User's guide to the Wave Propagation Program (WPP) version 1.2-β[1]

N. Anders Petersson[2] Arthur J. Rodgers[3] Björn Sjögreen[2]

June 16, 2008

[1]Lawrence Livermore National Laboratory technical report LLNL-SM-230257. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

[2]Center for Applied Scientific Computing, Computations Directorate, Lawrence Livermore National Laboratory

[3]Atmospheric, Energy and Earth Department, CMELS Directorate, Lawrence Livermore National Laboratory

Contents

Chapter 1

Introduction

WPP is a parallel computer program for simulating time-dependent elastic and viscoelastic wave propagation, with some provisions for acoustic wave propagation. *WPP* solves the governing equations in displacement formulation using a node-based finite difference approach on a Cartesian grid, see [9] for details. *WPP* implements substantial capabilities for 3-D seismic modeling, with a free surface condition on the top boundary, non-reflective far-field boundary conditions on the other boundaries, (many) point force and point moment tensor source terms with many time dependencies, fully 3-D material model specification, output of synthetic seismograms in the *SAC* [3] format, output of *GMT* [11] scripts for laying out simulation information on a map, and output of 2-D slices of (derived quantites of) the solution field as well as the material model.

To allow the user to locally refine the computational mesh in areas where finer resolution is needed, version 1.1 of *WPP* implements local mesh refinement. The resolution of the solution is often charaterized in terms of the number of grid points per wave length. For a signal of frequency f, the wave length is V_s/f, where V_s is the wave speed in the material. Hence the grid size needs to be small where the wave speed is small, and vice versa. A locally refined mesh can therefore be used to reduce the variations in resolution in terms of grid points per wave length, which otherwise occur when a mesh with constant grid spacing discretizes a heterogeneous material model. For seismic applications, the savings in computational effort can be significant, since the wave speed often is an order of magnitude smaller in sedimentary basins near the surface than in the mantle below the MoHo.

In seismic applications, viscoelastic behavior is often significant for modeling the dissipative nature of realistic materials, especially for higher frequencies. Version 1.1 of *WPP* implements the eight term viscoelastic model described in [6], and models quality factors Q_p and Q_s for the attenuation of P- and S-waves. 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 tests/examples subdirectory of the *WPP* source distribution contains several examples and validation tests, which are discussed in Section 10 of this document.

Chapter 2

Getting started

2.1 Running *WPP*

WPP is run from the UNIX prompt with an input file name as its argument. The ASCII input file 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 *WPP*, so each command in the actual input file must be written on the same line.

Since *WPP* is a parallel code, it is required to be run under a parallel operating environment such as srun or mpirun. *WPP* reads all input from the file given as the first command line argument. For example,

```
shell> mpirun -np 2 wpp wpp.in
```

tells *WPP* to read input from a file named `wpp.in`. Throughout this document we use the convention that input files have the file suffix `.in`, but *WPP* reads files with any extension.

Running on the Livermore Computing parallel linux clusters:

```
shell> srun -ppdebug -N 16 -n 32 wpp xxx.in
```

The above command runs wpp on 16 nodes utilizing 32 processors on the debug parition using xxx.in as the input file. Note that the CPU time limit on the debug partition is 30 minutes. Jobs requiring more than 30 minutes and/or more than 32 processors must be submitted through the batch system using the psub command. Refer to the Livermore Computing web pages for detailed information (https://computing.llnl.gov).

Running on other platforms (Linux desktop/laptop):

```
shell> mpirun -np 2 wpp wpp.in
```

This command runs the wpp code on two processors, using `wpp.in` as the input file. Note: Before executing mpirun, you need to setup an mpd daemon (see mpich2-doc-user.pdf for more info).

version information (-v) This option outputs version information for the wpp executable. The same information is by default printed to standard out at the beginning of every run.

```
shell> mpirun wpp -v
----------------------------------------------------------------
            WPP Version 1.1
 Copyright (C) 2007 Lawrence Livermore National Security, LLC.

 WPP 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: Wed Dec 12 10:21:47 2007
  By:       andersp
  Machine:  tux230.llnl.gov
  Compiler: /usr/casc/wpp/tools/lib9.0.3/bin/mpicxx
  Lib:      /usr/casc/wpp/tools/lib9.0.3/lib
----------------------------------------------------------------
```

2.2 Introductory example: Lamb's problem

The version of Lamb's problem [5] considered here consists of a single vertical time-dependent point force acting downward on the surface of a uniform half-space. Receivers are placed on the surface at different distances from the source, see Figure 2.1.

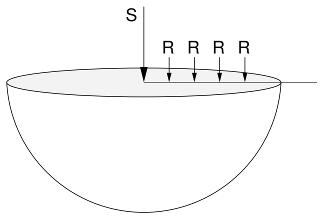


Figure 2.1: A schematic picture of Lamb's problem. A time-dependent force directed downward into an elastic half-space. Four recording stations are located on the surface which gather the vertical and radial displacement histories.

There are closed form solutions available for Lamb's problem (see e.g. [8]) which can be used to test that *WPP* is working properly by measuring the error for successively finer grids, and establishing the order of convergence.

