<td>$ni_p$</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>85 + 32(p − 1)</td>
<td>$jb_p$</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>89 + 32(p − 1)</td>
<td>$nj_p$</td>
<td>int</td>
<td>4</td>
</tr>
</tbody>
</table>

Here $h_p$ is the grid size and $zmin_p$ is the starting value of the z-coordinate in patch $p$. There are $ni_p$ by $nj_p$ data points on patch $p$ with starting indices $ib_p$ and $jb_p$. Currently, $ib_p = 1$ and $jb_p = 1$. The Cartesian grid can be constructed from the grid size information,
The header is followed by grid function data on each of the $N_p$ patches. The data is saved as floats ($\text{prec}=4$), or doubles ($\text{prec}=8$). Each patch of the data is stored as in a two-dimensional array, $u^{(p)}(i,j)$, where $ib_p \leq i \leq ib_p + ni_p - 1$ and $jb_p \leq j \leq jb_p + nj_p - 1$. The data can be read as outlined by the following pseudo code:

\[
\text{for (} p = 1; p \leq N_p; p++ \text{)} \{
    \text{for (} j = 1; j \leq nj_p; j++ \text{)} \{
        \text{for (} i = 1; i \leq ni_p; i++ \text{)} \{
            \text{read } u^{(p)}(i-1+ib_p, j-1+jb_p)
        \}
    \}
\}
\]

If $\text{grdinfo}=0$, there is no additional information on the image file. If $\text{grdinfo}=1$, the $z$-coordinates of the curvilinear grid are also saved on the image file (as floats or doubles depending on that value of $\text{prec}$). Note that the curvilinear data always corresponds to patch number $p = N_p$. The $z$-coordinates corresponding to the data are stored as an additional patch, which can be read using the pseudo code:

\[
\text{p} = N_p;
\text{for (} j = 1; j \leq nj_p; j++ \text{)} \{
    \text{for (} i = 1; i \leq ni_p; i++ \text{)} \{
        \text{read } z^{(p)}(i-1+ib_p, j-1+jb_p)
    \}
\}
\]

The dimensions of the arrays $u^{(N_p)}$ and $z^{(N_p)}$ are always the same. When $\text{grdinfo}=1$, the $z^{(N_p)}_{i,j}$ coordinates replace the above definition of the grid for the cases $\text{plane}=0$ and $\text{plane}=1$. For $\text{plane}=2$, the grid is defined by the $(x,y)$ coordinates, which currently remains Cartesian also for a curvilinear grid.

### 12.8 volimage

The `volimage` command generates (often very large) binary files holding three-dimensional volumetric data. A reader for `volimage` files is provided in the matlab/octave script `readimage3d.m` in the tools directory. Alternatively, users might want to visualize the data in these files with the open source VisIt post processor. `volimage` files can also be used to define the material properties in SW4, using the `vimaterial` command. SW4 `volimage` files have extension `.3D.mode.sw4img`, where `mode` is one of `ux`, `uy`, `uz`, `rho`, `lambda`, `mu`, `p`, `s`, `qp`, `qs`.

The format of the `volimage` files is essentially the same as the format of image files, with the exception that the sizes of the data patches are specified by six integers in the `volimage` files, instead of four in the image file format. The first 61 bytes of the header of the `volimage` format is identical to the image format given in the previous section, i.e.,
The plane and coordinate (ξ) variables are not needed for three-dimensional data. They are both set to -1 in a volimage file. The number of possible variables to save is more restricted than for the image command. The volume image format stores one of the following variables with corresponding integer code for the mode variable in the file header:

<table>
<thead>
<tr>
<th>mode</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>$u(x)$</td>
<td>$u(y)$</td>
<td>$u(z)$</td>
<td>$\rho$</td>
<td>$\lambda$</td>
<td>$\mu$</td>
<td>$C_p$</td>
<td>$C_s$</td>
<td>$Q_p$</td>
<td>$Q_s$</td>
</tr>
</tbody>
</table>

