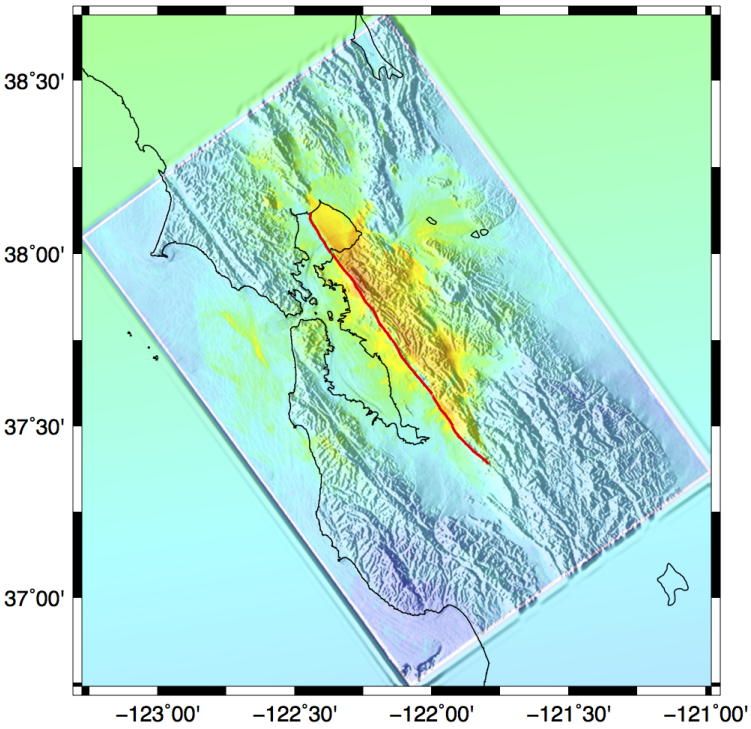
COMPUTATIONAL INFRASTRUCTURE FOR GEODYNAMICS (CIG)

Lawrence Livermore National Laboratory

SW4



User's Guide

Version 3.0

N. Anders Petersson
Bjorn Sjogreen
Houjun Tang

www.geodynamics.org

Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, 7000 East Ave., Livermore, CA 94550. This is contribution LLNL-SM-741439.

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.

Contents

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 lenghts 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 materials, 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 implemented in Fortan-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 $\mathcal{O}(100,000)$ 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 functionality 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 validation 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 acknowledgments in talks and publications. The following peer-reviewed papers discuss the numerical methods that are implemented in *SW4*:

- Petersson, N.A. and B. Sjögreen (2015). Wave propagation in anisotropic elastic materials and curvilinear coordinates using a summation-by-parts finite difference method, Journal of Computational Physics, 299, 820-841. DOI: 10.1016/j.jcp.2015.07.023, URL: http://linkinghub.elsevier.com/retrieve/pii/S0021999115004684.

- Petersson, N.A. and B. Sjögreen (2012). Stable and efficient modeling of anelastic attenuation in seismic wave propagation, Communications in Computational Physics, 12 (01), 193-225.

- Sjögreen, B. and N.A. Petersson (2012). A Fourth Order Accurate Finite Difference Scheme for the Elastic Wave Equation in Second Order Formulation, Journal of Scientific Computing, 52 (1) , 17-48, doi: 10.1007/s10915-011-9531-1, url: http://link.springer.com/10.1007/s10915-011-9531-1

To cite the *SW4* software and manual, use:

- Petersson, N.A. and B. Sjögreen (2017). SW4 v2.0. Computational Infrastructure of Geodynamics, Davis, CA. DOI: 10.5281/zenodo.1045297.

- Petersson, N.A. and B. Sjögreen (2017). User's guide to SW4, version 2.0. Technical report LLNL-SM-741439, Lawrence Livermore National Laboratory, Livermore, CA.

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 *SW4*

This section assumes that *SW4* has already been installed on your computer system. We refer to the report by Petersson and Sjogreen [19] for instructions on how to install *SW4*.

SW4 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 *SW4*, so each command must be given on one (sometimes long) line in the input file.

Since *SW4* is a parallel code, it is required to be run under a parallel execution environment such as mpiexec, mpirun, openmpirun, or srun. The `srun` command is currently always used on the parallel machines at Livermore Computing. It is important to start *SW4* with the correct parallel execution tool. For example, if you build *SW4* with the `openmpi` compilers, you should use the `openmpirun` environment. Also note that some systems require you to start an `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 *SW4*, 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 `.in`. However, *SW4* will attempt to read any input file, regardless of its extension.

If your system is setup for using `mpiexec`, the command

```
shell> mpiexec -np 2 sw4 test.in
```

runs *SW4* on 2 processes, and tells it to read the input from a file named `test.in`. If you are using `mpirun`, you would instead use the command

```
shell> mpirun -np 2 sw4 test.in
```

Remark: If *SW4* produces strange looking outputs, for example where the same text is repeated several times (e.g. once per processor), you are probably running *SW4* under the wrong parallel execution environment. Make sure you are running *SW4* under an environment that is compatible with the compiler that was used to build *SW4*.

2.1.1 Version information (-v)

Version information for the *SW4* executable can be obtained through the `-v` flag:

```
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

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

runs *SW4* on 32 processors on the debug partion 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

SW4 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 $\mathbf{x} \in \Omega$ during the time interval $0 \leq t \leq T$. By default, the motion starts from rest and is driven by a forcing function $\mathbf{F}(\mathbf{x}, t)$. In the elastic case, the motion is governed by

$$\rho \mathbf{u}_{tt} = \nabla \cdot \mathcal{T} + \mathbf{F}(\mathbf{x}, t), \qquad \mathbf{x} \text{ in } \Omega,\ 0 \leq t \leq T, \tag{3.1}$$

$$\mathbf{u}(\mathbf{x}, 0) = 0, \quad \mathbf{u}_t(\mathbf{x}, 0) = 0, \qquad \mathbf{x} \text{ in } \Omega. \tag{3.2}$$

Here, ρ is the density, $\mathbf{u}(\mathbf{x}, t)$ is the displacement vector, and $\mathcal{T} = \mathcal{T}(\mathbf{u})$ is the stress tensor. The computational domain Ω 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,

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

Here $\mathbf{n}$ 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.

SW4 uses a right-handed Cartesian coordinate system with the z-direction pointing downwards into the medium, see figure 3.1. *SW4* 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 *SW4* 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. *SW4* 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,

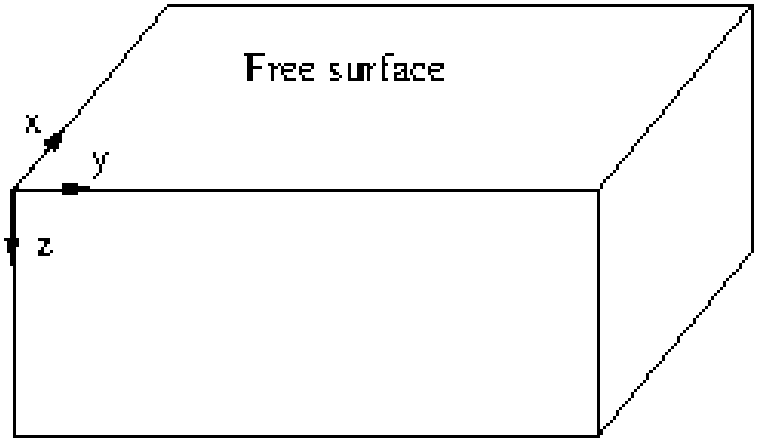


Figure 3.1: *SW4* uses a right handed coordinate system with the z-axis pointing downwards.

the computational domain is given by

$$0 \le x \le x_{max}, \quad 0 \le y \le y_{max}, \quad \tau(x,y) \le z \le z_{max}. \tag{3.3}$$

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

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

The extent in the x-direction is thereafter adjusted to

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

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

$$h = x/(nx - 1) = 303.03.$$

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 `time` command. For example,

```
time t=1.6
```

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

```
time steps=1200
```

The end time will in this case be $T = 1200\Delta t$, where the time step Δt is determined automatically by *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 `utcstart` sets the UTC that corresponds to simulation time $t = 0$, for example,

```
time t=1.6 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 `rec` command). Note that the UTC can be very useful for aligning simulated and observed time series.

3.1 Geographical coordinates and projections

SW4 supports geographical coordinates as an alternative way of specifying spatial locations. The geographic location of the origin of the Cartesian coordinte system (lat_0, lon_0), see Figure 3.2, is specified in the grid command. If no location is given, the default location is $\text{lat}_0 = 37$ degrees, $\text{lon}_0 = -118$ 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 `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,

$$\text{lat} = \text{lat}_0 + \frac{x\cos(\alpha) - y\sin(\alpha)}{M}, \quad \alpha = \text{az}\frac{\pi}{180}, \tag{3.6}$$

$$\text{lon} = \text{lon}_0 + \frac{x\sin(\alpha) + y\cos(\alpha)}{M\cos(\phi\pi/180)}. \tag{3.7}$$

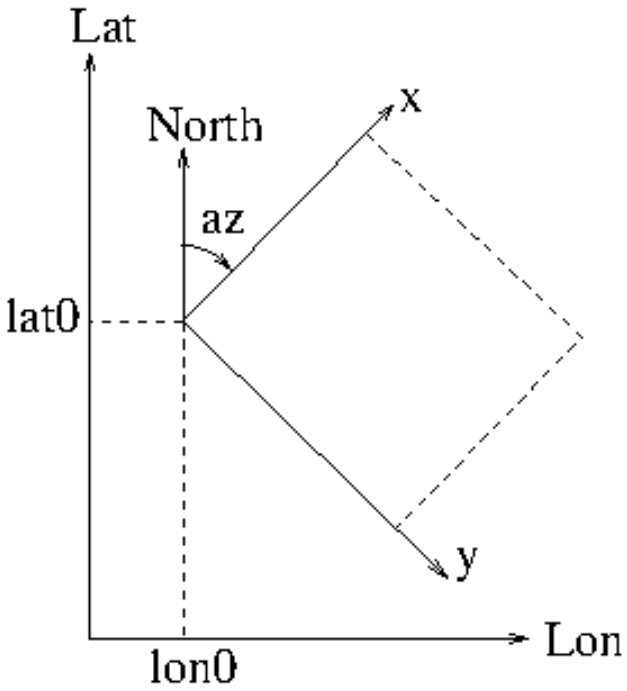


Figure 3.2: Geographical coordinates in *SW4*.

In this formula, lat, lon, az, lat_0, and 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:

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

3.1.2 The Proj.4 library

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

```
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 streching 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 *SW4*, 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 accomodate 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 $\mathbf{F}$ in equation (3.1) consists of a sum of point forces and point moment tensor source terms. For a point forcing we have

$$\mathbf{F}(\mathbf{x},t) = g(t,t_0,\omega)F_0 \begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix} \delta(\mathbf{x}-\mathbf{x_0}),$$

where $\mathbf{x}_0 = (x_0, y_0, z_0)$ 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 ω. 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

$$\mathbf{F}(\mathbf{x},t) = g(t,t_0,\omega)\,\mathcal{M}\cdot\nabla\delta(\mathbf{x}-\mathbf{x_0}), \quad \mathcal{M} = \begin{pmatrix} M_{xx} & M_{xy} & M_{xz} \\ M_{xy} & M_{yy} & M_{yz} \\ M_{xz} & M_{yz} & M_{zz} \end{pmatrix}.$$

The seismic moment of a moment tensor source is defined by

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

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 (M0): 1.7162e+17 Nm
  Moment magnitude     (Mw): 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 *S*W4 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 $\mathbf{x}_0 = (5000, 4000, 600)$ with amplitude 10^{15} Nm, using the `RickerInt` time function with offset time $t_0 = 1$ s and frequency parameter $\omega = 5$ Hz. The off-diagonal moment tensor elements (M_{xy}, M_{xz} 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 *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, incompatibilty problems can only be avoided if t_0 is positive and of the order $\mathcal{O}(1/\omega)$, where ω equals the `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 σ) is related to ω 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

$$g(t, t_0, \omega) = \left(2\pi^2\omega^2 (t-t_0)^2 - 1\right) e^{-\pi^2\omega^2 (t-t_0)^2}.$$

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

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

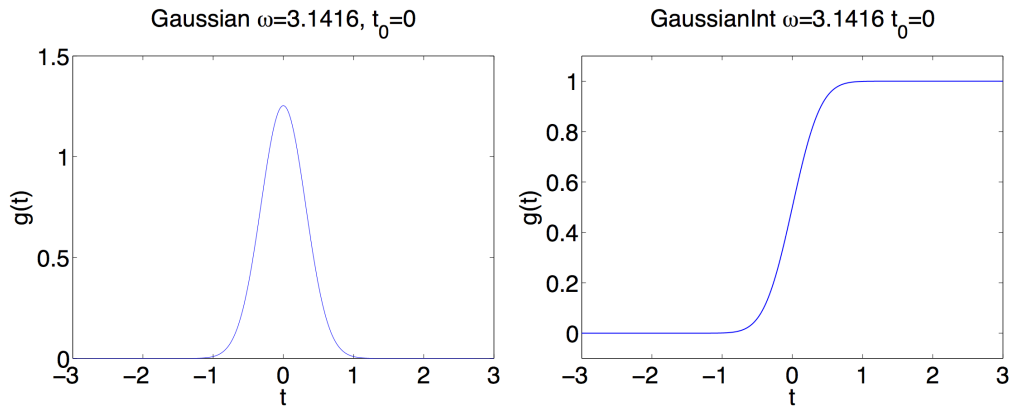


Figure 4.1: Gaussian (left) and GaussianInt (right) with $\omega = \pi$ and $t_0 = 0$.

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,

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

The parameter τ_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

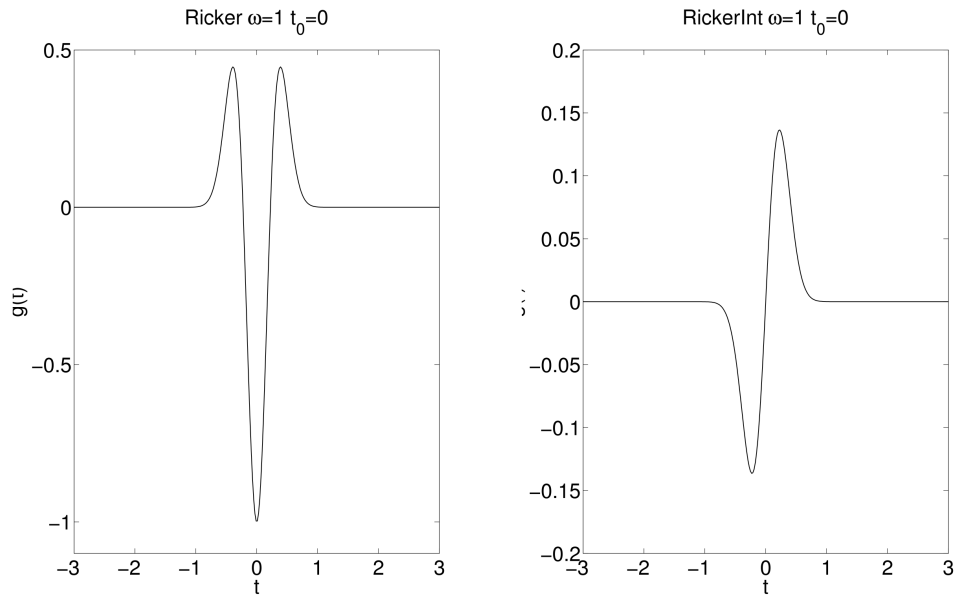


Figure 4.2: Ricker (left) and RickerInt (right) with $\omega = 1$ and $t_0 = 0$.

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

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

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.

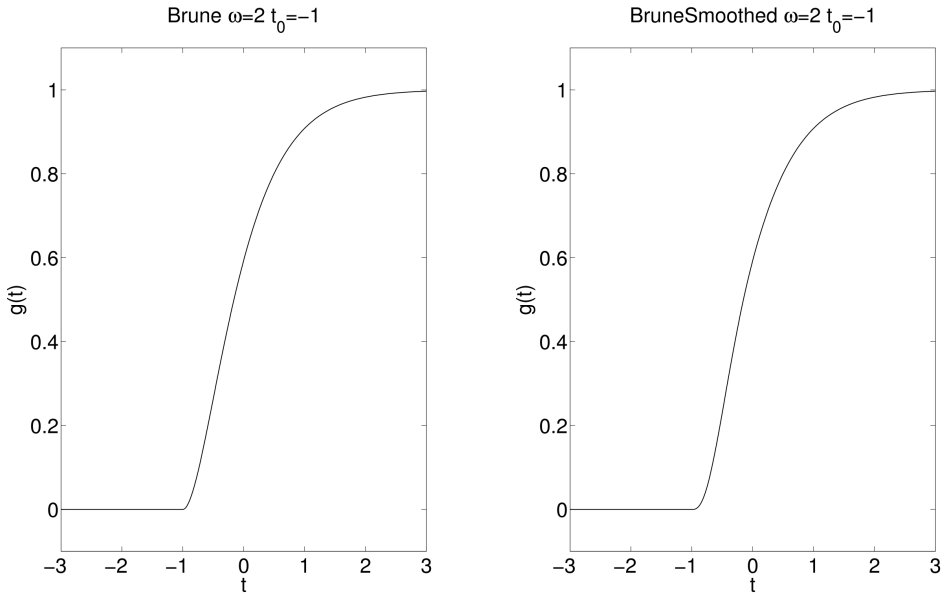


Figure 4.3: Brune (left) and BruneSmoothed (right) with $\omega = 2$ and $t_0 = -1$.

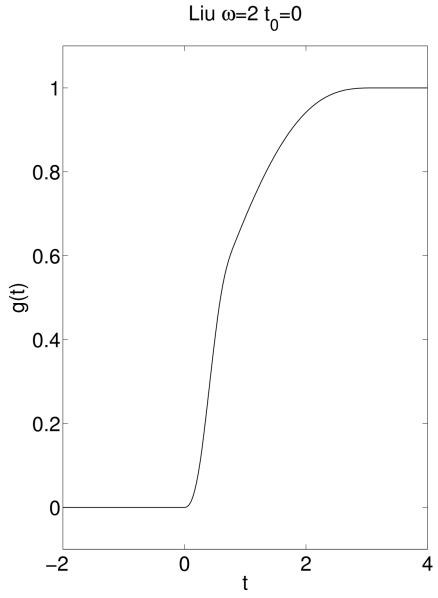


Figure 4.4: Liu time function with $\omega = 2$ and $t_0 = 0$.

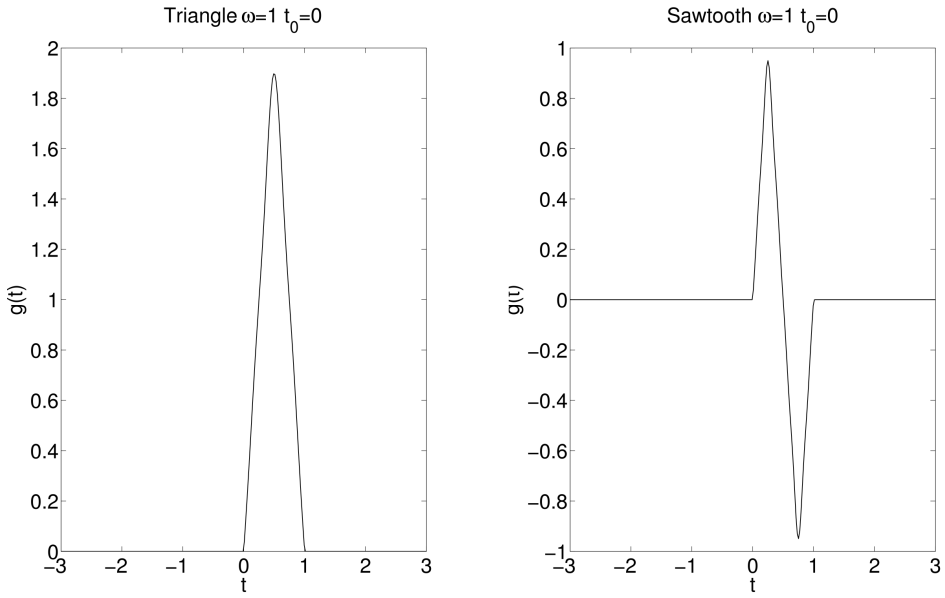


Figure 4.5: Triangle (left) and Sawtooth (right) with $\omega = 1$ and $t_0 = 0$.

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

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

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

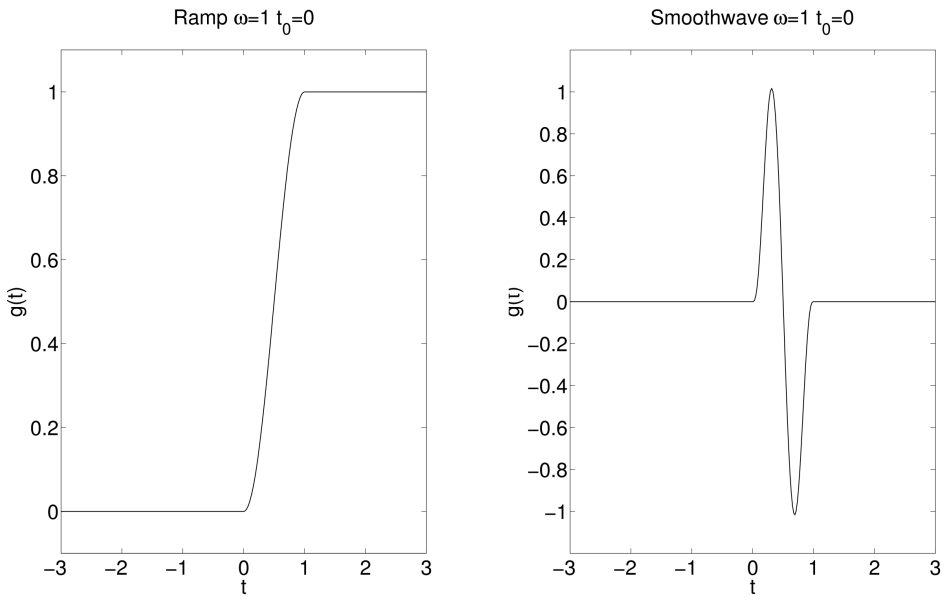


Figure 4.6: Ramp (left) and Smoothwave (right) with $\omega = 1$ and $t_0 = 0$.

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

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

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

4.2.13 C6SmoothBump

$$g(t, t_0, \omega) = \begin{cases} 0, & t < t_0, \\ 51480\,\omega^7(t - t_0)^7(1 - \omega(t - t_0))^7, & t_0 \le t \le t_0 + 1/\omega, \\ 0, & t > t_0 + 1/\omega. \end{cases}$$

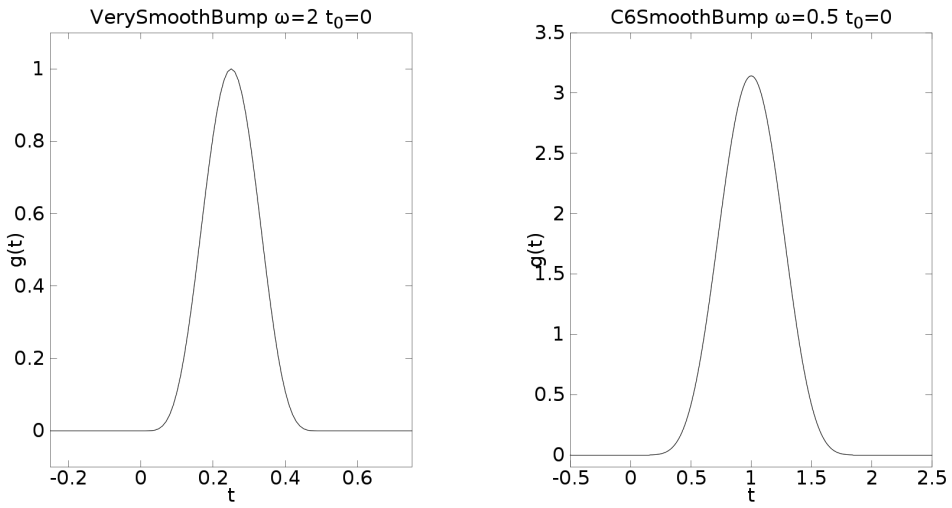


Figure 4.7: VerySmoothBump (left) with $\omega = 0.5$ and $t_0 = 0$. C6SmoothBump (right) with $\omega = 2$ and $t_0 = 0$.

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

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

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,

$$\int f(t)\delta(t - \tau_0)\, dt = f(\tau_0). \tag{4.1}$$

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 τ_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.

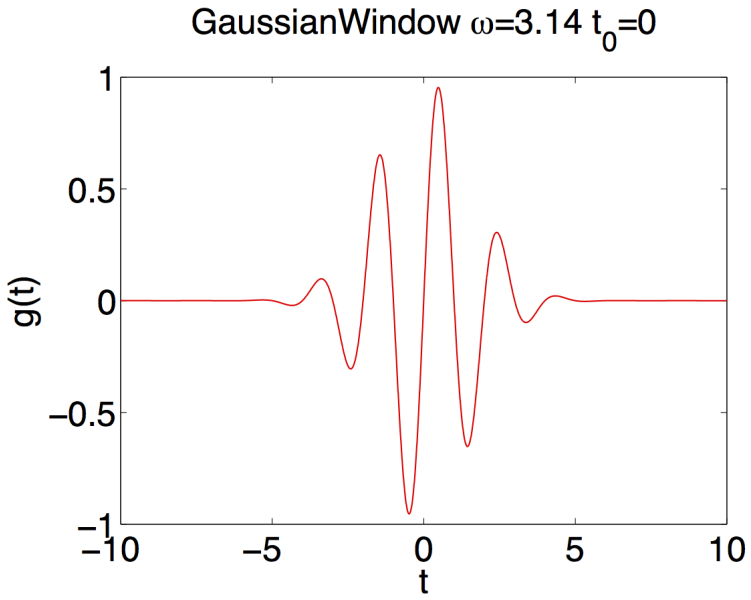


Figure 4.8: GaussianWindow with $\omega = 3.14$, $t_0 = 0$, and $N_c = 5$.

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 $0 \leq t < \tau_1$. Similarily, 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 decribed in Section 12.1.

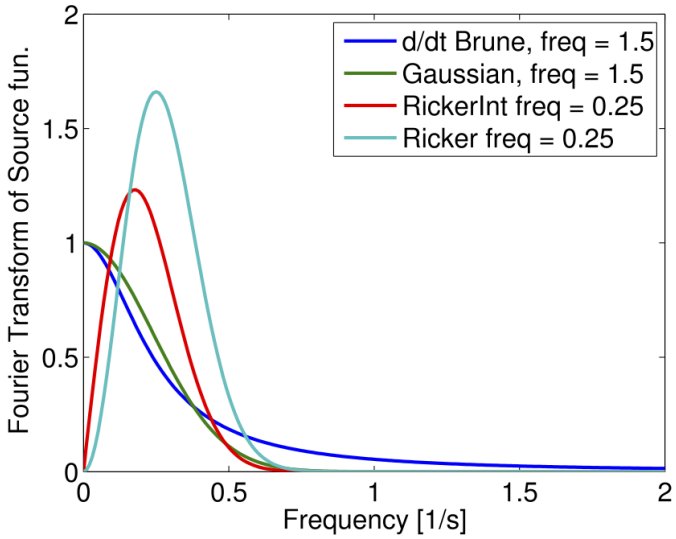


Figure 4.9: Magnitude of the Fourier transform of d/dt(Brune) (dark blue), the Gaussian (green), the RickerInt (red), and the Ricker (light blue) time-functions. Here `freq`=1.5 for the Gaussian and d/dt(Brune), and `freq`=0.25 for Ricker and RickerInt.

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 `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 `freq` parameter is given by

$$f_0 = \begin{cases} \texttt{freq}, & \text{for Ricker, RickerInt, VerySmoothBump, C6SmoothBump,} \\ \dfrac{\texttt{freq}}{2\pi}, & \text{for Liu, Brune, BruneSmoothed, Gaussian, GaussianInt, and GaussianWindow.} \end{cases} \tag{4.2}$$

The plots in Figure 4.9 were made with frequency parameter `freq`=0.25 for the Ricker and RickerInt functions and frequency parameter `freq`=1.5 for the Gaussian and d/dt(Brune) functions. Hence, $f_0 \approx 0.25$ for all functions in Figure 4.9. Note that the Fourier transform of the d/dt(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_{max}, that generates the shortest waves and therefore determines how fine the computational grid must be. For practical purposes f_{max} can be defined as the frequency where the amplitude of the Fourier transform falls below 5 % of its max value. We have

$$f_{max} \approx \begin{cases} 2.5 f_0, & \text{Ricker, RickerInt, Gaussian time functions,} \\ 3.0 f_0, & \text{C6SmoothBump time function,} \\ 4.0 f_0, & d/dt\text{(Brune) time function.} \end{cases} \tag{4.3}$$

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_{max} in (4.3) is very important. For time functions that not are listed in that formula, it is possible to estimate f_{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

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

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 wave length and propagate at approximately the same speed, we can estimate the shortest wave length by

$$L_{min} = \frac{\min C_s}{f_{max}}.$$

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

$$P = \frac{L_{min}}{h} = \frac{\min C_s}{h\, f_{max}}. \tag{4.4}$$

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_{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 `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" (see Figure 4.7) 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 `examples/lamb/seismic1.in`

```
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 `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 `examples/lamb/seismic2.in`. In this case, we evaluate the solution of Lamb's problem using the Fortran program `src/lamb_one_point.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

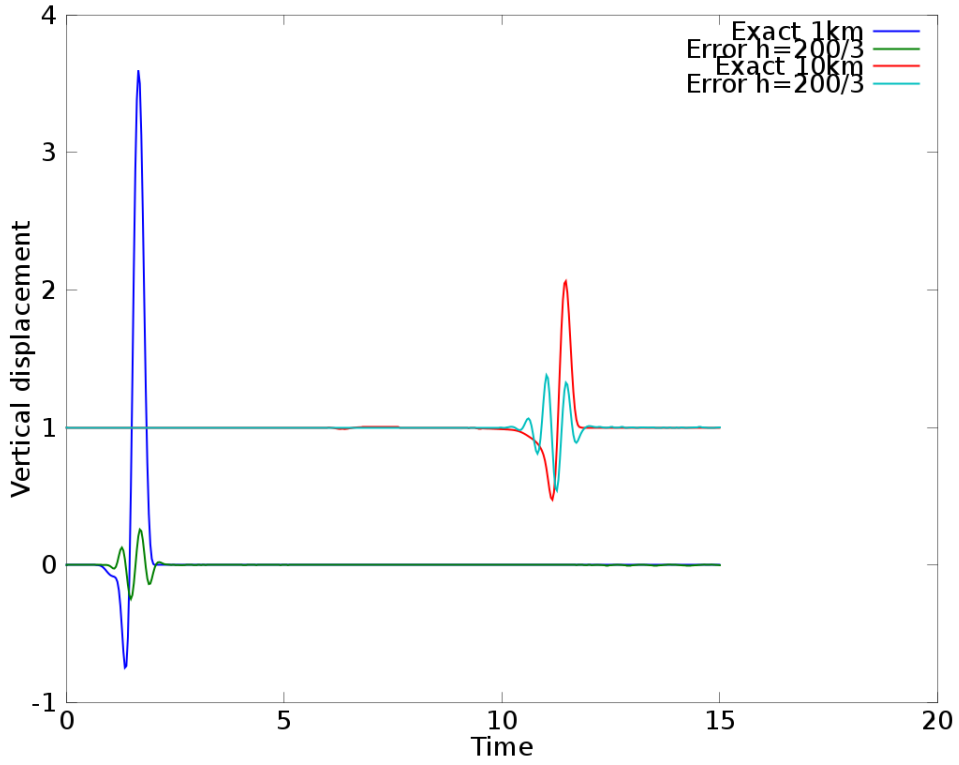


Figure 4.10: Lamb's problem: Vertical displacement at 1,000 (blue) and 10,000 (red) meters from the source. The green and light blue lines show the corresponding errors in the numerical solution with $h = 200/3$, corresponding to $P = 5$ grid points per shortest wave length. The displacement is offset 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 $f_{max} = 3.0$ 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_{GP}) increases when the grid is refined.

The errors are also plotted in Figure 4.11. The error decreases at a rate approaching $\mathcal{O}(h^4)$ 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 *SW4* depends on the distance between the source and the receiver, which can be normalized by the dominant wave length in the solution.

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

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.

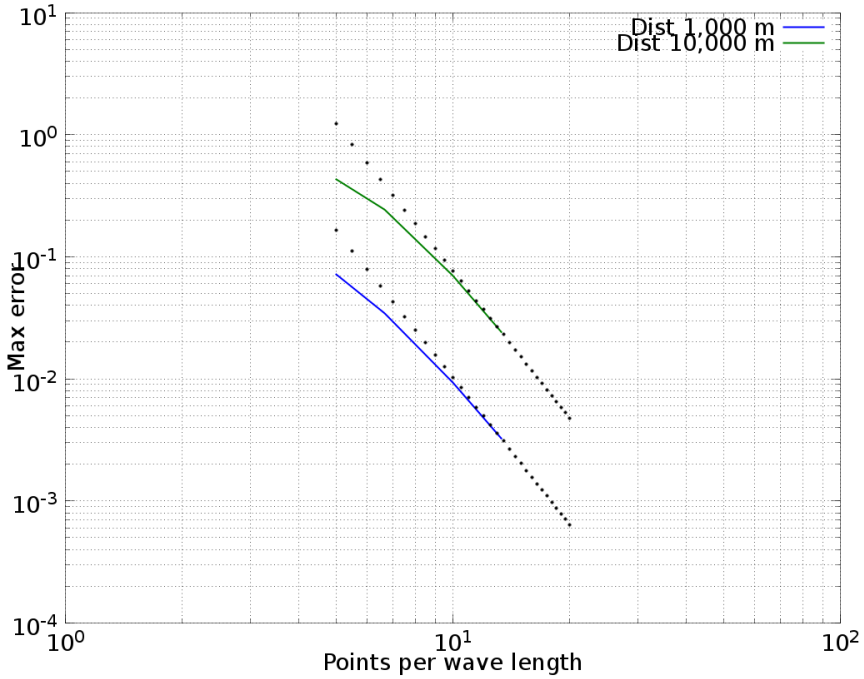


Figure 4.11: Lamb's problem: Error in max norm as function of the number of points per wave length at 1,000 meters (blue) and 10,000 meters (green) from the source. The dotted lines indicate the asymptotic decay of the error in a fourth order accurate method.

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 *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 a `rfile` raster file (§ 5.4), an `sfile` (§ 5.5), or by using a `gmg` (§ 5.6)file.

A curvilinear grid is automatically constructed between the topography surface and a user specified depth $z =$`zmax`. 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 $\tau_{min} \leq z \leq \tau_{max}$ (z is positive downwards), the grid generation usually works well if

$$\texttt{zmax} \geq \tau_{max} + 3(\tau_{max} - \tau_{min}), \tag{5.1}$$

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

$$e_{grid} \leq e_{min} - 3(e_{max} - e_{min}), \quad e_{min} \leq e(x, y) \leq e_{max}. \tag{5.2}$$

You then set `zmax`$= -e_{grid}$.

After reading the topography, *SW4* prints out the min and max z-coordinates, as well as the specified value of `zmax`. For example,

```
***Topography grid: min z = -1.1443e+03, max z = 1.0929e+03, \
   top Cartesian z = 8.000000e+03
```

Here, $\tau_{min} = -1144.3$, $\tau_{max} = 1092.9$ and `zmax=` 8000. We have $\tau_{max} + 3(\tau_{max} - \tau_{min}) = 7804.5$, which satisfies (5.1). Alternatively, in terms of elevation, $e_{max} = 1144.3$ and $e_{min} = -1092.9$, so $e_{min} - 3(e_{max} - e_{min}) = -7804.5$ and $e_{grid} = -8000$, 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 `smooth` option in the topography command. You can inspect the result of the smoothing by saving the top grid surface in an image file,

```
image mode=grid z=0 cycle=0 file=test
```

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,

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

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 \
           gaussianXc=3.6 gaussianYc=2.4 \
           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 `sfile` 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

We deprecated the ability to read an `efile` in SW4. Users are advised to use `sfile` or `gmg` formats.

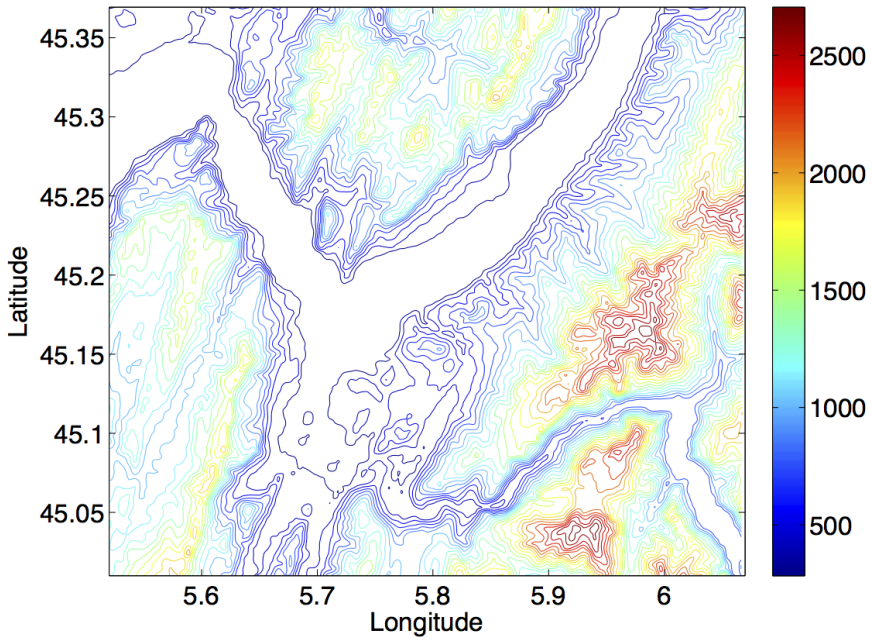


Figure 5.1: Topography in the vicinity of Grenoble, France.