In this example, the elastic half-space consists of a Poisson solid ($\lambda = \mu$) with S-wave velocity 1000 m/s and P-wave velocity $1000 \times \sqrt{3}\, m/s$ and density $1000\ kg/m^3$. Our elastic "half-space" consists of the rectangular box $(x, y, z) \in [0, 10000] \times [0, 10000] \times [0, 5000]$. 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 "RickerInt" signal (see Figure 4.2) with $\omega = 1$ Hz, $t_0 = 1$ s and magnitude 10^{13} N. We record the solution at four receivers on the surface at distances 1, 2, 3 and 4 km from the source. In *WPP* the above setup is realized by the input file shown below (with grid spacing $h = 40\ m$):

```
grid x=10000.0 y=10000.0 z=5000.0 h=40
time t=5.0
block vp=1732.05080756 vs=1000 r=1000
source x=5000 y=5000 z=0 fx=0 fy=0 fz=1e13 type=RickerInt \
```

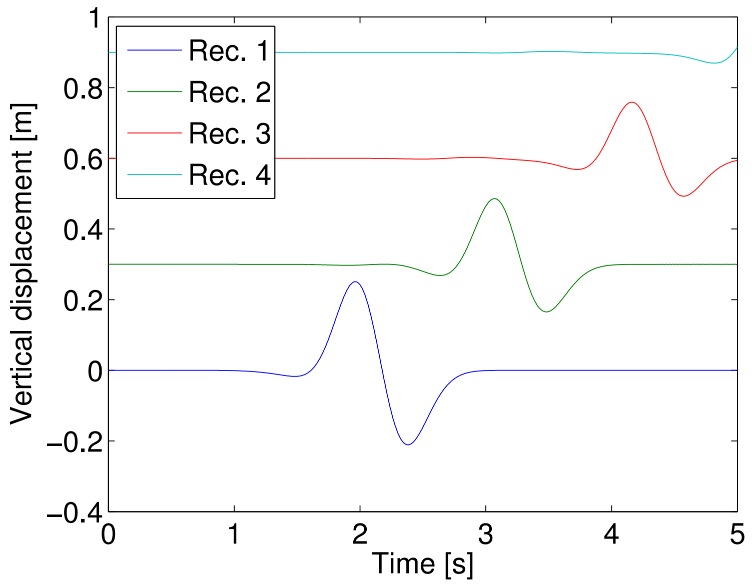


Figure 2.2: The vertical displacements as a function of time. The displacement recorded at receivers 2, 3 and 4 have been offset in the vertical direction to improve readability.

```
       freq=1.0 t0=1.0
sac x=6000 z=0 y=5000 file=s1
sac x=7000 z=0 y=5000 file=s2
sac x=8000 z=0 y=5000 file=s3
sac x=9000 z=0 y=5000 file=s4
```

Note that by default the computational domain has a free surface boundary condition at $z = 0$ and non-reflecting boundary conditions on the other five boundaries. When the time function is not explicitly given in the source command, *WPP* uses the default RickerInt function, which is defined in Section 4.1.

For the grid spacing $h = 40$ the vertical displacement for the different receivers can be found in Figure 2.2. The waveforms are all smooth and the problem appears to be well resolved. In Figure 2.3 the errors computed relative to the exact solution in the vertical displacement for the grid spacings $h = 10$, $h = 20$ and $h = 40$ are plotted. The errors decay by a factor of 4 as the grid spacing is reduced by a factor of 2, confirming the second order convergence of the method. To the right in Figure 2.3 the vertical displacement for the three different grid spacings are plotted together with the exact solution. In the "eye norm" there is no difference between the two finer grids and the exact solution.

Running Lamb's problem: The input file for Lamb's problem is provided with the source distribution of *WPP*, in the file `tests/examples/lamb.in`. Under the `srun` environment, the above example is executed by the command

```
thunder2{appelo2}70: srun -n16 -ppdebug ./wpp lamb.in
```

which produces the following output:

```
Running WPP on inputfile: lamb.in
* Setting nx to 251 to be consistent with h=40
* Setting ny to 251 to be consistent with h=40
```