After the 61 byte header follows the dimensional information for the patches stored on the file in the following order:

```c
for (p = 1; p ≤ N_p; p++) {
    Offset (bytes) Name Type Bytes
    61 + 40(p - 1)  $h_p$ double 8
    69 + 40(p - 1)  $zmin_p$ double 8
    77 + 40(p - 1)  $ib_p$ int 4
    81 + 40(p - 1)  $ni_p$ int 4
    85 + 40(p - 1)  $jb_p$ int 4
    89 + 40(p - 1)  $nj_p$ int 4
    93 + 40(p - 1)  $kb_p$ int 4
    97 + 40(p - 1)  $nk_p$ int 4
}
```

Next, the data patches are stored as 4 or 8 byte floats, as indicated by the prec entry in the header. These can be read as a one-dimensional sequence of bytes in the order

```c
for (p = 1; p ≤ N_p; p++) {
    for (k = 1; k ≤ nk_p; k++) {
        for (j = 1; j ≤ nj_p; j++) {
            for (i = 1; i ≤ ni_p; i++) {
                read $u^{(p)}(i - 1 + ib_p, j - 1 + jb_p, k - 1 + kb_p)$
            }
        }
    }
}
```
If \( \text{grdinfo}=0 \), there is no additional information on the \( \text{volimage} \) file. If \( \text{grdinfo}=1 \), the \( z \)-coordinates of the curvilinear grid are also saved on the \( \text{volimage} \) file (as floats or doubles depending on \( \text{prec} \)). Note that the curvilinear grid always corresponds to the last patch, \( p = N_p \). The grid coordinate can be read in the same way as the data patches:

\[
p = N_p; \\
\text{for} \ (k = 1; \ k \leq nk_p; \ k++)\{ \\
\text{for} \ (j = 1; \ j \leq nj_p; \ j++)\{ \\
\text{for} \ (i = 1; \ i \leq ni_p; \ i++)\{ \\
\text{read} \ z^{(p)}(i - 1 + ib_p, j - 1 + jb_p, k - 1 + kb_p) \\
\}
\}
\}
\]

The \( x \)- and \( y \)- coordinates are uniformly distributed, also for the curvilinear grid, and can therefore always be reconstructed from the grid size \( h \), and the index bounds. The \( z \)-coordinate on other patches, when present, will always be uniform, and can be reconstructed from \( h \) and the \( z\text{min} \) variable.
Appendix A

Testing SW4

Once SW4 has been installed on your system, it is a good idea to verify that the code is working properly. For this purpose, the SW4 source code distribution includes a python(3) script for running several tests and checking the solutions against previously verified results. Here we describe the procedure when SW4 is built with make. The tests can also be performed when SW4 is built with CMake, see the SW4 installation guide [19] for details.

After SW4 has been built with make, go to the pytest directory and run test_sw4.py. If the sw4 executable resides in the optimize directory, you can run the basic tests by doing:

shell> cd pytest
shell> ./test_sw4.py

If all goes well, you should see the following output:

>shell test_sw4.py
Running all tests for level 0 ...
Starting test # 1 in directory: meshrefine with input file: refine-el-1.in
Test # 1 Input file: refine-el-1.in PASSED
Starting test # 2 in directory: meshrefine with input file: refine-att-1.in
Test # 2 Input file: refine-att-1.in PASSED
...
Starting test # 12 in directory: lamb with input file: lamb-1.in
Test # 12 Input file: lamb-1.in PASSED
Out of 12 tests, 12 passed and 0 failed.

Some aspects of the testing can be modified by providing command line arguments to test_sw4.py. For a complete list of options do test_sw4.py --help, which currently give the output:

shell> ./test_sw4.py --help
usage: test_sw4.py [-h] [-v] [-l {0,1,2}] [-m MPITASKS] [-d SW4_EXE_DIR]

optional arguments:
  -h, --help              show this help message and exit
  -v, --verbose           increase output verbosity
  -l {0,1,2}, --level {0,1,2}
                          testing level
Note that the directory name for the `sw4` executable should be given relative to the main `sw4` directory.