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=berkeley.rfile
```

Here, the topography command tells *SW4* to read the topography from the specified `rfile`. The `order=3` option specifies the type of stretching to use when making the curvilinear grid, and the `zmax=2e3` option tells *SW4* to put the bottom boundary of the curvilinear grid at $z = 2000$.

Note that the topography must be described by the same `rfile` as the material model, see Section 6.3 for further information. The format for the `rfile` is described in Section 12.5.

5.5 sfile topography

The `sfile` also contains the same topography data as `rfile`, and can be accessed using the `topo input=sfile` argument. See Section 6.4 for more information.

5.6 gmg topography

The `gmg` also contains topography data, and can be accessed using the `topo input=gmg` argument. See Section 6.5 for more information.

Chapter 6

The material model

In *SW4*, an isotropic elastic material model is defined by the grid point values of the density (ρ), the compressional velocity (V_p), and the shear velocity (V_s). The material properties can be specified by the `block` command (§ 6.1), the `rfile` command (§ 6.3), the `pfile` command (§ 6.6), the `ifile` command (§ 6.7), the `gmg` 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 § 6.8 for details.

This chapter only discusses isotropic elastic materials. See Chapter 8 for isotropic visco-elastic material models and Section 11.3.11 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, sfile, gmg, 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 sfile 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`.

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

```
---------- Material properties ranges ---------------
1590 kg/m^3 <=  Density <= 3300 kg/m^3
768 m/s    <=  Vp      <= 7790 m/s
500 m/s    <=  Vs      <= 4420 m/s
1.536        <=  Vp/Vs   <= 4.48
3.975e+08 Pa     <=  mu      <= 6.44701e+10 Pa
1.4282e+08 Pa     <=  lambda  <= 7.13863e+10 Pa
-----------------------------------------------------
```

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=2500 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.

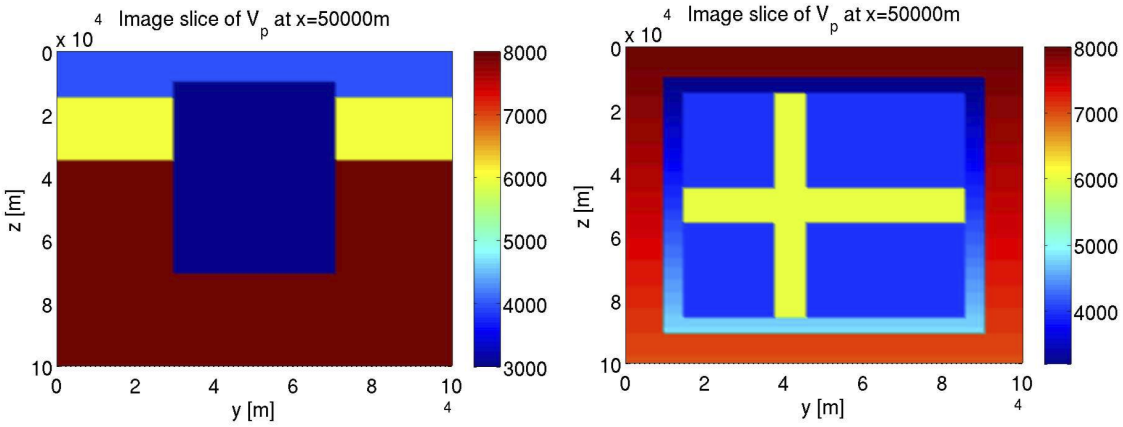


Figure 6.1: Examples of material models specified with the block command.

6.2 The efile command

We deprecated the ability to read an `efile` in SW4. Users are advised to use `sfile` or `gmg` formats.

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 (deprecated since 2021)` 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 6.2. In this case,

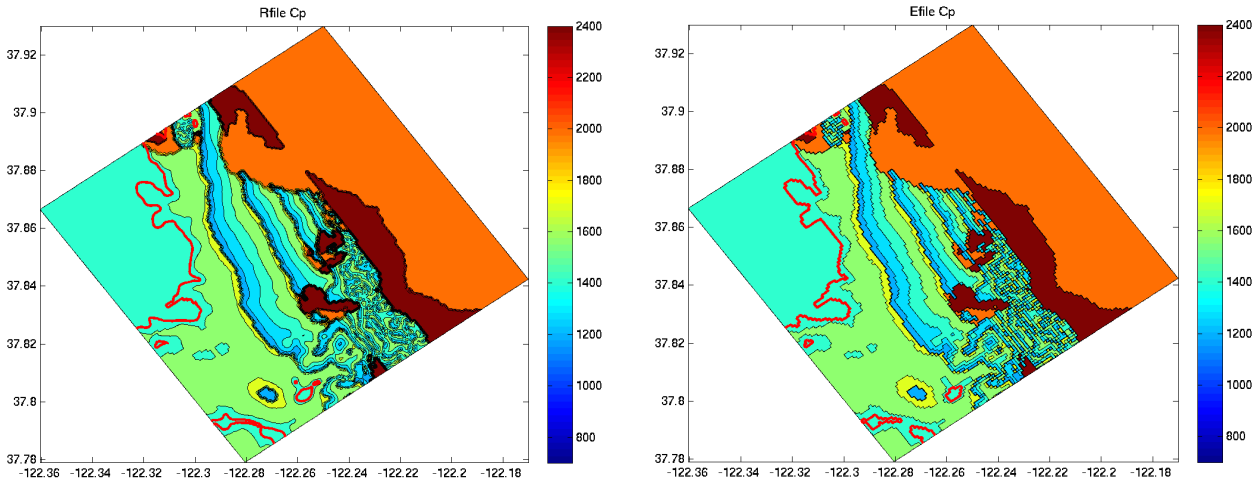


Figure 6.2: The compressional wave speed (C_P) along the topography using the `rfile` (left) and the `efile (deprecated since 2021)` (right) commands. Here, the computational grid has size $h = 20$ m, while the horizontal grid sizes of the `rfile` and `efile` are both $hh = 100$ m. Notice the stair-stepping in the right figure.

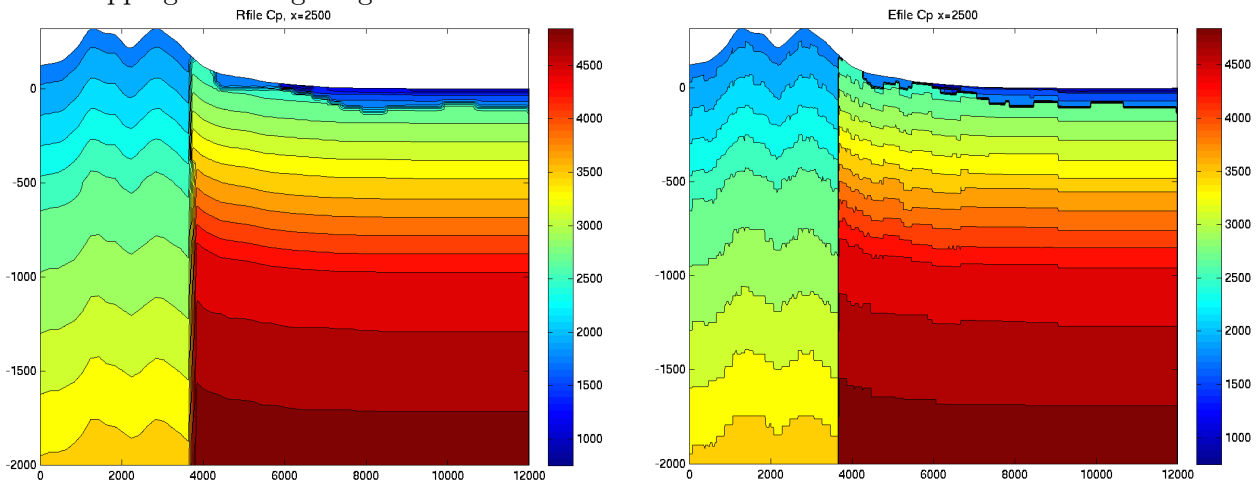


Figure 6.3: The compressional wave speed in a vertical cross-section, using the `rfile` (left) and the `efile` (right) commands. Here, the computational grid has size $h = 20$ m. Near the surface, the grid sizes of the `rfile` and `efile` are both $hh = 100$ m in the horizontal direction and $hv = 25$ m in the vertical direction. Below $z = 400$, the horizontal and vertical grid sizes are doubled.

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 6.3.

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.

6.4 The sfile command

The `sfile` command ("s" being the next letter after "r"), was created to include a depth-based curvilinear grid for the material model, which provides finer resolution close to the surface and avoids storing "air" or "water" points above the surface. It also includes the ability to write the material model for seismic inversion calculations (using the `sfileoutput` command, see *SW4-mopt* documentation).

```
sfile filename=USGSBayAreaVM-08.3.0.sfile directory=/path/to/sfile
```

The file contents are similar to the `rfile` file, but are defined with internal (curvilinear) refinement boundaries, and all 5 material properties. Note that the `rfile` caveats about the computational domain still apply, and geographical region bounds should all be checked against the `sfile`'s *origin*, (*nx,ny*), *hh*, *z_bot*, etc.

Figure 6.4 shows a schematic of the data layout in a 2D slice of the `sfile` domain. The material data grids and interfaces are numbered from the top to bottom, with the very bottom interface assumed to be flat (constant *z_bot*). Each grid is bounded above by a *z_interface_** ("*" is the same grid index) at the same constant horizontal grid resolution, *hh*. The very top interface is the finest topography used by SW4 `topo` command for the computational domain. The number of vertical points is constant, and the vertical grid resolution *hv* is constant along each grid line (but different across grid lines) when there is topography. The refinement boundaries halve the horizontal grid resolution from bottom to top, and double the number of grid intervals. In order to avoid gaps at refinement boundaries, the bottom interfaces of the upper grid are defined as co-linear points calculated from the top interface of the lower grid.

See 11.3.5 for usage information and 12.6 for HDF5 file format details.

6.5 The gmg command

The `gmg` command uses data stored in the GeoModelGrids format (developed by Brad Aagard from USGS, `https://geomodelgrids.readthedocs.io`), which has georeferenced grid-based earth models composed of blocks with different grid resolutions. The latest San Francisco Bay region 3D seismic velocity model (v21.1, released on December 9, 2021) can be downloaded at `https://www.usgs.gov/data/san-francisco-bay-region-3d-seismic-velocity-model-v211`. The `gmg` command has the same syntax with `rfile` and `sfile`:

```
gmg filename=USGS_SFCVM_v21-1_detailed.h5 directory=/path/to/gmgfile/
```

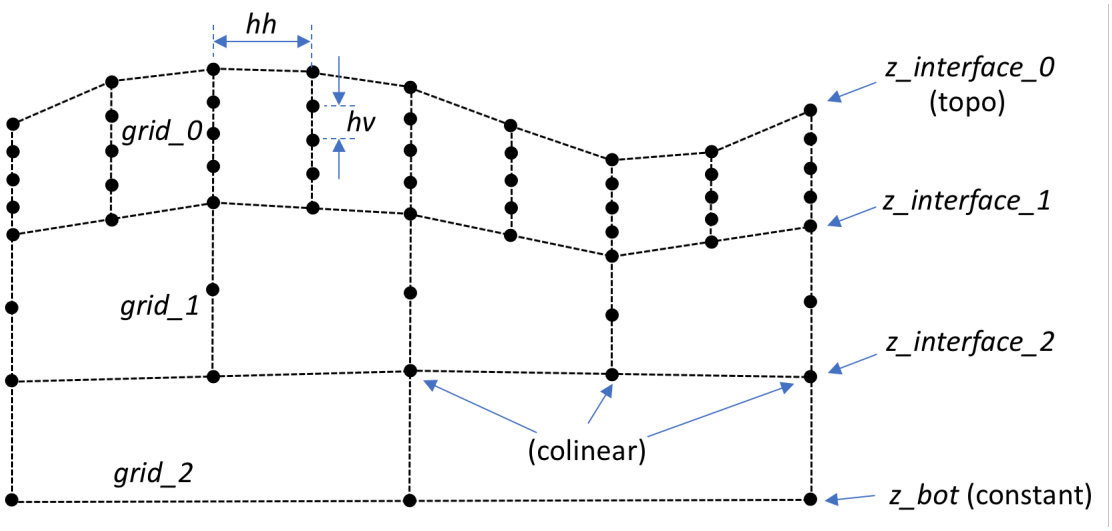


Figure 6.4: A schematic of the grid and data layout in `sfile` files. Material data lives at grid points (black circles) on grids specified by their grid index, horizontal grid spacing, and number of points along grid lines connecting the top and bottom interfaces. In this case, there are 3 total grids of material data, starting from a constant elevation at the bottom of the domain, then separated by two interior interfaces, with the domain topography on top.

The file contents are similar to the `sfile` format, both are using HDF5 as the underlying file format, but they use different schema and curvilinear refinement boundaries. More tools and descriptions about the GeoModelGrids are available at `https://github.com/baagaard-usgs/geomodelgrids` and `https://geomodelgrids.readthedocs.io`).

See Section 11.3.6 for more details options.

6.6 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 θ, latitude ϕ and depth d. Material properties are first linearly interpolated in the depth direction along each

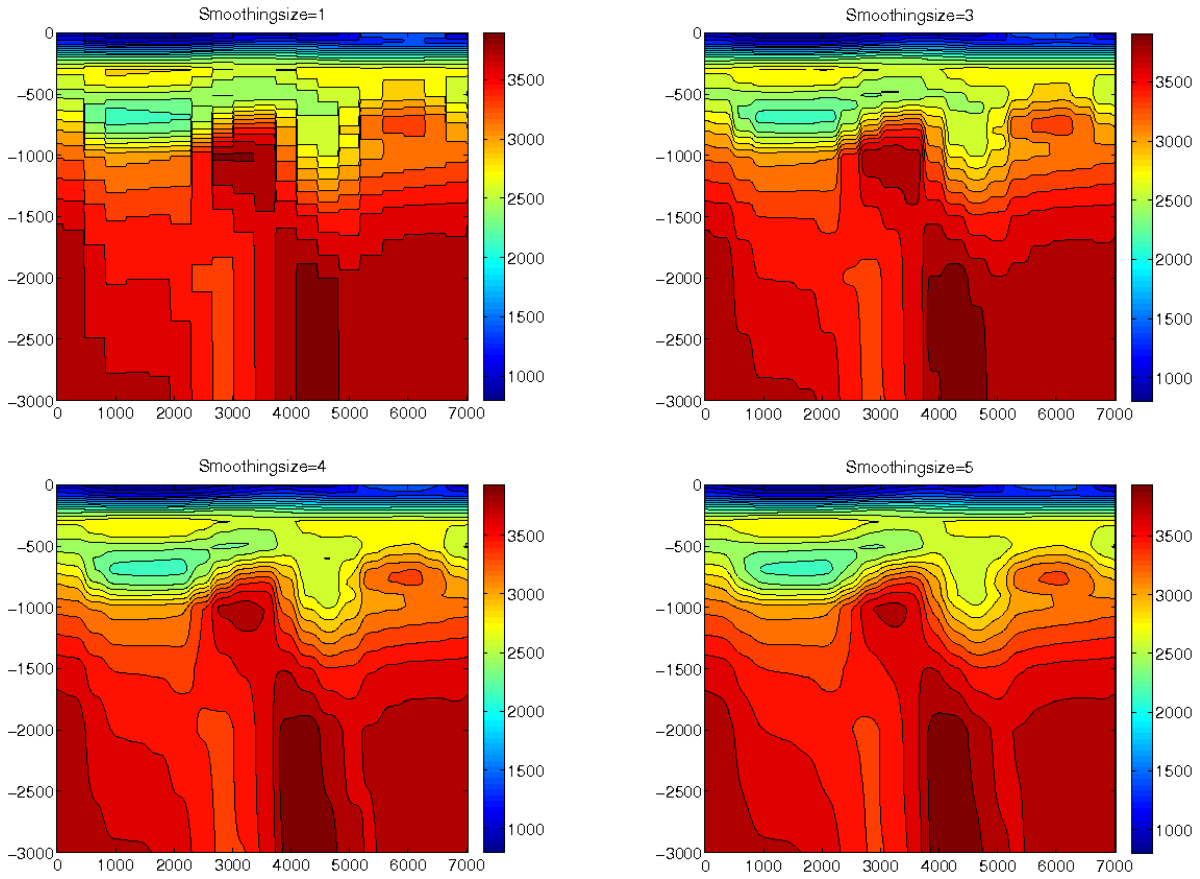


Figure 6.5: The `smoothingsize` parameter can be used to average out imprinting from the horizontal lattice in a coarse pfile material model. Here we show V_P in the plane $x = 3,500$ as function of y and $-z$. In this case, `smoothingsize`=1 in the top left, `smoothingsize`=3 in the top right, `smoothingsize`=4 in the bottom left, and `smoothingsize`=5 in the bottom right plot.

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 (≥ 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, (ϕ_i, θ_j). The material property c (ρ, V_p, V_s, Q_s, or Q_p) is assigned by the formula

$$c(\phi,\theta) = \frac{\sum_{m=i-W}^{i+W}\sum_{n=j-W}^{j+W} c_{m,n}\omega_{m,n}}{\sum_{m=i-W}^{i+W}\sum_{n=j-W}^{j+W}\omega_{m,n}}, \quad W = \frac{N_G-1}{2}, \tag{6.1}$$

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 Δ 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 α makes the weights $\omega_{m,n} < 10^{-6}$ for points that are further from (ϕ_m, θ_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:

```
pfile filename=materialxy.ppmod vsmin=1000 vpmin=1732 smoothingsize=4 \\
      style=cartesian
```

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.

6.7 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 gegraphic 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 ϕ_i, latitude θ_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. Similarily, 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.,

$$0 \leq d \leq d_{Nmat,i,j}.$$

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.

6.8 The vimaterial command

The `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 `grid` and `topography` commands must be identical between runs. An initial run must construct the material model using the commands described in the previous sections (block, pfile, ifile). The initial run must also save the material model using the `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 `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 `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 `volimage` files,

```
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=rfile file=USGSBayAreaVM-08.3.0.rfile zmax=8e3 order=3
attenuation nmech=3 phasefreq=1 maxfreq=10

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
```

The `volimage` files are written in the `MaterialModel` subdirectory, as specified by the `path` option in the `fileio` command.

Subsequent runs, that use identical grid and topography commands, can read the material model directly from the `volimage` files,

```
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=rfile file=USGSBayAreaVM-08.3.0.rfile 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
```

Note that a path can be specified in the `vimaterial` command, and that the `fileio` command now writes the results to a different directory.

Chapter 7

Mesh refinement

The refinement command in *SW4* 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 *SW4*, 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,

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

The two refinement commands

```
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. However, 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. *SW4* will stop with an error message if the number of grid points in the z-direction is below either of these limits.

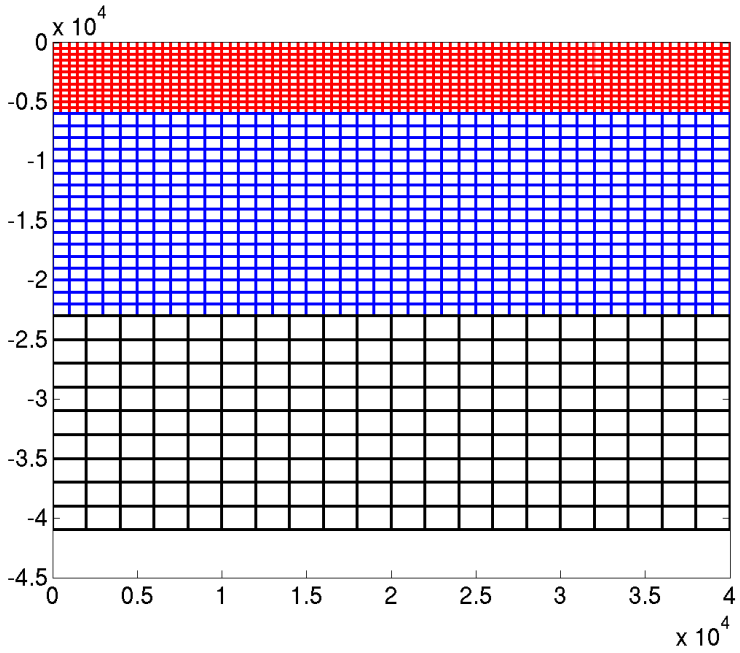


Figure 7.1: Composite grid with two mesh refinement interfaces and three Cartesian grids.