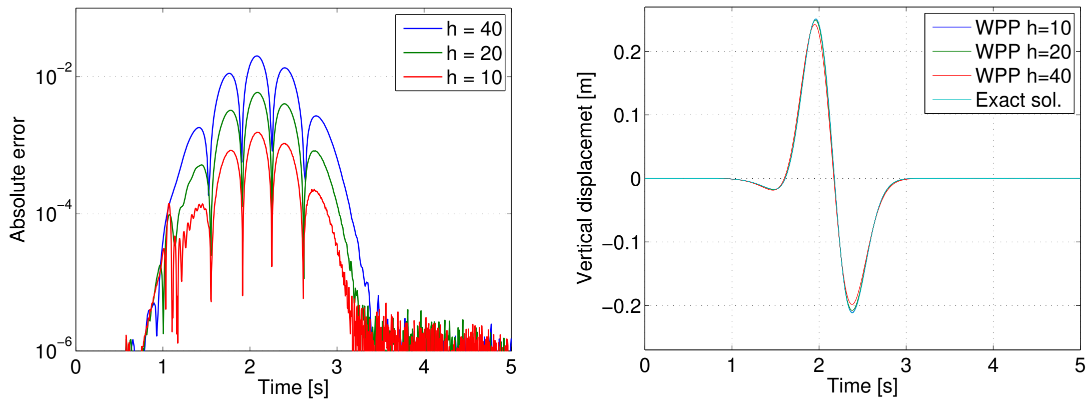


Figure 2.3: To the left: Max spatial error as function of time, for three different grid spacings. To the right: *WPP* solutions for grid spacings $h = 10, 20, 40$ together with the exact solution from [8].

```
* Setting nz to 126 to be consistent with h=40

  Nx=251, Ny=251, Nz=126, h=40

Block 1 has bounds 0 10000 0 10000 0 5000

   Execution time for process input phase 0.272136 seconds
Running WPP on 16 processors...
Writing output to directory: ./
         With name: wpp

       ----------- Material properties ranges ---------------
         For grid [0][0]

       1000 kg/m^3 <=  Density <= 1000 kg/m^3
       1732.05 m/s    <=  Vp      <= 1732.05 m/s
       1000 m/s    <=  Vs      <= 1000 m/s
       1.73205        <=  Vp/Vs   <= 1.73205
       1e+09 Pa     <=  mu      <= 1e+09 Pa
       1e+09 Pa     <=  lambda  <= 1e+09 Pa
       ------------------------------------------------------
----------------------------------------------------
 Making Output Directory: ./

... Done!
----------------------------------------------------
d4 coeff  = 0
curlcoeff = 3.72678e+41
SAC 0 moved to grid 0 0
SAC 1 moved to grid 0 0
SAC 2 moved to grid 0 0
SAC 3 moved to grid 0 0
============================================================
 Starting program wpp

----------------------------------------------------------------
            WPP Version 1.1
 Copyright (C) 2007 Lawrence Livermore National Security, LLC.

 WPP 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: Tue Dec 11 14:46:57 2007
  By:       appelo2
  Machine:  thunder3
  Compiler: /usr/global/tools/mpi/bin2.2/scripts/chaos_3_ia64_elan4/mpiicpc
  Lib:      /usr/gapps/wpp/chaos_3_ia64_elan4/tools/lib9.1.3.tau/lib
----------------------------------------------------------------

============================================================

 Using the following data :

 Start Time = 0 Goal Time = 5
 Number Steps = 349 dt: 0.0143266

-----------------------------------------------------------------------
  Total seismic moment (M0): 0 Nm
  Number of sources 1
-----------------------------------------------------------------------
Time step 1 t = 0
Time step 101 t = 1.43266
Time step 201 t = 2.86533
Time step 301 t = 4.29799
Time step 349 t = 4.98567
Writing BINARY SAC Files, of size 350: ./s3.201.126.1.[x|y|z]
Writing BINARY SAC Files, of size 350: ./s1.151.126.1.[x|y|z]
Writing BINARY SAC Files, of size 350: ./s4.226.126.1.[x|y|z]
Writing BINARY SAC Files, of size 350: ./s2.176.126.1.[x|y|z]
============================================================
 Wave Propagation Program (WPP) Finished!
============================================================
```

Chapter 3

Coordinate system, units and the grid

WPP uses a right-handed Cartesian coordinate system with the z-direction pointing downwards into the medium, see figure 3.1. *WPP* 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/second (m/s), densities are in kilogram/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.

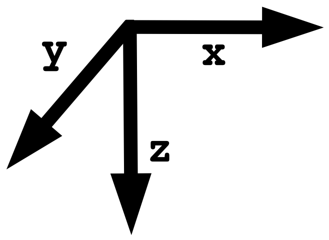


Figure 3.1: *WPP* uses a right handed coordinate system with the z-axis pointing downwards.

In *WPP* the computational domain is the box shaped region

$$0 \le x \le x_{max}, \quad 0 \le y \le y_{max}, \quad 0 \le z \le z_{max}. \tag{3.1}$$

The grid command in the input file specifies the extent of the computational domain and the grid size h. When mesh refinement is enabled (see Chapter 7), this is the grid size in the coarsest grid. The most obvious 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 line 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,

```
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:

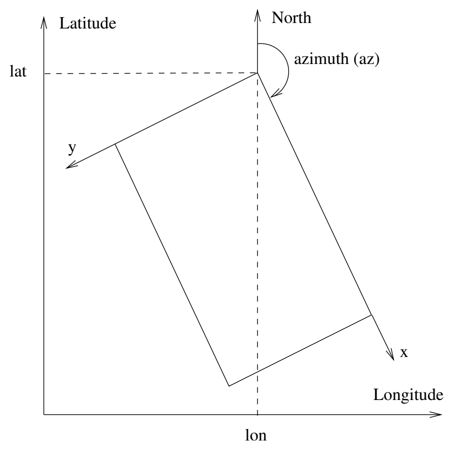


Figure 3.2: Geographical coordinates in *WPP*.

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. For example

$$nx = \text{(int)}1.5 + x/h, \tag{3.2}$$

rounds nx to be the nearest integer value of $1 + x/h$. The extent in the x-direction is thereafter adjusted to

$$x = (nx - 1)h. \tag{3.3}$$

A corresponding procedure is performed in the other directions.

The third option is to give the spatial range in each of the three dimensions and specify the number of grid points in a particular direction:

```
grid x=30000.0 y=20000.0 z=10000.0 nx=100
```

In this case, the grid spacing is be 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 (3.2) 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 (3.3). The syntax for the grid command is given in Section 9.1.

3.1 Geographic coordinates

WPP supports geographic coordinates as an alternative way of specifying spatial locations, see Figure 3.2. The location of the Cartesian coordinte system is specified in the grid command, and if no location is given the origin ($x = 0$, $y = 0$, $z = 0$) defaults to latitude 37 degrees (North), longitude -118 degrees (West), with a 135 degree azimuthal angle from North to the x-axis. By default the elevation is at sea level. The latitude (ϕ) and longitude (θ) are calculated using the approximative formulae (where lat, lon, and az are in degrees)

$$\phi = \text{lat} + \frac{x\cos(\alpha) - y\sin(\alpha)}{M}, \quad \alpha = \text{az}\frac{\pi}{180}, \tag{3.4}$$

$$\theta = \text{lon} + \frac{x\sin(\alpha) + y\cos(\alpha)}{M\cos(\phi\pi/180)}, \tag{3.5}$$

where $M = 111319.5$ meters/degree. You can change the location and orientation of the grid by specifying the latitude and longitude of the grid origin, and the azimuthal angle between North and the x-axis. For example:

```
grid h=500.0 x=30000.0 y=20000.0 z=10000.0 lat=39.0 lon=-117.0 az=150
```

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

Chapter 4

Sources, time-functions and grid sizes

4.1 Sources and time-functions in *WPP*

WPP solves the elastic wave equation in displacement formulation,

$$\begin{aligned} \rho \mathbf{u}_{tt} &= \nabla \cdot \mathcal{T} + \mathbf{F}(\mathbf{x}, t), \qquad \mathbf{x} \text{ in } \Omega,\ t \geq 0, \\ \mathbf{u}(\mathbf{x}, 0) = 0, &\quad \mathbf{u}_t(\mathbf{x}, 0) = 0, \qquad\qquad \mathbf{x} \text{ in } \Omega, \end{aligned}$$

where ρ is the density, $\mathbf{u}(\mathbf{x}, t)$ is the displacement vector, $\mathcal{T}$ is the stress tensor. The computational domain Ω is the box shaped region (3.1). By default, first order Clayton-Engquist non-reflecting boundary conditions are imposed on all sides of the computational domain, except on $z = 0$ where a stress-free boundary condition is imposed:

$$\mathcal{T} \cdot \mathbf{n} = 0, \quad z = 0,\ t \geq 0.$$

Here $\mathbf{n}$ is the normal vector to the $z = 0$ plane.

There are six linearly independent invariants in the solution of the elastic wave equation: translations along the three coordinate directions and rotations along the three coordinate axes. The projection command (Section 9.6.2) removes these invariants from the solution at regular time intervals. By default, projection occurs every 1000 time steps, ensuring the calculated displacements be relative to a reference frame which is fixed with respect to the computational domain. Projection takes a small amount of computational effort, so the calculation will execute slightly faster if the projection interval is made longer. The amount of correction to the solution is interpolated linearly in time in between projections, so the solution becomes (slightly) more accurate by projecting more often. To change the default projection interval to every 100 time steps, you put the following line in the input file:

```
projection projectionInterval=100
```

In situations where external point forces are applied such that their sum is non-zero, it is inaccurate to remove the solid body motion. In these cases, projection should be disabled by setting `projectionInterval=0`.

The forcing term $\mathbf{F}$ consists of a sum of point force and point moment 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 F_0 is the amplitude, $\mathbf{x}_0 = (x_0, y_0, z_0)$ is the location of the point force in space, $g(t, t_0, \omega)$ is the time function, with offset time t_0 and frequency parameter ω, and (F_x, F_y, F_z) are the Cartesian components of the force vector. Each of the Cartesian components are scaled by the amplitude F_0. For a point moment source we have

$$\mathbf{F}(\mathbf{x},t) = g(t,t_0,\omega)M_0\,\mathcal{M}\cdot\nabla\delta(\mathbf{x}-\mathbf{x_0}), \quad \mathcal{M} = \begin{pmatrix} M_{xx} & M_{xy} & M_{xz} \\ M_{xy} & M_{yy} & M_{yz} \\ M_{xz} & M_{yz} & M_{zz} \end{pmatrix}.$$

In this case the seismic moment of the moment tensor is M_0, otherwise the notation is the same as for a point force. Note that the moment tensor always is symmetric.

In *WPP* the forcing is specified in the input file using the `source` command. 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 m0=1e15 mxx=1 myy=1 mzz=1 \
       type=RickerInt t0=1 freq=5
```

which specifies an isotropic source (explosion) at the point $\mathbf{r}_0 = (5000, 4000, 600)$ with amplitude 10^{15} Nm, using the RickerInt time function with offset time $t_0 = 1$ s and frequency parameter $\omega = 5$ Hz. This command sets the off-diagonal moment tensor elements (M_{xy}, M_{xz} and M_{yz}) to zero (which is the default value).

A convenient way of specifying moment sources is by using the dip, strike and rake angles (see Section 9.4 for syntax) defined in Aki and Richards [1]. In this case, the total seismic moment $\sum M_0$ [Nm] is related to the moment magnitude by the formula

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

After parsing all source commands in an input file, *WPP* outputs the moment magnitude using this formula.

Notes about sources and time functions:

- It is not necessary to place the sources exactly on grid points. The discretization of the source terms is second order accurate for any location within the computational domain.

- The source time function can be selected from the set of predefined functions described below. All functions start from zero ($\lim_{t\to-\infty} g(t,t_0,\omega) = 0$). The Gaussian and the 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 time derivative of $g(t,t_0,\omega)$, is often refered to as the moment-rate function.

- For moment tensor sources, a displacement formulation code (such as *WPP*) uses the time-integral of the time function used by velocity-stress formulation codes (such as *E3D*). For example, if you use a Gaussian time function in *E3D*, you should use the GaussianInt function in *WPP* to get the same effect. However, the same type of time-function is used for point forces in both types of codes.