NOTE: On some systems, it is necessary to start an MPI daemon before any parallel programs can be executed. This is often done by issuing the command

```bash
mpd &
```

Ask your local system administrator if you have problems running `sw4` in parallel.

### A.1 Method of manufactured solutions

The method of manufactured solutions provides a general way of testing the accuracy of numerical solutions of partial differential equations, including effects of heterogeneous material properties and various boundary conditions on complex geometries. The test scripts can be found in the directory

`.../sw4/examples/twilight`

In these tests, we take the material properties to be

\[
\rho(x, y, z) = A_ρ (2 + \sin(\omega_m x + \theta_m) \cos(\omega_m y + \theta_m) \sin(\omega_m z + \theta_m)),
\]
\[
\mu(x, y, z) = A_μ (3 + \cos(\omega_m x + \theta_m) \sin(\omega_m y + \theta_m) \sin(\omega_m z + \theta_m)),
\]
\[
\lambda(x, y, z) = A_λ (2 + \sin(\omega_m x + \theta_m) \sin(\omega_m y + \theta_m) \cos(\omega_m z + \theta_m)).
\]

The internal forcing, boundary forcing and initial conditions are chosen such that the exact (manufactured) solution becomes\[1\]

\[
\begin{align*}
    u_e(x, y, z, t) &= \sin(\omega(x - c_e t)) \sin(\omega y + \theta) \sin(\omega z + \theta), \\
    v_e(x, y, z, t) &= \sin(\omega x + \theta) \sin(\omega(y - c_e t)) \sin(\omega z + \theta), \\
    w_e(x, y, z, t) &= \sin(\omega x + \theta) \sin(\omega y + \theta) \sin(\omega(z - c_e t)).
\end{align*}
\]

The values of the material parameters ($\omega_m$, $\theta_m$, $A_ρ$, $A_λ$, $A_μ$) and the solution parameters ($\omega$, $\theta$, $c_e$), can be modified in the input script. Since the exact solution is known, it is possible to evaluate the error in the numerical solution. By repeating the same test on several grid sizes, it is possible to establish the convergence rate of the numerical method.

The basic twilight tests use a single grid, a flat topography, and use the unit cube $(x, y, z) \in [0, 1]^3$ as the computational domain. The numerical solution is simulated up to time $t = 0.8$ on a grid with $31^3$, $61^3$, $121^3$, and $241^3$ grid points, respectively. These cases are provided in the four input scripts:

`flat-twi-1.in flat-twi-2.in flat-twi-3.in flat-twi-4.in`

\[1\] A contour plot of these functions could help explain why we call it twilight zone testing.
The first three cases are small enough to be run on a single or dual core laptop computer. The fourth case uses about 14 million grid points and needs to be executed on a slightly larger machine. Assuming openmpirun is used to execute parallel programs, you run the first of these cases on 2 processes with the command

```
  cd examples/twilight
  openmpirun -np 2 ../.../optimize/sw4 flat-twi-1.in
```

The results for the four cases are given in the output files

```
  flat-twi-1.out  flat-twi-2.out  flat-twi-3.out  flat-twi-4.out
```

The errors in max and $L_2$ norm in the numerical solution are reported near the bottom of the output files. Some of these numbers are summarized in Table A.1.

<table>
<thead>
<tr>
<th>input file</th>
<th>$N_x$</th>
<th>$h$</th>
<th>$e_h = |u - u_{ex}|_{\infty}$</th>
<th>ratio$_\infty$</th>
<th>rate$_\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>flat-twi-1.in</td>
<td>31</td>
<td>$3.333 \cdot 10^{-2}$</td>
<td>$6.85 \cdot 10^{-4}$</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>flat-twi-2.in</td>
<td>61</td>
<td>$1.667 \cdot 10^{-2}$</td>
<td>$3.99 \cdot 10^{-5}$</td>
<td>17.16</td>
<td>4.10</td>
</tr>
<tr>
<td>flat-twi-3.in</td>
<td>121</td>
<td>$8.333 \cdot 10^{-3}$</td>
<td>$2.01 \cdot 10^{-6}$</td>
<td>19.85</td>
<td>4.31</td>
</tr>
<tr>
<td>flat-twi-4.in</td>
<td>241</td>
<td>$4.167 \cdot 10^{-3}$</td>
<td>$1.40 \cdot 10^{-7}$</td>
<td>14.36</td>
<td>3.84</td>
</tr>
</tbody>
</table>