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

```
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 zmax=6e3 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 `zmax=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

```
Global grid sizes (without ghost points)
```

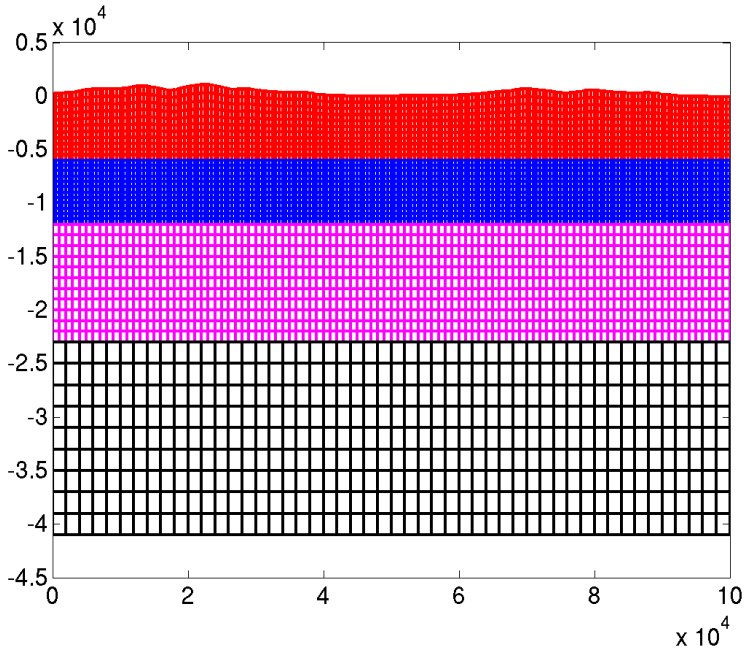


Figure 7.2: A composite grid with two mesh refinement interfaces and topography. In this case there are three Cartesian components and one curvilinear grid following a non-planar topography.

```
Grid          h         Nx         Ny         Nz       Points
   0       2000         51         51         10        26010
   1       1000        101        101         12       122412
   2        500        201        201         13       525213
   3        500        201        201         12       484812
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, *SW4*, 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

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

where the spatial operator is

$$\mathbf{L}(\lambda, \mu)\mathbf{u} =: \nabla(\lambda(\nabla \cdot \mathbf{u})) + \nabla \cdot (2\mu \mathcal{D}(\mathbf{u})), \quad \mathcal{D}(\mathbf{u}) = \frac{1}{2}\left(\nabla \mathbf{u} + \nabla \mathbf{u}^T\right). \tag{8.2}$$

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

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

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 $\mathbf{u}$ and $\bar{\mathbf{u}}^{(\nu)}$, $\nu = 1, 2, \ldots, n$, 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 μ_ν and λ_ν, as well as the relaxation frequencies ω_ν are determined by Emmerich and Korn's [6] 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

$$\omega_{min} \leq \omega \leq \omega_{max}.$$

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

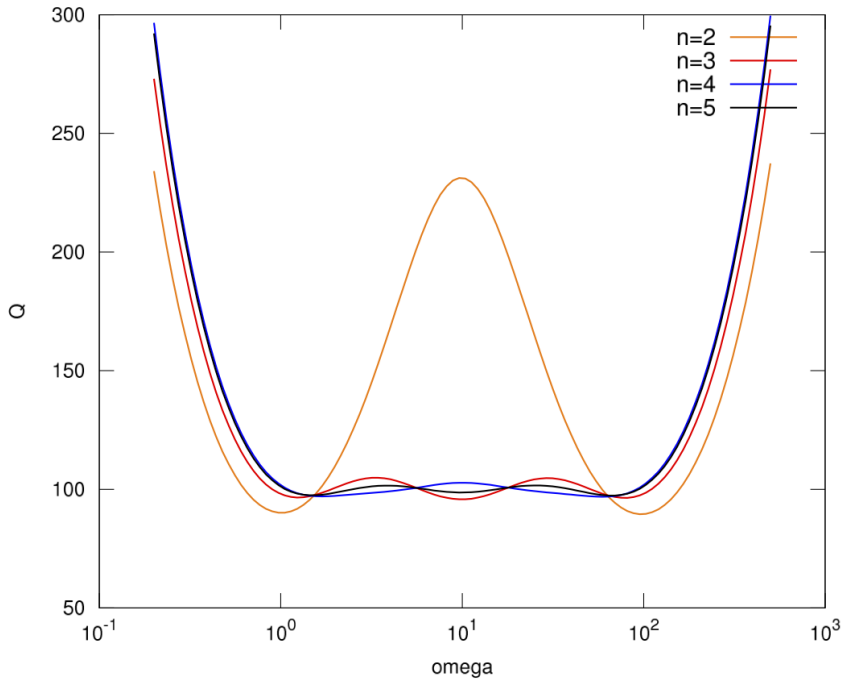


Figure 8.1: Actual quality factor $Q(\omega)$ approximating $Q_0 = 100$ in the frequency band $\tilde{\omega} \in [1, 100]$, for different numbers of viscoelastic mechanisms.

n further only leads to small improvements. It is interesting to note that in all models, $Q(\omega)$ grows rapidly for $\omega > \omega_{max}$. 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 ω, throughout the frequency band $[\omega_{min}, \omega_{max}]$. 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, ω_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 $f_{max} = 10$ 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,

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

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

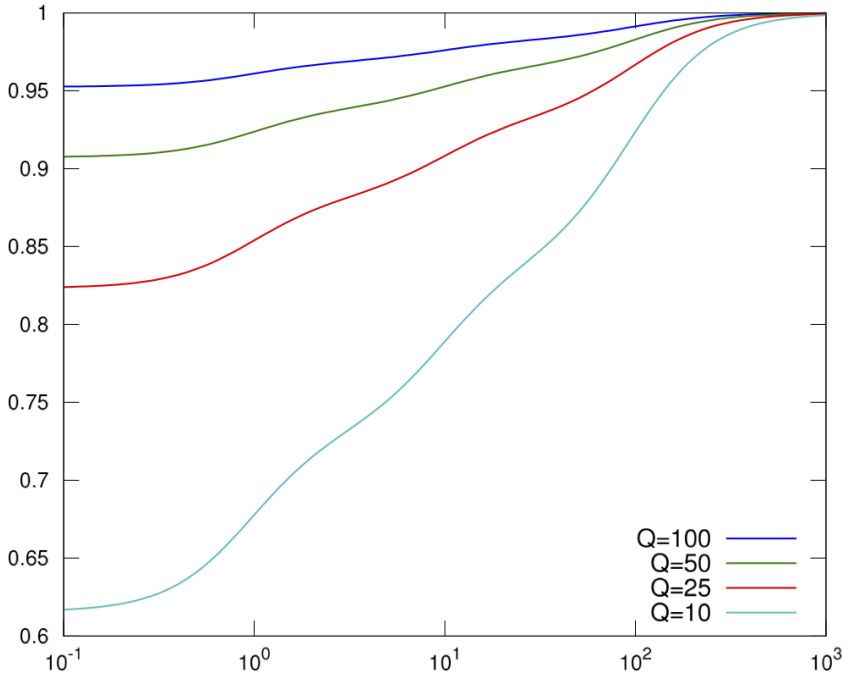


Figure 8.2: Relative phase velocity over the frequency band $\omega \in [1, 100]$. Here, $n = 3$, and the different colors correspond to different values of Q.

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 `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 `minppw` keyword by `maxfreq=`f_{max}.

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

```
*** 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 `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/lscratcha/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 $\mathbf{u}(\mathbf{x}_r, t) = (u_x, u_y, u_z)^T$ 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:

`variables`	`nsew=0`	`nsew=1`
displacement	.x, .y, .z	.e, .n, .u
velocity	.xv, .yv, .zv	.ev, .nv, .uv

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.curly`, 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 The SAC HDF5 format

SW4 can also output receiver time-histories in the HDF5 format,

```
rec lat=38.25 lon=-122.20 depth=0 file=sta1 hdf5file=sta.hdf5 sacformat=0 hdf5format=1
```

The HDF5 file holds all components (usually three, but only one when `variables=div`, and six when `variables=strains`) in a single file named `sta.hdf5`. When the `hdf5format=1` and `hdf5file=sta.hdf5` option is used. Note that you need to give the `sacformat=0` option unless you want the solution to be output in both formats.

9.2.3 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 12.8.
- The ASCII text format is very simple and is outlined in its header.
- The HDF5 format is described in Section 12.9.
- 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 11.5.3 for details.

The cross-sectional plane is specified by a Cartesian coordinate (x, y, 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

```
picturefile.cycle=100.x=1000.div.sw4img
picturefile.cycle=200.x=1000.div.sw4img
...
```

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 $\max(|u_t^N|, |u_t^E|)$, 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 maximuma 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 12.10 for details.
- We provide matlab/octave scripts for reading image files in the `tools` directory. The basic function is called `readimage.m`.

9.4 Checkpoint and Restart the checkpoint command

SW4 can write out checkpoint files with a fixed time step interval. The checkpoint file can be used for restarting a simulation from when the latest checkpoint is written. SW4 writes out one single file for each checkpoint, and user can choose to use either a binary format or the HDF5 format. Checkpoint files can be very large, as it contains data of the entire domain. For the HDF5 format, compression is also supported (with the same parameter as the `ssioutput` command), users are advised to set a relatively low tolerance with the lossy compression to prevent significant precision loss with the restart. And user should also be aware that a restart from a checkpoint file that uses lossy compression will likely result in different solution results.

Below is an example command to write a checkpoint file every 80,000 steps to the `output` directory. The checkpoint file name will be starting with `HFCheck` and includes the time step value when it is written (e.g. HFCheck.cycle=160000.sw4checkpoint).

```
checkpoint cycleInterval=80000 restartpath=output file=HFCheck
```

To use the HDF5 format and optionally with compression:

```
checkpoint cycleInterval=80000 restartpath=output file=HFCheck \
    hdf5=yes zfp_accuracy=1e-5
```

To restart from a previously written checkpoint file, add the `restartfile` key word and the checkpoint file name to the `checkpoint` command:

```
checkpoint cycleInterval=80000 restartpath=output file=HFCheck \
    restartfile=HFCheck.cycle=160000.sw4checkpoint \
    hdf5=yes zfp_accuracy=1e-5
```

Note that when using the `checkpoint` command in combination with the `rec`, `rechdf5`, and `ssioutput` commands, it is advisable to align their write intervals, such that the latest time-series data are also written at the time writing a checkpoint file. For example, a user may set `writeEvery=1000` (default) in the `rechdf5` command, and set `dumpInterval=400` in the `ssioutput` command, and set `cycleInterval=40000`, as 40000 is divisible by both 1000 and 4000.

More details are provided in Section 11.5.9

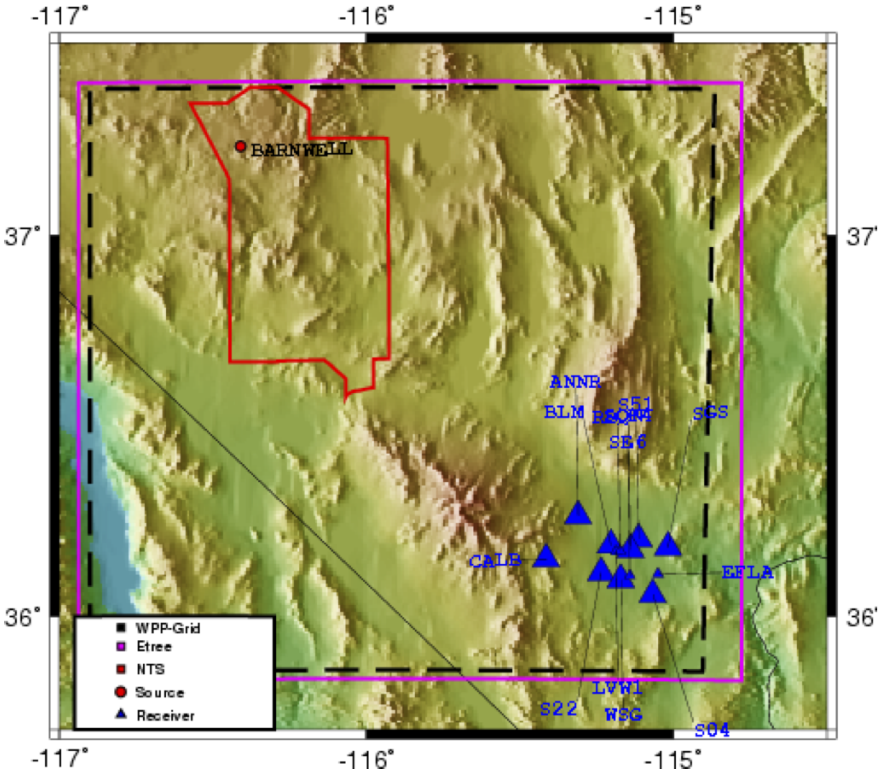


Figure 9.1: Location of the source and stations for the Barnwell simulation. This figure was generated using the GMT command, see Section 11.5.7 for details.

9.5 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

```
gmt file=bolinas.gmt
```

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/scec/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 choosen to be $h = 50\,m$ and the material properties in the different layers are described by

```
grid h=50 x=30000 y=30000 z=17000
block vp=4000 vs=2000 rho=2600
block vp=6000 vs=3464 rho=2700 z1=1000
block vp=4630.76 vs=2437.56 rho=2650 z1=999 z2=1001
```

The last `block` command defines the properties right at the interface, using a harmonic average for the Lamé parameters (μ, λ), and a artithmetic average for denisty. 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:

```
time t=9
source x=15000 y=15000 z=2000 mxy=1e18 t0=0.36 freq=16.6667 \
       type=Gaussian
```

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

```
image mode=mag z=0 file=surf timeInterval=0.5
```

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

```
rec x=15600 y=15800 z=0 file=sta01 usgsformat=1
rec x=16200 y=16600 z=0 file=sta02 usgsformat=1
rec x=16800 y=17400 z=0 file=sta03 usgsformat=1
rec x=17400 y=18200 z=0 file=sta04 usgsformat=1
rec x=18000 y=19000 z=0 file=sta05 usgsformat=1
rec x=18600 y=19800 z=0 file=sta06 usgsformat=1
rec x=19200 y=20600 z=0 file=sta07 usgsformat=1
rec x=19800 y=21400 z=0 file=sta08 usgsformat=1
rec x=20400 y=22200 z=0 file=sta09 usgsformat=1
rec x=21000 y=23000 z=0 file=sta10 usgsformat=1
```

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 scipt in `tools/loh1exact.m`. The syntax is

```
octave:4> [t ra tr ve]=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 `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.

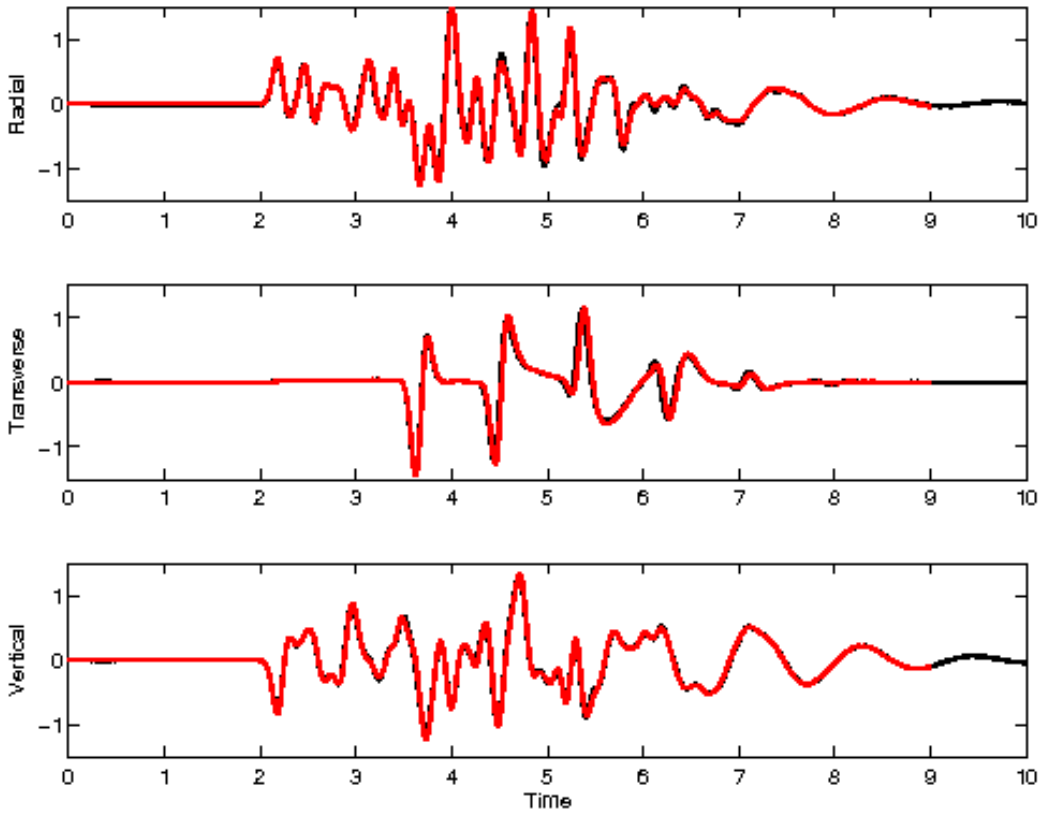


Figure 10.1: LOH.1: The radial (top), transverse (middle) and vertical (bottom) velocities for receiver number 10. Here the numerical solutions are plotted in red ($h = 50$) and the semi-analytical solution is plotted in black.