- The Triangle, Sawtooth, Ramp, Smoothwave, Brune, BruneSmoothed, Liu and VerySmoothBump functions are identically zero for $t < t_0$, so they will give reasonable simulation results if $t_0 \geq 0$. However, the Gaussian, GaussianInt, Ricker, and RickerInt functions are centered around $t = t_0$

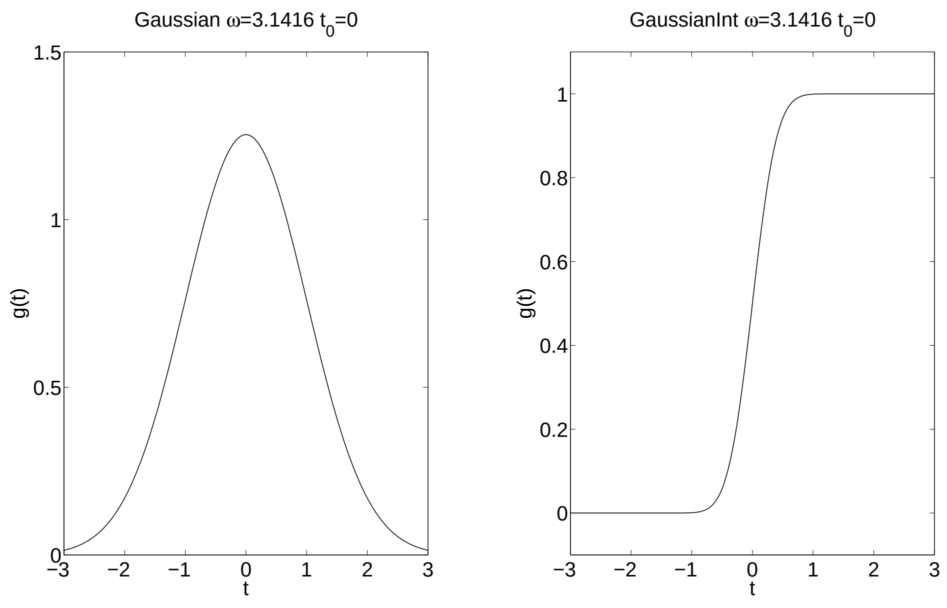


Figure 4.1: Gaussian (left) and GaussianInt (right) with $\omega = \pi$ and $t_0 = 0$.

with exponentially decaying tails for $t < t_0$. Hence t_0 must be positive and of the order $\mathcal{O}(1/\omega)$ to avoid incompatibilty problems with the initial conditions. We recommend choosing t_0 such that $g(0, t_0, \omega) \leq 10^{-8}$ for these functions.

4.1.1 Gaussian

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

Note that $\sigma = 1/\omega$ in terms of the usual notation for a Gaussian function. A plot of the Gaussian time-function is shown in Figure 4.1.

4.1.2 GaussianInt

The GaussianInt function is often used in earthquake modeling since it leads to a permanent displacement.

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

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

4.1.3 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.

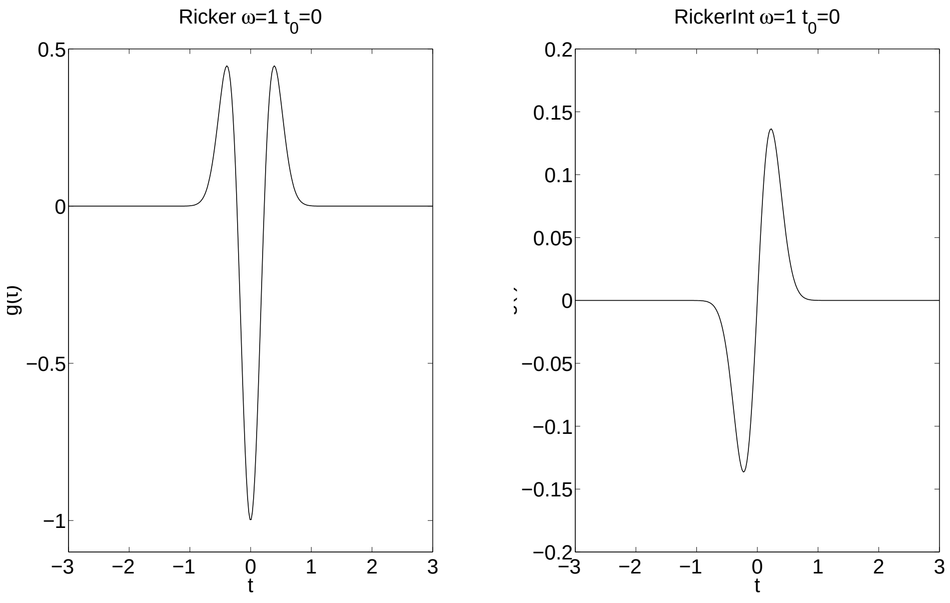


Figure 4.2: Ricker (left) and RickerInt (right) with $\omega = 1$ and $t_0 = 0$.

4.1.4 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. The RickerInt function is sometimes used in seismic exploration simulations. 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.

4.1.5 Brune

The Brune function has one continuous derivative but its second derivative is discontinuous at $t = t_0$,

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

The Brune function is often used in earthquake modeling.