Table A.1: Max norm errors in the displacement at time $t = 0.8$, when the free surface is flat. The convergence rate is calculated as $\log_2(e_{2h}/e_h)$.

The case of a non-planar free surface is tested by the scripts

```
  gauss-twi-1.in  gauss-twi-2.in  gauss-twi-3.in  gauss-twi-4.in  gauss-twi-5.in
```

Here, the computational domain is basically a cube with side 1, where the top surface is shaped as a Gaussian hill. Let the shape of the top surface be described by $z = \tau(x,y)$. Because the $z$-axis is directed downwards, and the Gaussian hill has amplitude 0.05, the function $\tau$ satisfies $-0.05 \leq \tau \leq 0$. The keyword $\text{zmax}=0.25$ in the topography command tells SW4 to build the curvilinear grid between $z = \tau(x,y)$ and $z = 0.25$. A Cartesian grid is used to cover the rest of the computational domain, i.e., for $0.25 \leq z \leq 1$. The numerical solution is calculated up to time $t = 0.8$ on grids with grid size $h = 1/30$, $1/60$, $1/120$, and $1/240$, respectively. The three first cases are relatively small and can be run on a single or dual core laptop computer. The number of grid point is increased by a factor of eight between each grid refinement, and the number of time steps is doubled. The results of the simulations are given in the four output files

```
  gauss-twi-1.out  gauss-twi-2.out  gauss-twi-3.out  gauss-twi-4.out
```

The errors in max norm are summarized in Table A.2.

Note that some image files are generated by these scripts. They are placed in the sub-directories `gauss_31`, `gauss_61`, etc. We encourage the reader to look at these image files, for example by reading them into matlab/octave using the script `tools/readimage.m`. An example is given in Figure A.1.
\[ e_h = \| u - u_{ex} \|_\infty \]

<table>
<thead>
<tr>
<th>input file</th>
<th>( N_x )</th>
<th>( h )</th>
<th>( e_h )</th>
<th>ratio(\infty)</th>
<th>rate(\infty)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gauss-twi-1.in</td>
<td>31</td>
<td>3.333 (\cdot) 10^{-2}</td>
<td>1.74 (\cdot) 10^{-3}</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>gauss-twi-2.in</td>
<td>61</td>
<td>1.667 (\cdot) 10^{-2}</td>
<td>1.93 (\cdot) 10^{-4}</td>
<td>9.01</td>
<td>3.17</td>
</tr>
<tr>
<td>gauss-twi-3.in</td>
<td>121</td>
<td>8.333 (\cdot) 10^{-3}</td>
<td>9.11 (\cdot) 10^{-6}</td>
<td>21.1</td>
<td>4.40</td>
</tr>
<tr>
<td>gauss-twi-4.in</td>
<td>241</td>
<td>4.167 (\cdot) 10^{-3}</td>
<td>5.83 (\cdot) 10^{-7}</td>
<td>15.6</td>
<td>3.96</td>
</tr>
</tbody>
</table>

Table A.2: Max norm errors in the displacement at time \( t = 0.8 \), with a Gaussian hill topography. The number of grid points and the grid size refer to the Cartesian grid. The convergence rate is calculated as \( \log_2(e_{2h}/e_h) \).