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

```
block vs=3464 vp=6000 rho=2700 Qs=69.3 Qp=155.9
block vs=2000 vp=4000 rho=2600 z2=1000 Qs=40 Qp=120
block vs=2437.6 vp=4630.8 rho=2650 Qs=54.65 Qp=137.95 z1=999 z2=1001
```

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 `freq`$= 1/\sigma = 20$ rad/s and center frequency $f_0 = 20/(2\pi) \approx 3.18$ Hz. The source is specified by the command

```
source x=15000 y=15000 z=2000 mxy=1e18 t0=0.3 freq=20 \
       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 `tools/loh3exact.m`. The syntax is

```
octave:15> [t ra tr ve]=loh3exact(0.05);
```

As for the LOH.1 problem, the semi-analytical solution is given in polar components. The solution from *SW4* is saved in Cartesian coordinates and the numerical solution at station 10 can be read into Octave using the command,

```
octave:23> [t1 ux uy uz]=readusgs("sta10.txt");
```

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

```
octave:25> ur =  0.6*ux + 0.8*uy;
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

```
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 $f_0 \approx 3.18$ Hz and we estimate the upper power frequency to be $f_{max} \approx 2.5 f_0 = 7.95$ Hz. The material model has $\min V_s = 2000$ m/s where the grid size is $h = 50$ m, and we arrive at

$$P = \frac{2000}{50 \cdot 7.95} \approx 5.03.$$

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 domainant wave length is approximately $2000/3.18 \approx 628.9$ 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.

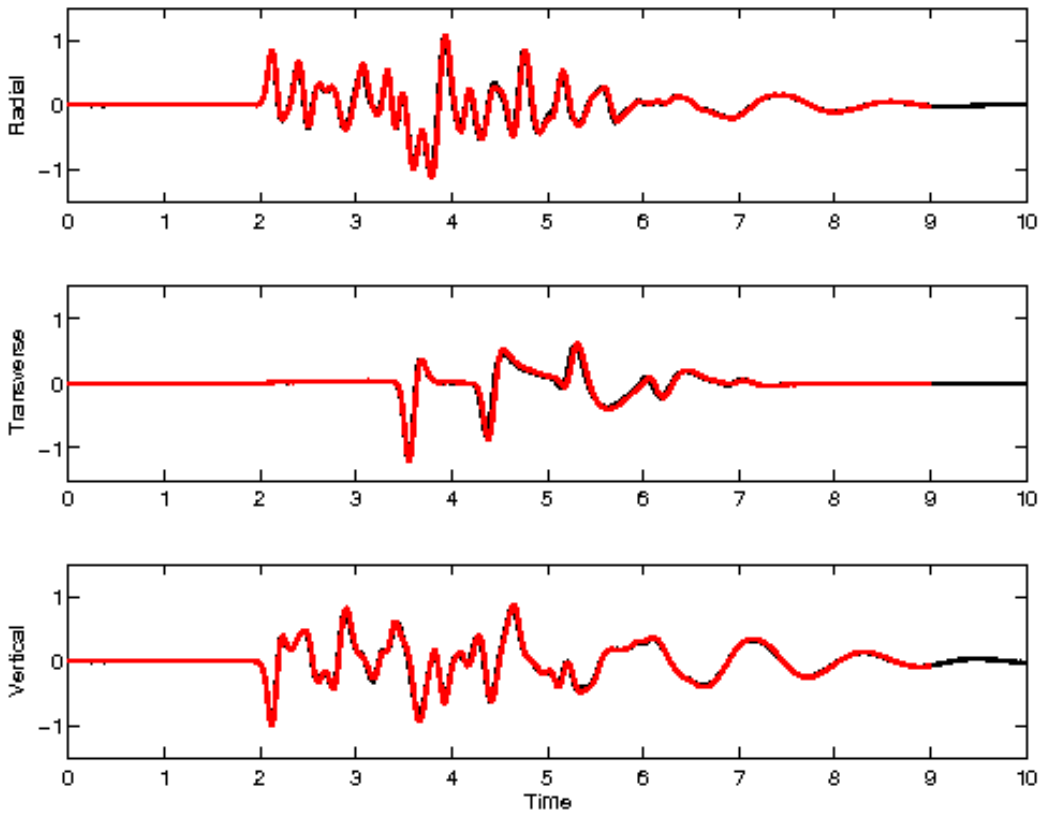


Figure 10.2: LOH.3: The radial (top), transverse (middle) and vertical (bottom) velocities at receiver number 10. Here the numerical solution with grid size $h = 50$ is plotted in red and the semi-analytical solution is plotted in black.

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, rfile, sfile, and gmg) 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, rfile, sfile, 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 specifed by at least one of the (block, pfile, ifile, rfile, sfile, gmg) 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*.

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

fileio command parameters			
Option	**Description**	**Type**	**Default**
path	path to a directory where all output will be written	string	.
verbose	sets the level of diagnostic messages written to standard out (≥ 0)	int	0
printcycle	sets the interval for printing the cycle, time, dt info	int	100
pfs	assume a parallel (1) or serial (0) file system when writing files (several processes can simultaneously write the same file on a parallel file system)	int	0
nwriters	set the number of processes that write an image or volimage file	int	8

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=...**
- lenghts in all three directions and the grid size: **x=... y=... z=... h=...**

- lenghts 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=...**.

grid command parameters (part 1)				
Option	**Description**	**Type**	**Units**	**Default**
x	physical dimension of grid in the x-direction	real	m	none
y	physical dimension of grid in the y-direction	real	m	none
z	physical dimension of grid in the z-direction	real	m	none
h	grid spacing	real	m	none
nx	number of grid points in the x-direction	int	none	none
ny	number of grid points in the y-direction	int	none	none
nz	number of grid points in the z-direction	int	none	none

The default projection is spheriodal as described by equations (3.6)-(3.7). You can change the parameter M with the **mlat** keyword. By using the **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 *SW4* was built with Proj4 support). These projections are enabled by using one of the keywords **proj**, **ellps**, **datum**, **scale**, **lat_p**, or **lon_p**. The values assigned to these keywords are used to generate a string which is passed directly to the `pj_init_plus` routine in the Proj4 library. For example, the `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 **lat_p** and **lon_p** are passed as arguments to **lat_0** and **lon_0**, respectively. See the Proj4 documentation for further guidance.

grid command parameters (geographical coordinates and projection)				
Option	**Description**	**Type**	**Units**	**Default**
az	clockwise angle from North to the x-axis	real	degrees	135.0
lat	latitude geographical coordinate of the origin	real	degrees	37.0
lon	longitude geographical coordinate of the origin	real	degrees	-118.0
mlat	meters per degree of latitude (spheroidal projection)	real	meters	111,319.5
mlon	meters per degree of longitude (spheroidal projection)	real	meters	None
proj	name of projection (see proj4 documentation)	string	None	utm
ellps	name of ellipse (see proj4 documentation)	string	None	WGS84
datum	datum of projection (e.g. NAD83)	string	None	None
lon_p	central meridian of projection	string	degrees	lon
lat_p	latitude of projection origin	string	degrees	lat
scale	scale factor for central meridian	string	None	None

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

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				
Option	**Description**	**Type**	**Units**	**Default**
gp	Number of grid points in the supergrid layer	int	none	30
width	Physical width of the supergrid layer	float	meters	none
dc	Damping coefficient in supergrid region	real	none	0.02

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 causual but gives the filtered signal a phase shift. When **passes=2** the filter has zero phase shift, but is acausual. In this case, the filtered signal is only exponentially small as $t \rightarrow -\infty$. 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.

prefilter command parameters				
Option	**Description**	**Type**	**Units**	**Default**
fc1	first (low) corner frequency in the filter (> 0)	real	Hz	0.1
fc2	second (high) corner frequency in the filter (> 0)	real	Hz	1.0
type	lowpass or bandpass	string	None	bandpass
passes	number of passes (1 or 2)	int	None	2
order	order of filter (1-10)	int	None	2

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 (**lat, lon, depth** or **topodepth**) coordinates. The depth below mean sealevel ($z = 0$) is specified with **z**, while **depth** or **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, **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 **type** keyword), or give the file name for a discrete time function (using the **dfile** keyword).

Note that all pre-defined time functions use the **t0** keyword, and all functions except **Dirac** also use the **freq** keyword. The only pre-defined time function that uses the **ncyc** keyword is **GaussianWindow**.

source command parameters (part 1)				
Option	**Description**	**Type**	**Units**	**Default**
x	x position of the source (within domain)	real	m	none
y	y position of the source (within domain)	real	m	none
z	z position of the source (within domain)	real	m	none
depth	depth of the source (≥ 0)	real	m	none
topodepth	(same as depth)	real	m	none
lat	latitude geographical coordinate of the source	real	degrees	none
lon	longitude geographical coordinate of the source	real	degrees	none
t0	offset in time (≥ 0)	real	s	0.0
freq	frequency (> 0) (not used for Dirac)	real	Hz or rad/s	1.0
type	Name of source time function	string	none	RickerInt
ncyc	Number of cycles (must be specified for the GaussianWindow function)	int	none	0
dfile	File name for discrete time function	string	none	none

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)				
Option	**Description**	**Type**	**Units**	**Default**
m0	moment amplitude	real	Nm	1.0
mxx	xx-component of the moment tensor	real	Nm	0.0
myy	yy-component of the moment tensor	real	Nm	0.0
mzz	zz-component of the moment tensor	real	Nm	0.0
mxy	xy-component of the moment tensor	real	Nm	0.0
mxz	xz-component of the moment tensor	real	Nm	0.0
myz	yz-component of the moment tensor	real	Nm	0.0
strike	strike angle (see Aki and Richards)	real	degrees	none
dip	dip angle	real	degrees	none
rake	rake angle	real	degrees	none

source command parameters (point force)				
Option	**Description**	**Type**	**Units**	**Default**
f0	point force amplitude	real	N	1.0
fx	forcing function in the x direction	real	N	0.0
fy	forcing function in the y direction	real	N	0.0
fz	forcing function in the z direction	real	N	0.0

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_Rup`

Note that when using a rupture file, the default SW4 time-series output is usually velocity instead of displacement.

11.2.3 rupturehdf5

Syntax:
`rupturehdf5 file=...`
Required parameters:
name of SRF file in HDF5 format.

The `rupturehdf5` command is similar to the `rupture` command, it reads a SRF-HDF5 file that is converted from a SRF file using the python script found at sw4/tools/srf2hdf5.py.

The SRF-HDF5 file format is described in Section 12.7

11.3 The material model [required]

It is required that the material model is defined in the entire computational domain. The material properies 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`, `rfile`, `sfile`, `gmg`, 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 particularily 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`.

attenuation command parameters				
Option	**Description**	**Type**	**Unit**	**Default**
phasefreq	The frequency (> 0) at which C_S and C_P are specified	real	Hz	1.0
nmech	Number of SLS mechanisms to approximate constant Q_P and Q_S (between 0 and 8)	int	None	3
maxfreq	The upper frequency limit (> 0) for approximating constant Q_P and Q_S	real	Hz	2.0
minppw	Calculate the upper frequency limit based on this number of grid points per shortest wave length (> 0)	real	None	None
qmultiplier	A number that will multiply both Q_P and Q_S	real> 0	None	1.0

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,

$$Q_P := \texttt{qmultiplier} \times Q_P \qquad Q_S := \texttt{qmultiplier} \times Q_S.$$

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., `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 $\mathbf{x}_{i,j,k}$ is simply multiplied by the factor

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

where δ_t is the time step, f_c is the center frequency, and $Q = Q_S(\mathbf{x}_{i,j,k})$. 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 **absdepth** flag determines how the z-coordinates are used. If **absdepth**=0 (default) **z1=...** and **z2=...** specify depths below the free surface. If **absdepth**=1, **z1=...** and **z2=...** bound the z-coordinate of the material block.

The gradient parameters **vpgrad**, **vsgrad**, and **rhograd** specify linear variations in the z-direction (downward). The units for **vpgrad** and **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 ($z = 0$ or depth=0 with topography), e.g.,

$$C_p(z) = \mathbf{vp} + z\,\mathbf{vpgrad}.$$

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

block command parameters (part 1)				
Option	**Description**	**Type**	**Units**	**Default**
vp	P-wave velocity	real > 0	m/s	none
vs	S-wave velocity	real > 0	m/s	none
rho	density	real > 0	kg/m^3	none
vpgrad	vertical gradient for vp	real	m/s/m	none
vsgrad	vertical gradient for vs	real	m/s/m	none
rhograd	vertical gradient for rho	real	kg/m^4	none
qp (or Qp)	P-wave quality factor	real > 0	none	none
qs (or Qs)	S-wave quality factor	real > 0	none	none

block command parameters (part 2)				
Option	**Description**	**Type**	**Units**	**Default**
x1	minimum x-dim for the box shaped sub-region	real	m	-max x
x2	maximum x-dim for the box shaped sub-region	real	m	2 max x
y1	minimum y-dim for the box shaped sub-region	real	m	-max y
y2	maximum y-dim for the box shaped sub-region	real	m	2 max y
z1	minimum z-dim for the box shaped sub-region	real	m	-max z
z2	maximum z-dim for the box shaped sub-region	real	m	2 max z
absdepth	`z1` and `z2` relative to topography (0), or absolute z-coordinate (1)	int	none	0

11.3.3 pfile

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

pfile command parameters				
Option	**Description**	**Type**	**Units**	**Default**
filename	name of input pfile	string	none	none
directory	name of directory for the input pfile	string	none	.
smoothingsize	smooth data over stencil of this width (≥ 1)	int	none	5
vpmin	minimum threshold value for C_p	real	m/s	0
vsmin	minimum threshold value for C_s	real	m/s	0
rhomin	minimum threshold value for density	real	m/s	0
flatten	Flatten the earth model (T or F)	string	none	F
style	type of grid data: `geographic` or `cartesian`	string	none	geographic

11.3.4 rfile

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

rfile command parameters				
Option	**Description**	**Type**	**Units**	**Default**
filename	name of raster file	string	none	none
directory	name of directory for the raster file	string	none	.

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

11.3.5 sfile

Syntax:
`sfile filename=... directory=...`
Required parameters:
`filename`

sfile command parameters				
Option	**Description**	**Type**	**Units**	**Default**
filename	name of raster file	string	none	none
directory	name of directory for the raster file	string	none	.

Note that sfile uses HDF5 format, so SW4 must be compiled with HDF5. Also only relative paths are currently supported in the `filename` keyword. Use the `directory` keyword to specify absolute paths.

11.3.6 gmg

Syntax:
`gmg filename=... directory=...`
Required parameters:
`filename`

gmg command parameters				
Option	**Description**	**Type**	**Units**	**Default**
filename	name of GMG file	string	none	none
directory	name of directory for the GMG file	string	none	.

Note that GMG uses HDF5 format, so SW4 must be compiled with HDF5. Also only relative paths are currently supported in the `filename` keyword. Use the `directory` keyword to specify absolute paths.

11.3.7 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.

ifile command parameters			
Option	**Description**	**Type**	**Default**
filename	name of input file holding material surfaces	string	None
input	`cartesian` or `geographic`	string	None

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.

11.3.8 material

Syntax:
`material id=... vp=... vs=... rho=... vpgrad=... vsgrad=... rhograd=... vp2=... vs2=... rho2=... vpsqrt=... vssqrt=... rhosqrt=... qp=... 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.

material command parameters (constants)			
Option	**Description**	**Type**	**Default**
id	material ID number > 0	int	None
vp	P-wave velocity	real	None
vs	S-wave velocity	real	None
rho	Density	real	None
qp or Qp	P-wave quality factor	None	None
qs or Qs	S-wave quality factor	None	None

material command parameters (gradients)			
Option	**Description**	**Type**	**Default**
vpgrad	P-velocity gradient	real	0.0
vsgrad	S-velocity gradient	real	0.0
rhograd	Density gradient	real	0.0

material command parameters (higher order)			
Option	**Description**	**Type**	**Default**
vp2	P-velocity quadratic coefficient	real	0.0
vs2	S-velocity quadratic coefficient	real	0.0
rho2	Density quadratic coefficient	real	0.0
vpsqrt	P-velocity $\sqrt{z}$ coefficient	real	0.0
vssqrt	S-velocity $\sqrt{z}$ coefficient	real	0.0
rhosqrt	Density $\sqrt{z}$ coefficient	real	0.0

11.3.9 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.

globalmaterial command parameters			
Option	**Description**	**Type**	**Default**
vpmin	Minimum P-wave velocity (> 0)	real	None
vsmin	Minimum S-wave velocity (> 0)	real	None

11.3.10 randomblock [optional]

Syntax:
`randomblock corrlen=... corrlenz=... sigma=... hurst=... zmin=... zmax=... seed=...`
Required parameters:
None

The `randomblock` command adds a random perturbation to the material velocities C_s and C_p. SW4 adds the random perturbation after the specified material model has been read. The randomblock command will work together with any of the material commands described in this section. 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 random perturbation uses the von Karman self-similar correlation function, which has energy spectrum proportional to

$$\frac{1}{(1+a^2k^2)^{H+3/2}}$$

where $a^2k^2 = a_h^2k_x^2 + a_h^2k_y^2 + a_v^2k_z^2$, and H is the Hurst exponent. a_h and a_v are normalized horizontal and vertical correlation lenghts, respectively. The material velocities are updated by the random block according to

$$C_s := C_s * (1 + \theta) \qquad C_p := C_p * (1 + \theta)$$

where θ is a generated random number, scaled to have mean zero and standard deviation σ.

The the limits `zmin` and `zmax` allow specification of different correlation lenghts and/or Hurst exponents at different depths, by using more than one `randomblock` command. If no limits are given, the random perturbation will be applied over the entire computational domain.

Unlike the old `randomize` command, the random numbers are not globally defined, i.e., two runs using different number of processors will use different sequences of random numbers, even if the random seeds are the same in the two runs.

Randomblock command parameters				
Option	**Description**	**Type**	**Units**	**Default**
corrlen	Horizontal correlation length	real	m	1000
corrlenz	Vertical correlation length	real	m	corrlen
sigma	standard deviation of perturbation, σ	real	None	0.1
hurst	Hurst exponent in correlation fcn.	real	None	0.3
zmin	Minimum z-coordinate of random block	real	m	min z of domain
zmax	Maximum z-coordinate of random block	real	m	max z of domain
seed	Random number generator seed	int	None	read from `/dev/urandom`

11.3.11 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 6×6 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 11.3.12. Material commands for the isotropic model will be ignored when the **anisotropy** command is used.

11.3.12 ablock [optional]

Syntax:

```
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 polynomical 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 $z = z_{max}$ (recall that z is directed downwards). If the elevation 'e' of the topography ranges between $e_{min} \leq e \leq e_{max}$, we recommend using $z_{max} \geq -e_{min} + 2|e_{max} - e_{min}|$.

There are five ways of specifying the topography:

- **input=geographic** Read the topography from a file, where the coordinates are given as as 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=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)				
Option	**Description**	**Type**	**Units**	**Default**
input	Type of input: geographic (or grid), cartesian or gaussian	string	none	none
file	File name if input=geographic or input=cartesian	string	none	none
zmax	z coordinate of the interface between Cartesian and curvilinear grid	real	m	0
order	Interpolation order (2-6)	int	none	4
smooth	Number of smoothing iterations of topography grid surface	int	none	10

topography command parameters (Gaussian Hill)				
Option	**Description**	**Type**	**Units**	**Default**
gaussianAmp	Amplitude for a Gaussian hill topography	real	meters	0.05
gaussianXc	x-coordinate of center for a Gaussian Hill	real	meters	0.5
gaussianYc	y-coordinate of center for a Gaussian Hill	real	meters	0.5
gaussianLx	Width of the Gaussian hill in the x-direction	real	meters	0.15
gaussianLy	Width of the Gaussian hill in the y-direction	real	meters	0.15

11.4.2 refinement [optional]

Each **refinement** command corresponds to a mesh refinement patch for $z \leq$ **zmax**. 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:
`zmax`

refinement command parameters				
Option	**Description**	**Type**	**Unit**	**Default**
zmax	maximum z-coordinate for the refinement region	real	m	None

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=... hdf5format=... variables=... hdf5file=... downsample=...`
Required parameters:
Location of the receiver in Cartesian or geographical coordinates.