4.1.6 BruneSmoothed

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

$$g(t, t_0, \omega) = \begin{cases} 0, & t < t_0, \\ 1 - e^{-\omega(t-t_0)} \Big[1 + \omega(t - t_0) + \dfrac{1}{2}(\omega(t - t_0))^2 & \\ \quad - \dfrac{3}{2x_0}(\omega(t - t_0))^3 + \dfrac{3}{2x_0^2}(\omega(t - t_0))^4 - \dfrac{1}{2x_0^3}(\omega(t - t_0))^5 \Big], & 0 < \omega(t - t_0) < x_0, \\ 1 - e^{-\omega(t-t_0)}(1 + \omega(t - t_0)), & \omega(t - t_0) > x_0. \end{cases}$$

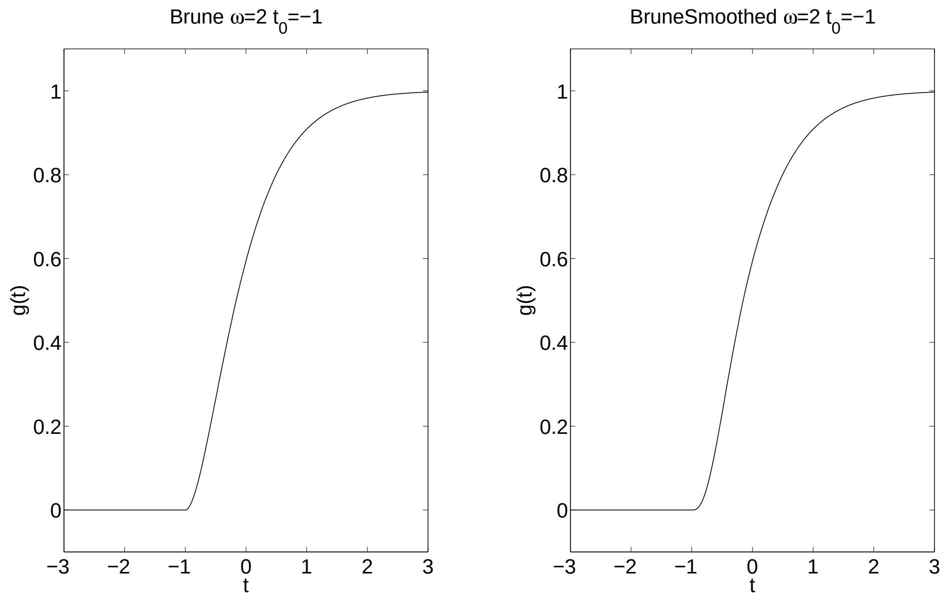


Figure 4.3: Brune (left) and BruneSmoothed (right) with $\omega = 2$ and $t_0 = -1$.

The parameter is fixed to $x_0 = 2.31$. Plots of the Brune and BruneSmoothed time-functions are shown in Figure 4.3. Since the BruneSmoothed function has three continuous derivatives, it generates less high frequency noise and leads to better accuracy for finer grids, compared to the Brune function (see Section 4.2.1).

4.1.7 Liu

This function was given in a paper by Liu et al., [7]. It is defined by

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

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 ressembles the Brune function, but the rise is somewhat steeper for small $t - t_0$, see Figure 4.4.

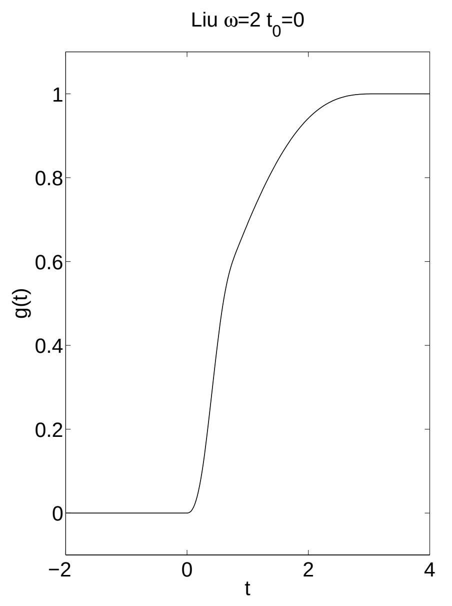


Figure 4.4: Liu time function with $\omega = 2$ and $t_0 = 0$.

4.1.8 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.

4.1.9 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.

4.1.10 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.

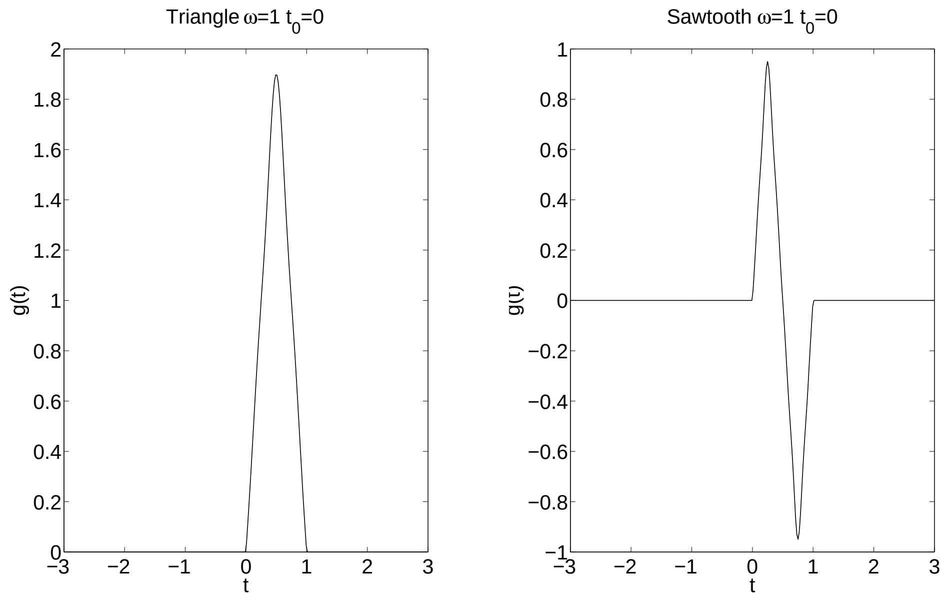


Figure 4.5: Triangle (left) and Sawtooth (right) with $\omega = 1$ and $t_0 = 0$.