Figure A.1: Contour plot of the \( u(x) \) component of the solution along the plane \( x = 0.5 \). The image file was written with the script gauss-twi-3.in. The solid black line shows the shape of the top surface. The dashed black line indicates the interface between the curvilinear and Cartesian grids.
A.2 Lamb’s problem

In this section we use the testlamb command to evaluate the accuracy of the numerical solution of Lamb’s problem. An introduction to Lamb’s problem is given in Section 4.5.1.

We consider an elastic half-space with shear speed \( C_s = 1 \) m/s, compressional speed \( C_p = \sqrt{3} \) m/s and density \( \rho = 1 \) kg/m\(^3\). The setup can be found in examples/lamb/lamb-1.in,

```plaintext
fileio path=lamb-h0p04
grid x=12 y=12 z=6 h=0.04
time t=15.0
supergrid gp=50
testlamb rho=1 cp=1.732050807568877
source x=6 y=6 z=0 fx=0 fy=0 fz=1 t0=0 freq=1 type=C6SmoothBump
rec x=10.0 y=6.0 z=0 file=sg1 usgsformat=1 sacformat=0
rec x=10.0 y=10.0 z=0 file=sg2 usgsformat=1 sacformat=0
```

The super-grid far-field layer is 50 grid points wide, corresponding to the width \( \ell = 2 \). The error is evaluated along the free surface \( z = 0 \), inside of the supergrid layers, i.e., \( 2 \leq (x,y) \leq 10 \), and is saved on the file lamb-h0p04/LambErr.txt. Each line of this file has four columns corresponding to the time, max error, \( L_2 \) error, max norm of solution. Since the surface waves propagate with speed \( C_r \approx 0.92 \) and the source time function is zero after \( t = 1 \), the exact solution becomes identically zero along the surface after time \( t \approx 1+4\sqrt{2}/0.92 \approx 7.15 \). After this time, the error in the numerical solution is due to artificial reflections from the super-grid layers. In Figure A.2 we plot the \( L_2 \) norm of the error as function of time for grid sizes \( h = 0.04 \) and \( h = 0.02 \). Here, the case with grid size \( h = 0.02 \) can be found in examples/lamb/lamb-2.in. Note that the error converges to zero as \( O(h^4) \) for \( t > 1 \), i.e., after the singular point force has stopped acting. Another way of assessing the quality of the super-grid layers is by inspecting the solution in the vicinity of the super-grid layers. The two rec commands save the solution at \((x,y,z) = (6,10,0)\) and \((x,y,z) = (10,10,0)\). We can use the plotusgs matlab/octave script to inspect the numerical solution, see Figure A.3.

We conclude that there are no visible artifacts due to the truncation of the computational domain.

A.3 Point source test

There is an analytical solution of the elastic wave equation when the forcing is a point source term, and the domain is unbounded. The forcing can either be a point force or a point moment tensor source. In both these cases, the analytical solution is singular (i.e. infinite) at the point source, while the point source is acting. For time functions that are identically zero for \( t \geq t_1 \), the analytical solution becomes bounded for \( t > t_1 \). The smoothness of the solution is determined by the smoothness of the source time function.

The accuracy of the numerical solution can be evaluated by comparing it to the analytical solution using the testpointsource command. As before we refine the grid by halving the grid size to evaluate the convergence rate of the numerical solution. Since we previously tested the point force discretization by solving Lamb’s problem, we will here focus on the point moment tensor source discretization. Example input files are given in the files ps-dir1.in, ps-dir2.in, and ps-dir3.in in the tools/pointsource directory. Each input file has a corresponding .out file with the results we obtained while running these cases.
The source properties are specified by exactly one regular source command. In this case the time function is a \texttt{C6SmoothBump}. The coarsest grid is setup in the file \texttt{ps-dir1.in},