The file format is described in Section 12.8.

sac command parameters (part 1)				
Option	**Description**	**Type**	**Units**	**Default**
x	x position of the receiver	real	m	0.0
y	y position of the receiver	real	m	0.0
z	z position of the receiver	real	m	0.0
lat	latitude geographical coordinate of the receiver	real	degrees	none
lon	longitude geographical coordinate of the receiver	real	degrees	none
depth	depth of the receiver (below topography)	real	m	none
topodepth	depth of the receiver (same as depth)	real	m	none

sac command parameters (part 2)				
Option	**Description**	**Type**	**Units**	**Default**
file	file name	string	none	station
sta	name of the station	string	none	(same as file)
hdf5file	file name	string	none	station
writeEvery	cycle interval to write the file to disk	int	none	1000
usgsformat	output all components in an ASCII text file	int	none	0
sacformat	output each component in a SAC file	int	none	1
hdf5format	output all components in an HDF5 file	int	none	0
nsew	output (x,y,z)-components (0) or East, North, and vertical ($-z$) components (1)	int	none	0
variables	displacement, velocity, div, curl, strains, or displacementgradient	string	none	displacement
downsample	only available when hdf5format=1, outputs all components with the given downsample rate, resulting in less output data size	int	none	1

Let $\mathbf{u} = (u^{(x)},\ u^{(y)},\ u^{(z)})$ denote the solution of (3.1) computed by SW4. The `variables` output options have the following meaning. `displacement` gives the three components of the computed solution, $\mathbf{u}$. `velocity` gives the three components of the time derivative, $\mathbf{u}_t$. `div` gives a single variable containing the divergence of $\mathbf{u}$. `curl` gives the three components of the curl of $\mathbf{u}$. `strains` give the six strain components in order: $u_x^{(x)}$,$u_y^{(y)}$,$u_z^{(z)}$,$(u_x^{(y)} + u_y^{(x)})/2$, $(u_x^{(z)} + u_z^{(x)})/2$, and $(u_z^{(y)} + u_y^{(z)})/2$. `displacementgradient` gives the nine components: $u_x^{(x)}$, $u_y^{(x)}$, $u_z^{(x)}$, $u_x^{(y)}$, $u_y^{(y)}$, $u_z^{(y)}$, $u_x^{(z)}$, $u_y^{(z)}$, and $u_z^{(z)}$.

The geographic coordinate option `nsew=1` is only effective when `variables` is `displacement`, `velocity`, or `div`.

11.5.2 rechdf5 (or sachdf5) [optional]

Similar to the `rec` command, the `rechdf5` command is used to save the time history of the solution at a fixed location in space in the HDF5 format, see § 9.2 for examples. The major difference is that all station's data is stored in a single file instead of multiple ones in either SAC or USGS format.

Syntax:
`rechdf5 infile=... outfile=... writeEvery=... downsample=... variables=...`
Required parameters:
Location of the receiver in Cartesian or geographical coordinates stored in HDF5 format.

The HDF5 file format is described in Section 12.9.

sachdf5 command parameters				
Option	**Description**	**Type**	**Units**	**Default**
infile	input file name	string	none	station.h5
outfile	input file name	string	none	station_out.h5
writeEvery	cycle interval to write the file to disk	int	none	1000
variables	displacement, velocity, div, curl, strains, or displacementgradient	string	none	displacement
downsample	outputs all components with the given downsample rate, resulting in less output data size	int	none	1

11.5.3 image [optional]

Syntax:
`image x=... y=... z=... time=... timeInterval=... cycle=... cycleInterval=... file=... mode=... precision=...`
Required parameters:
Location of the image plane (x, y, or z)
Time for output (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 12.10.

image command parameters (part 1)				
Option	**Description**	**Type**	**Units**	**Default**
x	x location of image plane (≥ 0)	real	m	none
y	y location of image plane (≥ 0)	real	m	none
z	z location of image plane (≥ 0)	real	m	none

image command parameters (part 2)				
Option	**Description**	**Type**	**Units**	**Default**
time	Time-level for outputting image (closest time step) (≥ 0)	real	s	none
timeInterval	Time-level interval for outputting a series of images (> 0)	real	s	none
cycle	Time-step cycle to output image (≥ 0)	int	none	none
cycleInterval	Time-step cycle interval to output a series of images (≥ 1)	int	none	none
file	File name header of image	string	none	image
precision	Floating point precision for saving data (float or double)	string	none	float
mode	The field to be saved	string	none	rho

mode can take one of the following values:

mode options (grid, location & topography)	
Value	**Description**
lat	latitude (in degrees)
lon	longitude (in degrees)
topo	elevation of topography [*only available with topography*]
grid	grid coordinates in the plane of visualization (*e.g.* y-z plane if x=const)
gridx	grid x-coordinates in the plane of visualization
gridy	grid y-coordinates in the plane of visualization
gridz	grid z-coordinates in the plane of visualization

mode options (material)	
Value	**Description**
rho	Density
lambda	1st Lamé parameter
mu	2nd Lamé parameter (shear modulus)
p	Compressional wave speed
s	Shear wave speed
qp	Q_P quality factor
qs	Q_S quality factor

mode options (solution)	
Value	**Description**
ux	displacement in the x-direction
uy	displacement in the y-direction
uz	displacement in the z-direction
div	divergence of **u** (displacement)
curl	magnitude of the rotation of **u**
mag	magnitude of **u**
hmag	magnitude of $(u^{(x)}, u^{(y)})$ (horizontal components)
hmax	maximum in time of magnitude of $(u^{(x)}, u^{(y)})$
vmax	maximum in time of $\lvert u^{(z)} \rvert$ (vertical component)
divdudt (veldiv)	divergence of $\mathbf{u}_t$ (velocity)
curldudt (velcurl)	magnitude of the rotation of $\mathbf{u}_t$
magdudt (velmag)	magnitude of the $\mathbf{u}_t$
hmagdudt (hvelmag)	magnitude of $(u_t^{(x)}, u_t^{(y)})$
hmaxdudt (hvelmax)	maximum in time of magnitude of $(u^{(x)}, u^{(y)})$
vmaxdudt (vvelmax)	maximum in time of $\lvert u_t^{(z)} \rvert$

mode options (testing)	
Value	**Description**
uxexact	x-component of exact solution
uyexact	y-component of exact solution
uzexact	z-component of exact solution
uxerr	x-component of error (difference between computed and exact solution)
uyerr	y-component of error (difference between computed and exact solution)
uzerr	z-component of error (difference between computed and exact solution)

11.5.4 imagehdf5 [optional]

Syntax:
`imagehdf5 x=... y=... z=... time=... timeInterval=... cycle=... cycleInterval=... file=... mode=... precision=...`
Required parameters:
Location of the image plane (x, y, or z)
Time for output (time, timeInterval, cycle, or cycleInterval)
Notes:

imagehdf5 is identical to the image command, while writing the image data in an HDF5 file. One can easily read and plot the data using python function provided in the jupyter notebook at `tools/plotimageh5.ipynb`

11.5.5 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 12.12.

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

The `mode` keyword can have the following values:

volimage mode sub-options	
Value	**Description**
ux	x-components of the solution (displacement)
uy	y-components of the solution (displacement)
uz	z-components of the solution (displacement)
rho	density of material
p	C_p material velocity
s	C_s material velocity
lambda	First Lamé parameter
mu	Second Lamé parameter
qp	Q_p material attenuation factor
qs	Q_s material attenuation factor

11.5.6 ssioutput [optional]

Syntax:
`ssioutput file=... dumpInterval=... bufferInterval=... xmin=... xmax=... ymin=... ymax=... depth=... compression_option=...`
Required parameter:
file, xmin, xmax, ymin, ymax, depth.

The `ssioutput` command in *SW4* allows you to save 3D volumetric data in a file for ESSI. These files are in an HDF5 format that can be post-processed for ESSI input directly. SW4 must be compiled with the `hdf5=yes` option to enable this option, and tests are available in the `examples/essi` directory.

Be aware that these files can be *very* large, depending on the x,y,z ranges (which could include the whole domain). The option **file=...** has the same meaning as the corresponding parameters in the image command. The `ssioutput` command produces files with extension `.ssi`. To reduce the output size, we have enabled the support to utilize ZFP `https://github.com/LLNL/zfp` and SZ `https://github.com/szcompressor/SZ` lossy compression. They can reduce the output size without significant precision loss. We found setting `zfp_accuracy=0.01` can reduce the output size by a factor of 40. Note ZFP and H5Z-ZFP must be installed and linked with SW4 to use this option, see the Installation Guide for detailed installation instructions.

The option **dumpInterval=...** affects the maximum number of time steps in a file, it allows down-sampling the data to reduce the overall output size. If omitted, the data is written after every time step of the simulation.

Note that when the `checkpoint` command is being used, it is best to have **cycleInterval** divisible by **dumpInterval**. This is to guarantee that both the checkpoint and SSI files are written after the same time step.

Note that only data from the topmost grid can be output, which may or may not include topography and in the case of mesh refinement, will not span the whole vertical domain. The **depth=...** option will use the horizontal grid spacing to estimate a fixed number of grid points to

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

To imporve the I/O performance, **bufferInterval** should be set. We found setting it 400 to 800 are good in large scale simulations.

The file format is described in Section 12.13.

ssioutput command options				
Option	**Description**	**Type**	**Units**	**Default**
file	file name header of image	string	none	ssioutput
dumpInterval	interval between file writes	string	none	-1
xmin	starting x location of requested SSI region	real	m	0
xmax	ending x location of requested SSI region	real	m	x_{max}
ymin	starting y location of requested SSI region	real	m	0
ymax	ending y location of requested SSI region	real	m	y_{max}
depth	approx depth of output over requested SSI region	real	m	0
zfp_accuracy	use ZFP lossy compression accuracy mode	float	none	N/A
zfp_precision	use ZFP lossy compression precision mode	float	none	N/A
zfp_rate	use ZFP lossy compression rate mode	float	none	N/A
zfp_reversible	use ZFP lossless compression mode	int	none	0

11.5.7 gmt [optional]

Syntax:
`gmt file=...`
Required parameters:
None.

gmt command parameters			
Option	**Description**	**Type**	**Default**
file	name of output file for gmt c-shell commands	string	sw4.gmt.csh

11.5.8 sfileoutput

Syntax:
`sfileoutput file=... sampleFactor=... sampleFactorH=... sampleFactorV=...`
Required parameter:
File name prefix: (file).

The `sfileoutput` command in *SW4* allows you to save 3D volumetric material data on a HDF5 file (sfile). These files can be read by the python/Jupyter script in `tools/plot_sfile.ipynb`. Be aware that these files can be *very* large if `sampleFactor` is not set properly. The output is written out at the end of SW4 run. The **sampleFactor=...** option sets the sub-sampling factor to the output, a `sampleFactor=n` will result in approximately $1/n^3$ of the file size from `sampleFactor=1`. Users can also set different sample factor in horizontal (`sampleFactorH`) and vertical directions (`sampleFactorV`). As the material data is often very sensitive on the vertical direction, it is advisable to set `sampleFactorV` to 1 and `sampleFactorH` to 2 or a larger value. The `sfileoutput` command produces files with extension `.sfile`.

The file format is described in Section 12.6.

sfileoutput command options				
Option	**Description**	**Type**	**Units**	**Default**
file	file name of sfile	string	none	"sfileoutput"
sampleFactor	specifies the sub-sampling factor for both horizontal and vertical directions	int	none	1
sampleFactorH	specifies the sub-sampling factor in the horizontal direction	int	none	1
sampleFactorV	specifies the sub-sampling factor in the vertical direction	int	none	1

11.5.9 checkpoint [optional]

Syntax:

```
checkpoint cycleInterval=...  file=...  restartpath=...
zfp_accuracy/precision/rate/reversible=...
checkpoint cycleInterval=...  file=...  restartpath=...  restartfile=...
zfp_accuracy/precision/rate/reversible=...
```

Required parameter:

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

Notes:

`restartpath`. argument sets the path for the checkpoint (write) and restart files

The `checkpoint` command in *SW4* allows you to save and restore the simulation using restart files. These files can be created using the input:

```
checkpoint cycleInterval=50 restartpath=loh1-results-restart file=LOH1_restart
```

The command above will create a checkpoint file in `restartpath`:

```
loh1-results-restart/LOH1_restart.cycle=050.sw4checkpoint
```

where the file name format is `[path]/[prefix].cycle=[XXX].sw4checkpoint`, and the format of the cycle number depends on the total number of time steps in the simulation. The checkpoint functions keep only the last 2 checkpoint files, and deletes all prior files. To assure continuity of the

time series data for the `rec` command, `checkpoint` forces those files to be written whenever the checkpoint file is written. These are written to the directory specified by the `fileio path` variable.

The simulation can be restarted with the input line:

```
checkpoint cycleInterval=5 restartfile=LOH1_restart.cycle=050.sw4checkpoint
    restartpath=loh1-results-restart file=LOH1_restart
```

On restart, the simulation will read in the state, and attempt to read in all the `rec` USGS and SAC files specified in the input from the `fileio path` directory. It will abort if it cannot find the checkpoint or `rec` files to read from the restart path. The `rec` USGS and SAC files are read in, and continued from the checkpoint cycle, and *overwritten* in the output directory `fileio path`. **Note:** Because SAC files are float (4 byte) instead of double (8 byte), there may be differences in the last (7th or higher) digits after restart.

restart parameters			
Option	**Description**	**Type**	**Default**
file	file header name to be prepended on restart files.	string	" "
cycleInterval	sets the interval for writing out restart files	int	0
restartfile	restart the code from this file	string	"restart"
restartpath	path to restart file	string	N/A
hdf5	use HDF5 output format	string	"no"
zfp_accuracy	use ZFP lossy compression accuracy mode	float	N/A
zfp_precision	use ZFP lossy compression precision mode	float	N/A
zfp_rate	use ZFP lossy compression rate mode	float	N/A
zfp_reversible	use ZFP lossless compression mode	int	0

11.6 SW4 testing commands [optional]

11.6.1 twilight

The **twilight** command runs *SW4* 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			
Option	**Description**	**Type**	**Default**
errorlog	Outputs error log in file twilight_errors.dat	int	0
omega	Wave number in exact solution	real	1.0
c	Phase speed in exact solution	real	1.3
phase	Solution phase coefficient	real	0.0
momega	Wave number in material	real	1.0
mphase	Material phase coefficient	real	0.4
amprho	Density amplitude	real	1.0
ampmu	Material μ amplitude	real	1.0
amplambda	Material λ amplitude	real	1.0

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			
Option	**Description**	**Type**	**Default**
x	x-coordinate of point source	real	0.0
y	y-coordinate of point source	real	0.0
cp	P-wave velocity	real	1.0
rho	Density	real	1.0
fz	Magnitude of the forcing	real	1.0

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=... directest=...`
Required parameters:
None

testpointsource command parameters			
Option	**Description**	**Type**	**Default**
cp	P-wave velocity	real	$\sqrt{3}$
cs	S-wave velocity	real	1
rho	Density	real	1
directest	Test moment conditions (0 or 1)	int	0

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

testrayleigh command parameters			
Option	**Description**	**Type**	**Default**
cp	P-wave velocity	real	$\sqrt{3}$
cs	S-wave velocity	real	1
rho	Density	real	1
nwl	number of wave lengths in domain	int	1

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 cpcsratio=... seed=... writeEvery=... filename=...`
Required parameters:
None

testenergy command parameters			
Option	**Description**	**Type**	**Default**
cpcsratio	P-wave velocity to S-wave velocity ratio	real	$\sqrt{3}$
seed	Pseudo-random number generator seed	int	2934839
writeEvery	frequency for saving log file to disk	int	1000
filename	name of log file	string	energy.log

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.

Boundary conditions parameters			
Option	**Description**	**Value**	**Default**
lx	Boundary condition at $x = 0$	int 0-3	2
hx	Boundary condition at $x = x_{max}$	int 0-3	2
ly	Boundary condition at $y = 0$	int 0-3	2
hy	Boundary condition at $y = y_{max}$	int 0-3	2
lz	Boundary condition at $depth = 0$	int 0-3	0
hz	Boundary condition at $z = z_{max}$	int 0-3	2

boundary condition values	
Value	**Type**
0	Stress-free boundary
1	Dirichlet boundary
2	Supergrid boundary
3	Periodic boundary

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 erros resulting in NaN (Not a Number). The check is performed after each time step.

developer parameters			
Option	**Description**	**Type**	**Default**
cfl	CFL number (> 0)	real	1.3
checkfornan	Scan solution arrays for NaN	string	off

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 (δ_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:

Line	Column 1	Column 2	Column 3
1	t_0 (real)	δ_t (real)	N_d (integer)
2	g_1 (real)		
3	g_2 (real)		
$\vdots$	$\vdots$		
$N_d + 1$	g_{N_d} (real)		

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:

$$N_{lon} \quad N_{lat}$$

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

$$\begin{array}{ccc}
\phi_1 & \theta_1 & e_{1,1} \\
\phi_2 & \theta_1 & e_{2,1} \\
\vdots & \vdots & \vdots \\
\phi_{Nlon} & \theta_1 & e_{Nlon,1} \\
\vdots & \vdots & \vdots \\
\phi_1 & \theta_{Nlat} & e_{1,Nlat} \\
\phi_2 & \theta_{Nlat} & e_{2,Nlat} \\
\vdots & \vdots & \vdots \\
\phi_{Nlon} & \theta_{Nlat} & e_{Nlon,Nlat}
\end{array}$$

12.2.2 Topography on a Cartesian lattice

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

$$x_i, \quad i = 1, 2, \ldots, Nx,$$

and y-coordinates

$$y_j, \quad j = 1, 2, \ldots, Ny,$$

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

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

where x_{max} and y_{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:

$$\begin{array}{ccc} x_1 & y_1 & e_{1,1} \\ x_2 & y_1 & e_{2,1} \\ \vdots & \vdots & \vdots \\ x_{Nx} & y_1 & e_{Nx,1} \\ \vdots & \vdots & \vdots \\ x_1 & y_{Ny} & e_{1,Ny} \\ x_2 & y_{Ny} & e_{2,Ny} \\ \vdots & \vdots & \vdots \\ x_{Nx} & y_{Ny} & e_{Nx,Ny} \end{array}$$

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 formated.