4.1.11 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.

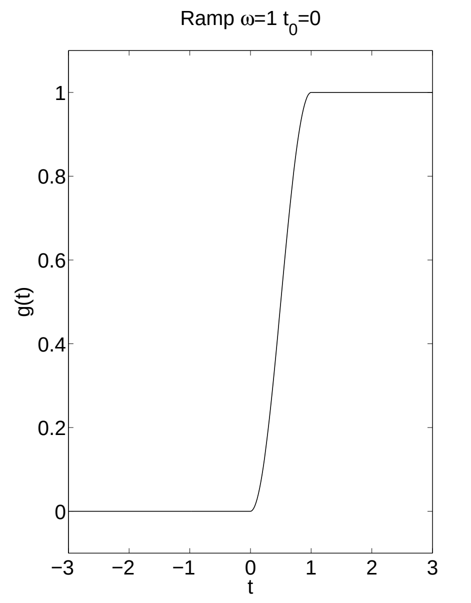
4.1.12 VerySmoothBump

$$g(t, t_0, \omega) = \begin{cases} 0, & t < t_0, \\ -1024(\omega(t-t_0))^{10} + 5120(\omega(t-t_0))^9 - 10240(\omega(t-t_0))^8 & \\ \quad + 10240(\omega(t-t_0))^7 - 5120(\omega(t-t_0))^6 + 1024(\omega(t-t_0))^5, & t_0 \le t \le t_0 + 1/\omega, \\ 0, & t > t_0 + 1/\omega. \end{cases}$$

A plot of the VerySmoothBump time-function is shown in Figure 4.7.

4.2 How fine does the grid need to be?

When preparing the input file, the most difficult parameter to choose is the grid size h. It is extremely important to use a grid size which is sufficiently small to adequately resolve the waves which are generated by the source. On the other hand we don't want to use an unnecessarily small grid size, because both the computational demand and the memory requirements increase with decreasing grid size.



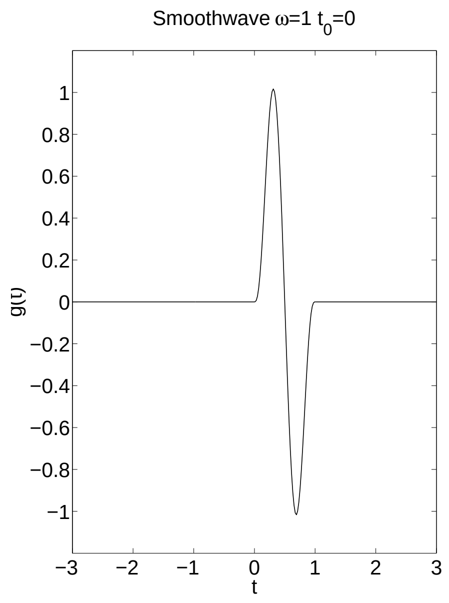


Figure 4.6: Ramp (left) and Smoothwave (right) with $\omega = 1$ and $t_0 = 0$.

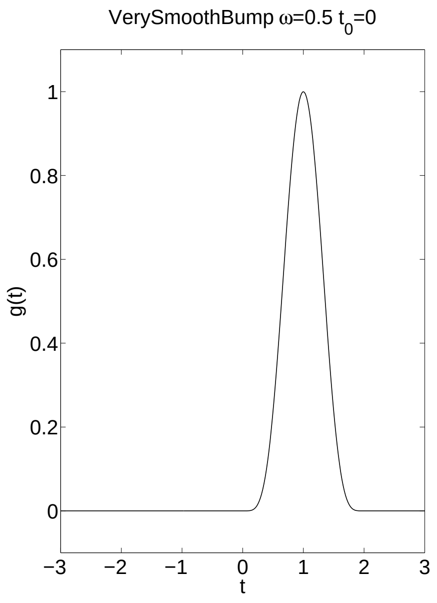


Figure 4.7: VerySmoothBump with $\omega = 0.5$ and $t_0 = 0$.

The number of points per wavelength, P, is a normalized measure of how well a solution is resolved on the computational grid. Since the shear waves have the lowest velocities and a shorter wave length than the compressional waves, the shortest wave length can be estimated by

$$\min \lambda = \frac{\min V_s}{f},$$

where V_s is the shear velocity of the material and f is the the temporal frequency of the wave. Hence the number of grid points per wave length equals $\min \lambda / h$, which is given by

$$P = \frac{\min V_s}{h\, f}. \tag{4.1}$$

Note that h needs to be made smaller to maintain the same value of P if either V_s is decreased or if the frequency is increased. *WPP* uses a second order accurate discretization and as we shall see below, ***P** needs to be around 15 or larger* to obtain a reasonably accurate solution, but the exact number depends on the particular type of time-function as well as the distance between the source and the reciever.

In formula (4.1), $\min V_s$ is found from the material properties and h is determined by the input grid specification. The frequencies present in the solution are determined by the frequencies present in the time function(s) in the source term(s). We here show some examples of computed solutions with different source time functions.