\begin{verbatim}
fileio path=dir-h0p04
grid x=8 y=8 z=8 h=0.04
time t=2.5

# dirichlet conditions on all sides
boundary_conditions lx=1 hx=1 ly=1 hy=1 lz=1 hz=1

# testing point sources
testpointsource rho=1 cp=1.6 cs=0.8

# source
source x=4.0 y=4.0 z=4.0 Mxx=1 Myy=1 Mzz=1 Mxy=0 Mxz=0 Myz=0 t0=0 freq=1 type=C6SmoothBump
\end{verbatim}

The source is located in the center of a cube with side 8, so the shortest distance from the source to a boundary is 4. The solution satisfies homogeneous Dirichlet boundary conditions on all boundaries. The fastest wave speed is 1.6, so the solution on the boundary is identically zero until time $t = 4/1.6 = 2.5$. There can therefore not be any reflections from the boundaries for $t \leq 2.5$, and the analytical solution is reliable up to this point in time. The error in the numerical solution at $t = 2.5$ (the end of the simulation) is reported at the end of each run. These numbers are summarized in Table A.3. Since the convergence rate tends to 4 as the grid is refined, we conclude that the numerical solution is a fourth order accurate approximation of the analytical solution.
Figure A.3: The $x$, $y$, and $z$-components of the solution of Lamb’s problem at $(x, y, z) = (10, 10, 0)$.

<table>
<thead>
<tr>
<th>input file</th>
<th>$h$</th>
<th>$e_h = |u - u_{ex}|_\infty$</th>
<th>ratio$_\infty$</th>
<th>rate$_\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ps-dir1.in</td>
<td>0.04</td>
<td>$2.09 \cdot 10^{-3}$</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>ps-dir2.in</td>
<td>0.02</td>
<td>$1.41 \cdot 10^{-4}$</td>
<td>14.82</td>
<td>3.88</td>
</tr>
<tr>
<td>ps-dir3.in</td>
<td>0.01</td>
<td>$8.94 \cdot 10^{-6}$</td>
<td>15.77</td>
<td>3.97</td>
</tr>
</tbody>
</table>

Table A.3: Max norm errors in the displacement at time $t = 2.5$. The convergence rate is calculated as $\log_2(e_{2h}/e_h)$.

More detailed information of the error in the numerical solution is saved in the file `PointSourceErr.txt`, which is written in the output directory that is specified by the `fileio` command. Each line in this file has 4 entries: time, max-error, $L_2$-error, and max-norm of the solution. In Figure A.4, we plot the max-error as function of time for the three grid sizes. The source time function is identically zero for $t \geq 1$. Before $t = 1$, the analytical solution is unbounded at the point source. For this reason, a few points around the point source are excluded from the calculation of the error in the numerical solution. For $t > 1$, we see that the error in the numerical solution is reduced by a factor $\approx 16$ each time the grid size is halved.

We remark that the `testpointssource` command can also be used to evaluate the accuracy of the supergrid far field layers. In this case, the Dirichlet boundary conditions should be replaced by supergrid conditions on all six boundaries of the computational domain. Such an experiment is set up in the input files `pointsource-sg1.in`, `pointsource-sg2.in`, and `pointsource-sg3.in` in the `tools/pointsource` directory. Each input file has a corresponding `.out` file that holds the output from these cases. As before, the errors for each case are reported in the file `PointSourceErr.txt`.
Figure A.4: The max norm of the error in the point source test. The blue, green, and red lines correspond to $h = 0.04$, $h = 0.02$, and $h = 0.01$, respectively.

in the sub-directories `pointsource-h0p04+`, `pointsource-h0p02+`, and `pointsource-h0p01+`. The $L_2$ norm of the error is shown in Figure A.5. The reader is encouraged to study these results, experiment with the duration of the simulation, and the width of the super grid layers.
Figure A.5: The $L_2$-norm of the error in the point source test. This simulation was run to time $t = 5$ and the errors after time $t \approx 3$ are due to artificial reflections from the super-grid layers. The blue, green, and red lines correspond to $h = 0.04$, $h = 0.02$, and $h = 0.01$, respectively.
Appendix B

Run time and memory requirements

B.1 Run time

The execution times shown in Table B.1 were obtained by running SW4 on problems with approximately $2.8 \times 10^5$ to $3.3 \times 10^5$ grid points per processor. Timings were measured on 16 processors of Cab, a parallel supercomputer at LLNL, in August of 2013. The SW4 executable was compiled with Intel compilers at optimization level -O. The execution times are given in seconds per grid point and time step. While the absolute numbers in Table B.1 are machine dependent and destined to change in the future, the relative cost of various solver configurations is likely to stay more or less constant. Note that the visco-elastic cases used 3 mechanisms. For the case with topography, about 31% of the grid points were in the curvilinear grid.