12.3.1 pfile on a geographic lattice

The header has 7 lines and follows the following format:

Line	Column 1	Column 2	Column 3	Column 4
1	Name (string)			
2	Δ [deg] (real)			
3	N_{lat} (integer)	Lat_{min} [deg] (real)	Lat_{max} [deg] (real)	
4	N_{lon} (integer)	Lon_{min} [deg] (real)	Lon_{max} [deg] (real)	
5	N_{dep} (integer)	d_{min} [km] (real)	d_{max} [km] (real)	
6	I_{sed} (integer)	I_{MoHo} (integer)	I_{410} (integer)	I_{660} (integer)
7	Q-available? (logical)			

The first line holds the optional name of the material model. Line 2 contains the parameter Δ, 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

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:

Index (int)	depth [km]	C_p [km/s]	C_s [km/s]	ρ [g/cm^3]	Q_P	Q_S

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 (ρ) 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:

Line	Column 1	Column 2	Column 3	Column 4
1	Name (string)			
2	h [m] (real)			
3	N_x (integer)	x_{min} [m] (real)	x_{max} [m] (real)	
4	N_y (integer)	y_{min} [m] (real)	y_{max} [m] (real)	
5	N_{dep} (integer)	d_{min} [m] (real)	d_{max} [m] (real)	
6	I_{sed} (integer)	I_{MoHo} (integer)	I_{410} (integer)	I_{660} (integer)
7	Q-available? (logical)			

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 (`[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:

```
for (y = y_min; y <= y_max; y+ = h)
      for (x = x_min; x <= x_max; x+ = h)
            (save depth profile for x, y)
      end
end
```

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 $x = 100.4\,m$ and $y = 30.6\,m$, 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:

Index (int)	depth [m]	C_p [m/s]	C_s [m/s]	ρ [kg/m^3]	Q_P	Q_S

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 pfiles 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:

$$N_{lon} \quad N_{lat} \quad N_{mat}$$

On subsequent lines, longitude, latitude and N_{mat} surface depth values are given in column first ordering:

$$\begin{array}{ccccc}
Lon_1 & Lat_1 & d_{1,1,1} & \dots & d_{N_{mat},1,1} \\
Lon_2 & Lat_1 & d_{1,2,1} & \dots & d_{N_{mat},2,1} \\
\vdots & \vdots & \vdots & & \vdots \\
Lon_{N_{lon}} & Lat_1 & d_{1,N_{lon},1} & \dots & d_{N_{mat},N_{lon},1} \\
\vdots & \vdots & \vdots & & \\
Lon_1 & Lat_{N_{lat}} & d_{1,1,N_{lat}} & \dots & d_{N_{mat},1,N_{lat}} \\
Lon_2 & Lat_{N_{lat}} & d_{1,2,N_{lat}} & \dots & d_{N_{mat},2,N_{lat}} \\
\vdots & \vdots & \vdots & & \vdots \\
Lon_{N_{lon}} & Lat_{N_{lat}} & d_{1,N_{lon},N_{lat}} & \dots & d_{N_{mat},N_{lon},N_{lat}}
\end{array}$$

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:

$$N_x \quad N_y \quad N_{mat}$$

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):

Offset (bytes)	Name	Type	Bytes
0	magic	int	4
4	prec	int	4
8	att	int	4
12	az	double	8
20	lon0	double	8
28	lat0	double	8
36	mlen	int	4
40	mercstr	char	mlen
40+mlen	N_b	int	4

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:

`for` ($b = 1$; $b \leq N_b$; b`++`)

Offset (bytes)	Name	Type	Bytes
$o_0 + 40(b-1)$	hh_b	double	8
$o_0 + 8 + 40(b-1)$	hv_b	double	8
$o_0 + 16 + 40(b-1)$	$z0_b$	double	8
$o_0 + 24 + 40(b-1)$	nc_b	int	4
$o_0 + 28 + 40(b-1)$	ni_b	int	4
$o_0 + 32 + 40(b-1)$	nj_b	int	4
$o_0 + 36 + 40(b-1)$	nk_b	int	4

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 it 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

```
for (b = 1; b ≤ Nb; b++)
 for (i = 1; i ≤ nib; i++)
  for (j = 1; j ≤ njb; j++)
   for (k = 1; k ≤ nkb; k++)
    for (c = 1; c ≤ ncb; c++)
        read ab(c, i, j, k)
```

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,

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

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 `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 ρ. 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 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 $nc_1 = 1$. The following blocks must have either 3 or 5 components, i.e.,

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

- Because the topography/bathymetry is a function of the horizontal coordinates, the first block must have $nk_1 = 1$.
- 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,

$$z0_b + (nk_b - 1)hv_b = z0_{b+1}, \quad b = 2, 3, \ldots, N_b - 1.$$

  However, neither $z0_1$ 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 sfile

The `sfile` command provides material data and topography, similar to the `rfile`, but with materials defined on a curvilinear mesh with refinement boundaries. Each grid of material data has corresponding top and bottom interfaces that defines the vertical distribution of grids points, which is constant grid spacing across *nz* total points (including end points). This layout, along with the coarsest horizontal grid spacing, *hh*, defines the (*x,y,z*) point location relative to the origin (*lon,lat,azim*) for every point.

The `sfile` files are self-describing in HDF5 format, with the following fields:

Name	Type	Size
"Origin longitude, latitude, azimuth"	H5T_IEEE_F64LE	3
"Coarsest horizontal grid spacing"	H5T_IEEE_F64LE	1
"Attenuation"	H5T_STD_I32LE	1
"ngrids"	H5T_STD_I32LE	1
"Min, max depth"	H5T_IEEE_F64LE	2
Group /Z_interfaces		
"z_interface_{0...ngrids}"	H5T_IEEE_F32LE	*(nx,ny)[ng]*
Group /Material_model		
"grid_{0...ngrids-1}/{mat}"	H5T_IEEE_F32LE	*(nx,ny,nz)[ng]*

The output of `h5dump -A` header and metadata is below (lightly edited):

```
HDF5 "berkeley.sfile" {
GROUP "/" {
   ATTRIBUTE "Attenuation" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SCALAR
      DATA {
      (0): 1
      }
   }
   ATTRIBUTE "Coarsest horizontal grid spacing" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SCALAR
      DATA {
      (0): 100
      }
   }
   ATTRIBUTE "Min, max depth" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
      DATA {
      (0): -557.6, 6387.5
```

```
   }
}
ATTRIBUTE "Origin longitude, latitude, azimuth" {
   DATATYPE  H5T_IEEE_F64LE
   DATASPACE  SIMPLE { ( 3 ) / ( 3 ) }
   DATA {
   (0): -122.25, 37.93, 143.638
   }
}
ATTRIBUTE "ngrids" {
   DATATYPE  H5T_STD_I32LE
   DATASPACE  SCALAR
   DATA {
   (0): 3
   }
}
GROUP "Material_model" {
   GROUP "grid_0" { ... }
   GROUP "grid_1" { ... }
   GROUP "grid_2" {
      ATTRIBUTE "Horizontal grid size" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SCALAR
         DATA {
         (0): 400
         }
      }
      ATTRIBUTE "Number of components" {
         DATATYPE  H5T_STD_I32LE
         DATASPACE  SCALAR
         DATA {
         (0): 5
         }
      }
      DATASET "Cp" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 31, 31, 34 ) / ( 31, 31, 34 ) }
      }
      DATASET "Cs" { ... }
      DATASET "Qp" { ... }
      DATASET "Qs" { ... }
      DATASET "Rho" { ... }
   }
}
GROUP "Z_interfaces" {
   DATASET "z_values_0" {
```

```
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 121, 121 ) / ( 121, 121 ) }
      }
      DATASET "z_values_1" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 121, 121 ) / ( 121, 121 ) }
      }
      DATASET "z_values_2" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 61, 61 ) / ( 61, 61 ) }
      }
      DATASET "z_values_3" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 31, 31 ) / ( 31, 31 ) }
      }
   }
}
}
```

12.7 SRF-HDF5

Based on the SRF (Standard Rupture Format), we have created the SRF-HDF5 format, which stores the same amount of information in a more compact HDF5 file that is smaller in size and can be read faster.

To convert an existing SRF file into HDF5 format, we have provided a python script under `tools/srf2hdf5.py` in the sw4 repository. The script requires two command line parameters to specify the input SRF file and the output SRF-HDF5 file, e.g. `python srf2hdf5.py rupture.srf rupture.h5`.

The output of `h5dump -A` header and metadata is below, the "PLANE" and "VERSION" attributes store the header block of the SRF file, with the number of plane segments implicitly implied in the dataspace of the "PLANE" attribute.

The "POINTS" and "SR1" datasets store the data block of the SRF file. Different from the SRF format, where the slip rate at each time step for a direction follows immediately after each point source, we concatenate all slip rates and put them in a single 1D dataset, which is optimized for data read. To access the slip rates of a particular point source, one needs to accumulate the number of rates before the desired one and calculate its offset in the "SR1" dataset.

Only "SR1" is converted in the SRF-HDF5 format as others (SR2 and SR3) are ignored by sw4.

```
HDF5 "m6.5-20.0x13.0.s500.v5.1.srf.h5" {
GROUP "/" {
   ATTRIBUTE "PLANE" {
      DATATYPE  H5T_COMPOUND {
         H5T_IEEE_F32LE "ELON";
         H5T_IEEE_F32LE "ELAT";
         H5T_STD_I32LE "NSTK";
         H5T_STD_I32LE "NDIP";
```

```
      H5T_IEEE_F32LE "LEN";
      H5T_IEEE_F32LE "WID";
      H5T_IEEE_F32LE "STK";
      H5T_IEEE_F32LE "DIP";
      H5T_IEEE_F32LE "DTOP";
      H5T_IEEE_F32LE "SHYP";
      H5T_IEEE_F32LE "DHYP";
   }
   DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
   DATA {
   (0): {
         -122.218,
         37.8433,
         100,
         65,
         20,
         13,
         144,
         90,
         3,
         -5,
         8
      }
   }
}
ATTRIBUTE "VERSION" {
   DATATYPE  H5T_IEEE_F32LE
   DATASPACE  SCALAR
   DATA {
   (0): 2
   }
}
DATASET "POINTS" {
   DATATYPE  H5T_COMPOUND {
      H5T_IEEE_F32LE "LON";
      H5T_IEEE_F32LE "LAT";
      H5T_IEEE_F32LE "DEP";
      H5T_IEEE_F32LE "STK";
      H5T_IEEE_F32LE "DIP";
      H5T_IEEE_F32LE "AREA";
      H5T_IEEE_F32LE "TINIT";
      H5T_IEEE_F32LE "DT";
      H5T_IEEE_F32LE "VS";
      H5T_IEEE_F32LE "DEN";
      H5T_IEEE_F32LE "RAKE";
      H5T_IEEE_F32LE "SLIP1";
```

```
         H5T_STD_I32LE "NT1";
         H5T_IEEE_F32LE "SLIP2";
         H5T_STD_I32LE "NT2";
         H5T_IEEE_F32LE "SLIP3";
         H5T_STD_I32LE "NT3";
      }
      DATASPACE  SIMPLE { ( 6500 ) / ( 6500 ) }
   }
   DATASET "SR1" {
      DATATYPE  H5T_IEEE_F32LE
      DATASPACE  SIMPLE { ( 144616 ) / ( 144616 ) }
   }
}
}
```

12.8 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`, 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.

12.9 sachdf5

The HDF5 format holds the time history of all components of the solution at a fixed point in space. There are 4 global attributes in the root group of the file, including `DATETIME`, a time string (in UTC) recording the start of the simulation; `UNIT`, the unit for the data ("m", "m/s", etc.); `DELTA`, the sample interval; and `ORIGINTIME`, the origin time in seconds, relative to the start time of SW4 calculation and seismogram of the earliest source.

Then the time-series data of each station in separate groups, using the station name as the group name. Inside each group, there is the number of valid points stored in the data array, the station location in both xyz coordiate in SW4 domain (m) and latitude, longitude, depth. In the case when the surface grids are too sparse, we also store the actual location and the distance between the actual and user-specified location in `ACTUALSTLA,STLO,STDP`, `ACTUALSTX,STY,STZ`, and `DISTFROMACTUAL`. The data array may be named with `X, Y, Z, or EW, NS, UP` depending on the `isnsew=` option specified in the rec/sac command. For each of these component, there are also `*CMPAZ and *CMINC` (e.g. `XCMPAZ, XCMPINC`) recording its azimuth and inclination.

Note that the dataspace of the data array is pre-allocated to hold the entire timeseries data of the simulation, when the SW4 is terminate early (i.e. in case of checkpoint/restart), only a portion of the data in the array is valid, with the size recorded in the `NPTS` field. When reading the data, it is recommended to read the `NPTS` value first and only use the first `NPTS` values from the data array.

The HDF5 file stores data in an self-describing way, with the following fields:

Name	Type	Size
DATETIME	H5T_STRING	string_len
UNIT	H5T_STRING	string_len
DELTA	H5T_STD_F32LE	1
ORIGINTIME	H5T_STD_F32LE	1
Group /BK.BDM (station name)		
ISNSEW	H5T_IEEE_I32LE	1
LOC	H5T_IEEE_I32LE	1
NPTS	H5T_IEEE_I32LE	1
STLA,STLO,STDP	H5T_STD_F32LE	3
ACTUALSTLA,STLO,STDP	H5T_STD_F32LE	3
STX,STY,STZ	H5T_STD_F32LE	3
ACTUALSTX,STY,STZ	H5T_STD_F32LE	3
DISTFROMACTUAL	H5T_STD_F32LE	1
X or EW	H5T_STD_F32LE	NPTS
XCMPAZ or EWCMPAZ	H5T_STD_F32LE	1
XCMPINC or EWCMPINC	H5T_STD_F32LE	1
Y or NS	H5T_STD_F32LE	NPTS
YCMPAZ or NSCMPAZ	H5T_STD_F32LE	1
YCMPINC or NSCMPINC	H5T_STD_F32LE	1
Z or UP	H5T_STD_F32LE	NPTS
ZCMPAZ or UPCMPAZ	H5T_STD_F32LE	1
ZCMPINC or UPCMPINC	H5T_STD_F32LE	1

The output of `h5dump -A` header and metadata is below (lightly edited):

```
HDF5 "sta.hdf5" {
GROUP "/" {
   ATTRIBUTE "DATETIME" {
      DATATYPE  H5T_STRING {
         STRSIZE 27;
         STRPAD H5T_STR_NULLTERM;
         CSET H5T_CSET_ASCII;
         CTYPE H5T_C_S1;
      }
      DATASPACE  SCALAR
      DATA {
      (0): "2019-10-02T02:18:58.000000"
      }
```

```
}
ATTRIBUTE "UNIT" {
   DATATYPE  H5T_STRING {
      STRSIZE 2;
      STRPAD H5T_STR_NULLTERM;
      CSET H5T_CSET_ASCII;
      CTYPE H5T_C_S1;
   }
   DATASPACE  SCALAR
   DATA {
   (0): "m"
   }
}
DATASET "DELTA" {
   DATATYPE  H5T_IEEE_F32LE
   DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
}
DATASET "ORIGINTIME" {
   DATATYPE  H5T_IEEE_F32LE
   DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
}
GROUP "BK.BDM" {
   DATASET "LOC" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
   }
   DATASET "NPTS" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
   }
   DATASET "STLA,STLO,STDP" {
      DATATYPE  H5T_IEEE_F32LE
      DATASPACE  SIMPLE { ( 3 ) / ( 3 ) }
   }
   DATASET "STX,STY,STZ" {
      DATATYPE  H5T_IEEE_F32LE
      DATASPACE  SIMPLE { ( 3 ) / ( 3 ) }
   }
   DATASET "X" {
      DATATYPE  H5T_IEEE_F32LE
      DATASPACE  SIMPLE { ( 2209 ) / ( 2209 ) }
   }
   DATASET "XCMPAZ" {
      DATATYPE  H5T_IEEE_F32LE
      DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
   }
```

```
      DATASET "XCMPINC" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
      }
      DATASET "Y" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 2209 ) / ( 2209 ) }
      }
      DATASET "YCMPAZ" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
      }
      DATASET "YCMPINC" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
      }
      DATASET "Z" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 2209 ) / ( 2209 ) }
      }
      DATASET "ZCMPAZ" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
      }
      DATASET "ZCMPINC" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
      }
   }
   GROUP "BK.BKS" {
   ...
   }
}
```

12.10 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:

Offset (bytes)	Name	Type	Bytes
0	prec	int	4
4	N_p	int	4
8	t	double	8
16	plane	int	4
20	ξ	double	8
28	mode	int	4
32	grdinfo	int	4
36	creation-time	char[25]	25

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 $t \geq 0$ 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 $x = \xi$. The type of data is saved in `mode`. The following 32 different types of data can be saved on an image file,

`mode`	1	2	3	4	5	6	7	8	9	10	11	12	13
Data	$u^{(x)}$	$u^{(y)}$	$u^{(z)}$	ρ	λ	μ	C_p	C_s	$u_{ex}^{(x)}$	$u_{ex}^{(y)}$	$u_{ez}^{(z)}$	$\text{div}(\mathbf{u})$	$\lvert\text{curl}(\mathbf{u})\rvert$

`mode`	14	15	16	17	18	19	20	21	22	23	24
Data	$\text{div}(\mathbf{u}_t)$	$\lvert\text{curl}(\mathbf{u}_t)\rvert$	lat	lon	topo	x	y	z	$u_{error}^{(x)}$	$u_{error}^{(y)}$	$u_{error}^{(z)}$

`mode`	25	26	27	28
Data	$\lvert\mathbf{u}_t\rvert$	$\sqrt{(u_t^{(x)})^2 + (u_t^{(y)})^2}$	$\max_t \sqrt{(u_t^{(x)})^2 + (u_t^{(y)})^2}$	$\max_t \left\lvert u_t^{(z)} \right\rvert$

`mode`	29	30	31	32
Data	$\lvert\mathbf{u}\rvert$	$\sqrt{(u^{(x)})^2 + (u^{(y)})^2}$	$\max_t \sqrt{(u^{(x)})^2 + (u^{(y)})^2}$	$\max_t \left\lvert u^{(z)} \right\rvert$

Here, $|\mathbf{a}|$ denotes the magnitude of the vector $\mathbf{a} \in \Re^3$ and $\mathbf{u}_{ex}$ denotes the exact solution, which also is needed to calculate the error in the solution, $\mathbf{u}_{error}$. 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:

`for` ($p = 1$; $p \le N_p$; p`++`)

Offset (bytes)	Name	Type	Bytes
$61 + 32(p-1)$	h_p	double	8
$69 + 32(p-1)$	$zmin_p$	double	8
$77 + 32(p-1)$	ib_p	int	4
$81 + 32(p-1)$	ni_p	int	4
$85 + 32(p-1)$	jb_p	int	4
$89 + 32(p-1)$	nj_p	int	4

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,

$$\texttt{plane=0:}\quad \begin{cases} x^{(p)}_{i,j} = \xi, & \\ y^{(p)}_{i,j} = h_p(i-1), & ib_p \le i \le ib_p + ni_p - 1, \\ z^{(p)}_{i,j} = zmin_p + h_p(j-1), & jb_p \le j \le jb_p + nj_p - 1, \end{cases}$$

$$\texttt{plane=1:}\quad \begin{cases} x^{(p)}_{i,j} = h_p(i-1), & ib_p \le i \le ib_p + ni_p - 1, \\ y^{(p)}_{i,j} = \xi, & \\ z^{(p)}_{i,j} = zmin_p + h_p(j-1), & jb_p \le j \le jb_p + nj_p - 1, \end{cases}$$

$$\texttt{plane=2:}\quad \begin{cases} x^{(p)}_{i,j} = h_p(i-1), & ib_p \le i \le ib_p + ni_p - 1, \\ y^{(p)}_{i,j} = h_p(j-1), & jb_p \le j \le jb_p + nj_p - 1, \\ z^{(p)}_{i,j} = \xi. & \end{cases}$$

The header is followed by grid function data on each of the N_p patches. The data is saved as floats (`prec`=4), or doubles (`prec`=8). Each patch of the data is stored as in a two-dimensional array, $u^{(p)}(i,j)$, where $ib_p \le i \le ib_p + ni_p - 1$ and $jb_p \le j \le jb_p + nj_p - 1$. The data can be read as outlined by the following pseudo code:

```
for (p = 1; p ≤ Np; p++){
  for (j = 1; j ≤ njp; j++){
    for (i = 1; i ≤ nip; i++){
      read u^(p)(i − 1 + ibp, j − 1 + jbp)
    }
  }
}
```

If `grdinfo`=0, there is no additional information on the image file.

If `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 `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:

$p = N_p$;
for ($j = 1$; $j \le nj_p$; j++){
  for ($i = 1$; $i \le ni_p$; i++){
    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 `grdinfo`=1, the $z_{i,j}^{(N_p)}$ coordinates replace the above definition of the grid for the cases `plane`=0 and `plane`=1. For `plane`=2, the grid is defined by the (x, y) coordinates, which currently remains Cartesian also for a curvilinear grid.

12.11 imagehdf5

The image files in HDF5 format hold identical data as in the image file, but are stored as HDF5 attributes and datasets. To improve the I/O efficiency, the header data and the patch data are stored in arrays instead of individually in the binary format (i.e. all the `ni` values of multiple patches are stored in the same HDF5 dataset, with its length equal to the number of patches).

The output of `h5dump -A imagehdf5.file` is below:

```
HDF5 "image.cycle=0.y=40000.p.sw4img.h5" {
GROUP "/" {
   ATTRIBUTE "creationtime" {
      DATATYPE  H5T_STRING {
         STRSIZE 25;
         STRPAD H5T_STR_NULLTERM;
         CSET H5T_CSET_ASCII;
         CTYPE H5T_C_S1;
      }
      DATASPACE  SCALAR
      DATA {
      (0): "Fri Jan 24 14:15:23 2020"
      }
   }
   DATASET "coordinate" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
   }
   DATASET "grid" {
      DATATYPE  H5T_IEEE_F32LE
      DATASPACE  SIMPLE { ( 31031 ) / ( 31031 ) }
   }
   DATASET "grid_size" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
   }
```

```
   DATASET "gridinfo" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
   }
   DATASET "mode" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
   }
   DATASET "ni" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
   }
   DATASET "nj" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
   }
   DATASET "npatch" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
   }
   DATASET "patches" {
      DATATYPE  H5T_IEEE_F32LE
      DATASPACE  SIMPLE { ( 307307 ) / ( 307307 ) }
   }
   DATASET "plane" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
   }
   DATASET "time" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
   }
   DATASET "zmin" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
   }
}
}
```

12.12 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.,

Offset (bytes)	Name	Type	Bytes
0	prec	int	4
4	N_p	int	4
8	t	double	8
16	plane	int	4
20	ξ	double	8
28	mode	int	4
32	grdinfo	int	4
36	creation-time	char[25]	25

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:

`mode`	1	2	3	4	5	6	7	8	14	15
Data	$u^{(x)}$	$u^{(y)}$	$u^{(z)}$	ρ	λ	μ	C_p	C_s	Q_p	Q_s

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

`for` ($p = 1$; $p \leq N_p$; 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

```
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 `grdinfo`=0, there is no additional information on the `volimage` file. If `grdinfo`=1, the z-coordinates of the curvilinear grid are also saved on the `volimage` file (as floats or doubles depending on `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;
for (k = 1; k ≤ nk_p; k++){
   for (j = 1; j ≤ nj_p; j++){
      for (i = 1; i ≤ ni_p; i++){
         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 $zmin$ variable.

12.13 ssioutput

The `ssiouput` command generates (potentially very large) binary HDF5 files holding three-dimensional volumetric data for the ESSI application.

The format of the `ssioutput` files is self-describing in HDF5 (use `h5dump -A` to see header metadata):

Name	Type	Size
“ESSI xyz grid spacing”	H5T_IEEE_F64LE	1
“ESSI xyz origin”	H5T_IEEE_F64LE	3
“Grid azimuth”	H5T_IEEE_F64LE	1
“Grid lon-lat origin”	H5T_IEEE_F64LE	2
“cycle start, end”	H5T_STD_I32LE	2
“time start”	H5T_IEEE_F64LE	1
“timestep”	H5T_IEEE_F64LE	1
“vel_{0,1,2} ijk layout”	H5T_IEEE_F64LE	cycle X 3D size
“z coordinates”	H5T_IEEE_F64LE	(optional) 3D size

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
```

```
-m MPITASKS, --mpitasks MPITASKS
                      number of mpi tasks
-d SW4_EXE_DIR, --sw4_exe_dir SW4_EXE_DIR
                      name of directory for sw4 executable
```

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

```
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

$$\begin{aligned}
\rho(x,y,z) &= A_\rho \left(2+\sin(\omega_m x+\theta_m)\cos(\omega_m y+\theta_m)\sin(\omega_m z+\theta_m)\right), \\
\mu(x,y,z) &= A_\mu \left(3+\cos(\omega_m x+\theta_m)\sin(\omega_m y+\theta_m)\sin(\omega_m z+\theta_m)\right), \\
\lambda(x,y,z) &= A_\lambda \left(2+\sin(\omega_m x+\theta_m)\sin(\omega_m y+\theta_m)\cos(\omega_m z+\theta_m)\right).
\end{aligned}$$

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

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

The values of the material parameters (ω_m, θ_m, A_ρ, A_λ, A_μ) and the solution parameters (ω, θ, c_e), can be modified in the input script. Since the exact solution is know, 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.

input file	N_x	h	$e_h = \|\mathbf{u} - \mathbf{u}_{ex}\|_\infty$	ratio_∞	rate_∞
flat-twi-1.in	31	$3.333 \cdot 10^{-2}$	$6.85 \cdot 10^{-4}$	–	–
flat-twi-2.in	61	$1.667 \cdot 10^{-2}$	$3.99 \cdot 10^{-5}$	17.16	4.10
flat-twi-3.in	121	$8.333 \cdot 10^{-3}$	$2.01 \cdot 10^{-6}$	19.85	4.31
flat-twi-4.in	241	$4.167 \cdot 10^{-3}$	$1.40 \cdot 10^{-7}$	14.36	3.84

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 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.

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 τ satisfies $-0.05 \leq \tau \leq 0$. The keyword `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.

input file	N_x	h	$e_h = \|\mathbf{u} - \mathbf{u}_{ex}\|_\infty$	ratio_∞	rate_∞
gauss-twi-1.in	31	$3.333 \cdot 10^{-2}$	$1.74 \cdot 10^{-3}$	–	–
gauss-twi-2.in	61	$1.667 \cdot 10^{-2}$	$1.93 \cdot 10^{-4}$	9.01	3.17
gauss-twi-3.in	121	$8.333 \cdot 10^{-3}$	$9.11 \cdot 10^{-6}$	21.1	4.40
gauss-twi-4.in	241	$4.167 \cdot 10^{-3}$	$5.83 \cdot 10^{-7}$	15.6	3.96

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 $\log_2(e_{2h}/e_h)$.

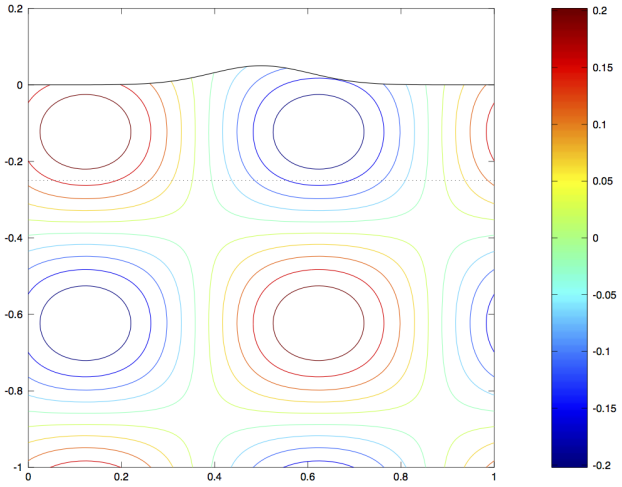


Figure A.1: Contour plot of the $u^{(x)}$ component of the solution along the plane $x = 0.5$. The image file was written with the script `gauss-twi-3.in`. The solid black line shows the shape of the top surface. The dashed black line indicates the interface between the curvilinear and Cartesian grids.

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`,

```
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., $2 \le (x, y) \le 10$, 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 $\mathcal{O}(h^4)$ 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 $t \ge t_1$, 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.

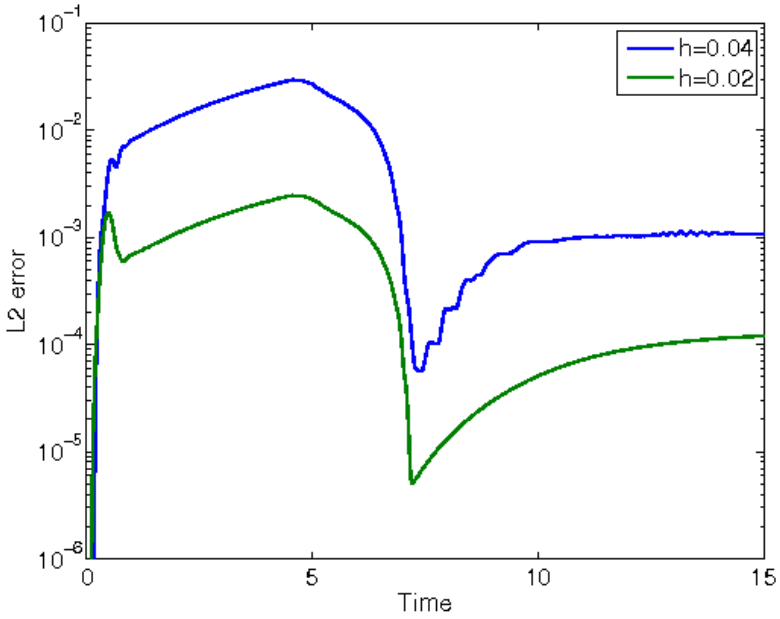


Figure A.2: The L_2 norm of the error in the solution of Lamb's problem, as function of time. The blue and green lines correspond to $h = 0.04$ and $h = 0.02$, respectively.

The source properties are specified by exactly one regular `source` command. In this case the time function is a `C6SmoothBump`. The coarsest grid is setup in the file `ps-dir1.in`,

```
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
```

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.

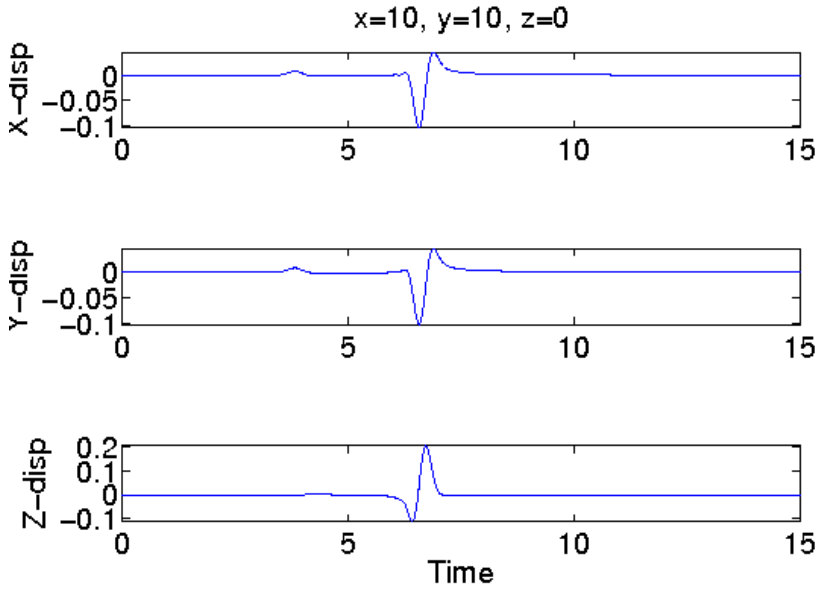


Figure A.3: The x, y, and z-components of the solution of Lamb's problem at $(x, y, z) = (10, 10, 0)$.

input file	h	$e_h = \|\mathbf{u} - \mathbf{u}_{ex}\|_\infty$	ratio_∞	rate_∞
ps-dir1.in	0.04	$2.09 \cdot 10^{-3}$	–	–
ps-dir2.in	0.02	$1.41 \cdot 10^{-4}$	14.82	3.88
ps-dir3.in	0.01	$8.94 \cdot 10^{-6}$	15.77	3.97

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 ≈ 16 each time the grid size is halved.

We remark that the `testpointsource` 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`

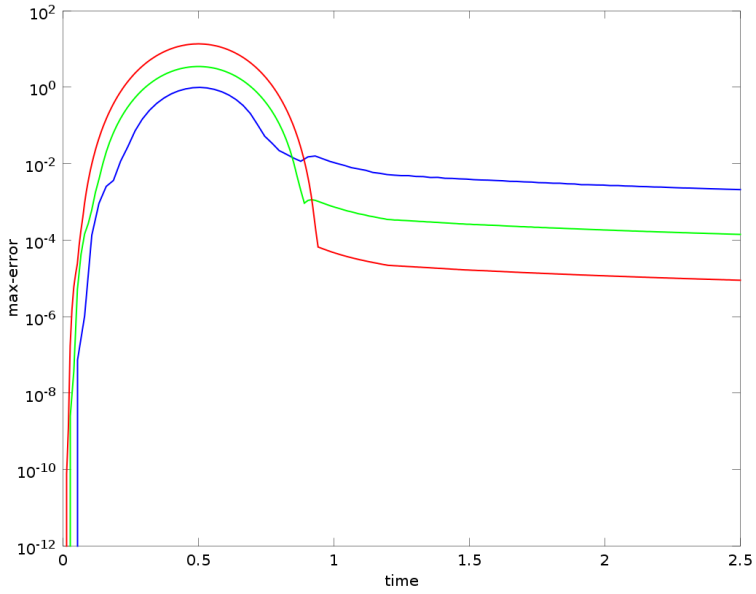


Figure A.4: The max norm of the error in the point source test. The blue, green, and red lines correspond to $h = 0.04$, $h = 0.02$, and $h = 0.01$, respectively.

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 enouraged to study these results, experiment with the duration of the simulation, and the width of the super grid layers.

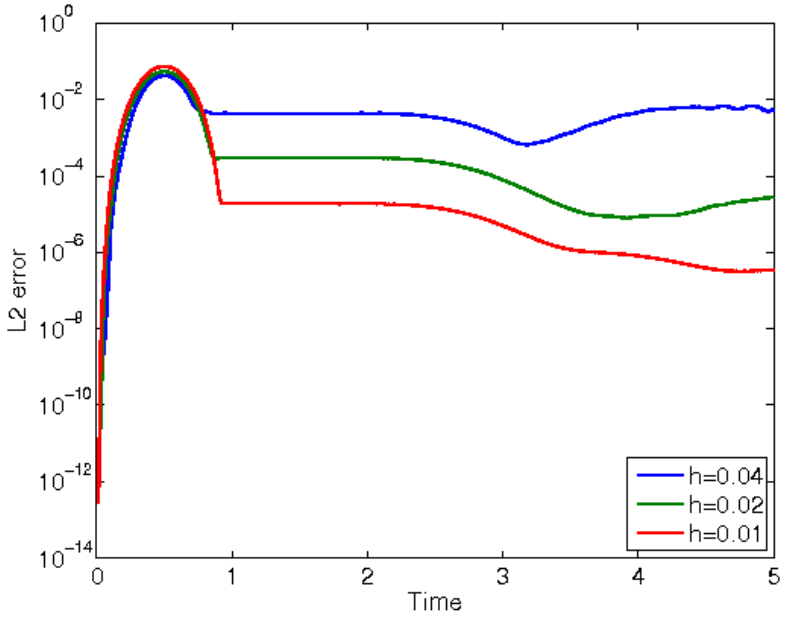


Figure A.5: The L_2-norm of the error in the point source test. This simulation was run to time $t = 5$ and the errors after time $t \approx 3$ are due to artificial reflections from the super-grid layers. The blue, green, and red lines correspond to $h = 0.04$, $h = 0.02$, and $h = 0.01$, respectively.

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 2.8×10^5 to 3.3×10^5 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*.

Configuration	Solver	Execution time [s]	Relative factor
Cartesian	elastic	$2.62 \cdot 10^{-8}$	1
Topography	elastic	$4.05 \cdot 10^{-8}$	1.5
Cartesian	visco-elastic	$7.96 \cdot 10^{-8}$	3.0
Topography	visco-elastic	$1.37 \cdot 10^{-7}$	5.2

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.

Configuration	solver	Memory [bytes/grid point]	Asymp. [bytes/grid point]
Cartesian grid	elastic	188	168
Topography	elastic	208	200
Cartesian grid	visco-elastic ($m = 3$)	475	448
Topography	visco-elastic ($m = 3$)	500	468

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 $2+11*m$ 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

[1] K. Aki and P. G. Richards. *Quantitative seismology*. University Science Books, Sausalito, CA, USA, 2nd edition, 2002.

[2] D. Appelö and T. Colonius. A high order super-grid-scale absorbing layer and its application to linear hyperbolic systems. *J. Comput. Phys.*, 228:4200–4217, 2009.

[3] J. M. Carcione. *Wave fields in real media: wave propagation in anisotropic, anelastic and porous media*, volume 31 of *Handbook of geophysical exploration: seismic exploration*. Pergamon, Elsevier Science, 2001.

[4] S. M. Day, J. Bielak, D. Dreger, S. Larsen, R. Graves, A. Pitarka, and K. B. Olsen. Test of 3D elastodynamic codes: Lifelines project task 1A01. Technical report, Pacific Earthquake Engineering Center, 2001.

[5] S. M. Day, J. Bielak, D. Dreger, S. Larsen, R. Graves, A. Pitarka, and K. B. Olsen. Test of 3D elastodynamic codes: Lifelines program task 1A02. Technical report, Pacific Earthquake Engineering Center, 2003.

[6] H. Emmerich and M. Korn. Incorporation of attenuation into time-dependent computations of seismic wave fields. *Geophysics*, 52(9):1252–1264, 1987.

[7] P. Goldstein, D. Dodge, M. Firpo, and L. Miner. *International Handbook of Earthquake and Engineering Seismology*, volume 81B, chapter SAC2000: Signal processing and analysis tools for seismologists and engineers, pages 1613–1614. International Association of Seismology and Physics of the Earth's Interior, 2003.

[8] B. Gustafsson, H.-O. Kreiss, and J. Oliger. *Time dependent problems and difference methods*. Wiley–Interscience, 1995.

[9] H.-O. Kreiss and N.A. Petersson. Boundary estimates for the elastic wave equation in almost incompressible materials. *SIAM J. Numer. Anal.*, 50:1556–1580, 2012.

[10] Horace Lamb. On the propagation of tremors over the surface of an elastic solid. *Phil. Trans. Roy. Soc. London, Ser. A*, 203, 1904.

[11] P. Liu, R. J. Archuleta, and S. H. Hartzell. Prediction of broadband ground-motion time histories: Hybrid low/high-frequency method with correlated random source parameters. *Bulletin of the Seismological Society of America*, 96:2118–2130, 2006.

[12] Harold M. Mooney. Some numerical solutions for Lamb's problem. *Bull. Seismo. Soc. Amer.*, 64, 1974.

[13] G. Evenden (original author). Proj.4 cartographic projections library. Source code and documentation available from trac.osgeo.org/proj.

[14] N. A. Petersson and B. Sjögreen. Stable grid refinement and singular source discretization for seismic wave simulations. *Comm. Comput. Phys.*, 8(5):1074–1110, November 2010.

[15] N. A. Petersson and B. Sjögreen. User's guide to WPP version 2.2. Technical Report LLNL-SM-487431, Lawrence Livermore National Laboratory, 2011. (Source code available from computation.llnl.gov/casc/serpentine).

[16] N. A. Petersson and B. Sjögreen. Stable and efficient modeling of anelastic attenuation in seismic wave propagation. *Comm. Comput. Phys.*, 12(1):193–225, 2012.

[17] N. A. Petersson and B. Sjögreen. Super-grid modeling of the elastic wave equation in semi-bounded domains. *Comm. Comput. Phys.*, 16:913–955, 2014.

[18] N. A. Petersson and B. Sjögreen. Wave propagation in anisotropic elastic materials and curvilinear coordinates using a summation-by-parts finite difference method. *J. Comput. Phys.*, 299:820–841, 2015.

[19] N. A. Petersson and B. Sjögreen. Installing SW4, version 2.0. Technical Report LLNL-SM-741310, Lawrence Livermore National Laboratory, 2017.

[20] B. Sjögreen and N. A. Petersson. A fourth order accurate finite difference scheme for the elastic wave equation in second order formulation. *J. Sci. Comput.*, 52:17–48, 2012. DOI 10.1007/s10915-011-9531-1.

[21] P. Wessel and W. H. F. Smith. New, improved version of generic mapping tools released. In *EOS trans. AGU*, volume 79, page 579, 1998.

Index