We note that solving the visco-elastic wave equations with $m = 3$, requires around 3 times more CPU time than the purely elastic case. Using topography requires about 1.5 times more CPU time when 31% of the domain is discretized on a curvilinear grid. By extrapolation, if all grid points were in the curvilinear grid, the increase in CPU time would have been about a factor 2.7. The strong scaling properties are not perfect. Hence, the performance might be degraded when the number of processors is increased while keeping the number of grid points fixed. However, the total computational time always decreases with increasing number of processors. More experimentation is needed to better evaluate the strong scaling properties of SW4.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Solver</th>
<th>Execution time [s]</th>
<th>Relative factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cartesian</td>
<td>elastic</td>
<td>$2.62 \cdot 10^{-8}$</td>
<td>1</td>
</tr>
<tr>
<td>Topography</td>
<td>elastic</td>
<td>$4.05 \cdot 10^{-8}$</td>
<td>1.5</td>
</tr>
<tr>
<td>Cartesian</td>
<td>visco-elastic</td>
<td>$7.96 \cdot 10^{-8}$</td>
<td>3.0</td>
</tr>
<tr>
<td>Topography</td>
<td>visco-elastic</td>
<td>$1.37 \cdot 10^{-7}$</td>
<td>5.2</td>
</tr>
</tbody>
</table>

Table B.1: Execution time per time step and grid point for different configurations, both for solving the elastic and the visco-elastic wave equations. The relative factor is the execution time relative to the Cartesian case for the elastic wave equation.
<table>
<thead>
<tr>
<th>Configuration</th>
<th>solver</th>
<th>Memory [bytes/grid point]</th>
<th>Asymp. [bytes/grid point]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cartesian grid</td>
<td>elastic</td>
<td>188</td>
<td>168</td>
</tr>
<tr>
<td>Topography</td>
<td>elastic</td>
<td>208</td>
<td>200</td>
</tr>
<tr>
<td>Cartesian grid</td>
<td>visco-elastic ($m = 3$)</td>
<td>475</td>
<td>448</td>
</tr>
<tr>
<td>Topography</td>
<td>visco-elastic ($m = 3$)</td>
<td>500</td>
<td>468</td>
</tr>
</tbody>
</table>

Table B.2: Observed memory usage (third column) and theoretical asymptotic limit (fourth column) for different grid and solver configurations. The visco-elastic case has 3 mechanisms and the case with topography has 31% of the grid point in the curvilinear grid.

### B.2 Memory usage

The memory usage in Table B.2 was obtained by running SW4 on a single processor using 4 million grid points. By inspection of the source code, we found the following number of 3-D arrays holding double precision (8 bytes) floating point variables: 21 for Cartesian grids, another 8 for topography (coordinates and metric coefficients in the curvilinear grid), and an additional $2 + 11 \times m$ for a $m$-mechanism visco-elastic material. These numbers correspond to the asymptotic memory requirements presented in the right column of Table B.2 which agree reasonably well with the observed memory usage, shown in column three. The observed numbers tend to the theoretical asymptotic numbers as the problem size increases. For the computations with topography, the curvilinear grid holds about 31% of the grid points. This fraction is taken into account in the asymptotic estimate.

For parallel runs, the memory per grid point will be somewhat higher, because of duplicated ghost points at the processor boundaries. Here we only report the memory usage for a single processor run. Note that the volimage command saves a copy of the image variable over the full 3D volume. This adds (at single precision) 4 bytes per grid point and per volume image command.
Bibliography