We place a single moment source in a homogeneous material of size 200×200×100 km, with a free surface on the top and non-reflecting boundary conditions on all other sides. The source is located at the point $(100, 100, 3.5)$ km and has a strength of 1.75×10^{17} Nm. The only non-zero component in the moment tensor is the xy element. The material has P-wave speed 6941 m/s, S-wave speed 3949 m/s, and density 2994 kg/m^3. This is expressed in the following input file to `wpp`, (see `tests/examples/source-resolution.in`),

```
grid x=200e3 y=200e3 z=100e3 h=1000
time t=30
block vp=6951 vs=3949 r=2994
source x=100e3 y=100e3 z=3.5e3 m0=1.75e17 Mxy=-1 freq=0.25 \
                               type=RickerInt t0=5.0
sac x=100e3 y=150e3 z=0 file=surf3
```

where we also note that the solution is sampled at the point $(100, 150, 0)$ km. Figure 4.8 displays the x-displacements as function of time at the point $(100, 150, 0)$ km for a sequence of four different grids, using the RickerInt time-function. Since the results with $h = 250$ m and $h = 500$ m are very close, we can take the $h = 250$ m (green) curve as a good approximation of the exact solution. The red curve, $h = 1000$ m predicts the shape of the solution correctly, although small errors are visible in the plot. However, the numerical errors in the blue curve with $h = 2000m$ are not acceptable since the peak amplitude is not well predicted and there is significant artificial ringing in the solution after time $t = 21$. If we take $f =$ `freq` and $h = 1000$ in formula (4.1),

$$P = 3949/(1000 * 0.25) = 15.8.$$

We conclude that for the RickerInt time function, $P \approx 15$ gives an acceptable accuracy in the solution.

To further investigate how the grid size effects the solution, we study the Fourier transform of the time-signals, see Figure 4.9. Note that the solutions using the coarser mesh deviate mostly in the higher frequencies. Also note that there is significant energy in modes up to 0.7 Hz, which is higher than the frequency parameter (`freq`=0.25) in the RickerInt time function.

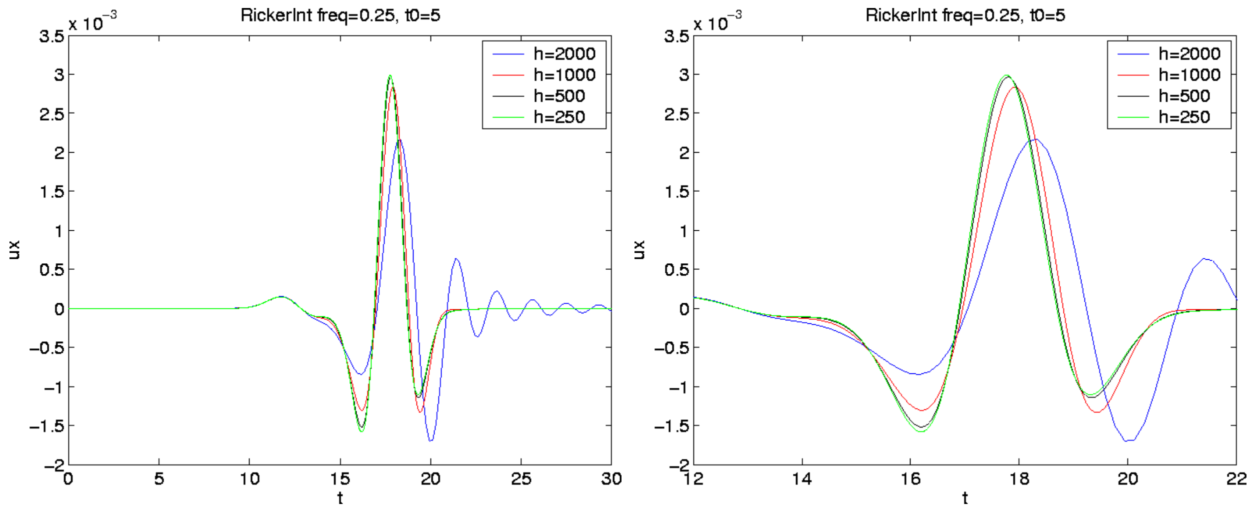


Figure 4.8: The x-component of the displacement due to a source with a RickerInt time function with frequency parameter `freq`= 0.25 on four different grids. Enlarged view to the right.

h	P
200m	64
500m	32
1000m	16
2000m	8

Table 4.1: Approximate grid points per wavelength (P) for a given grid spacing (h)

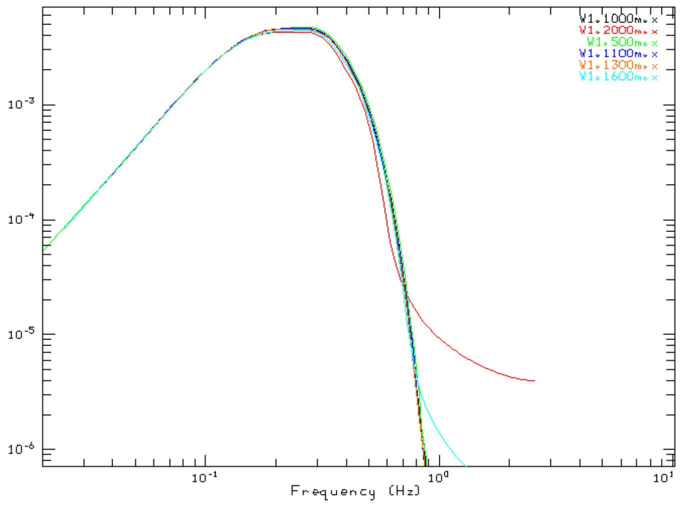


Figure 4.9: The Fourier transform of the time signal for different grid spacings.

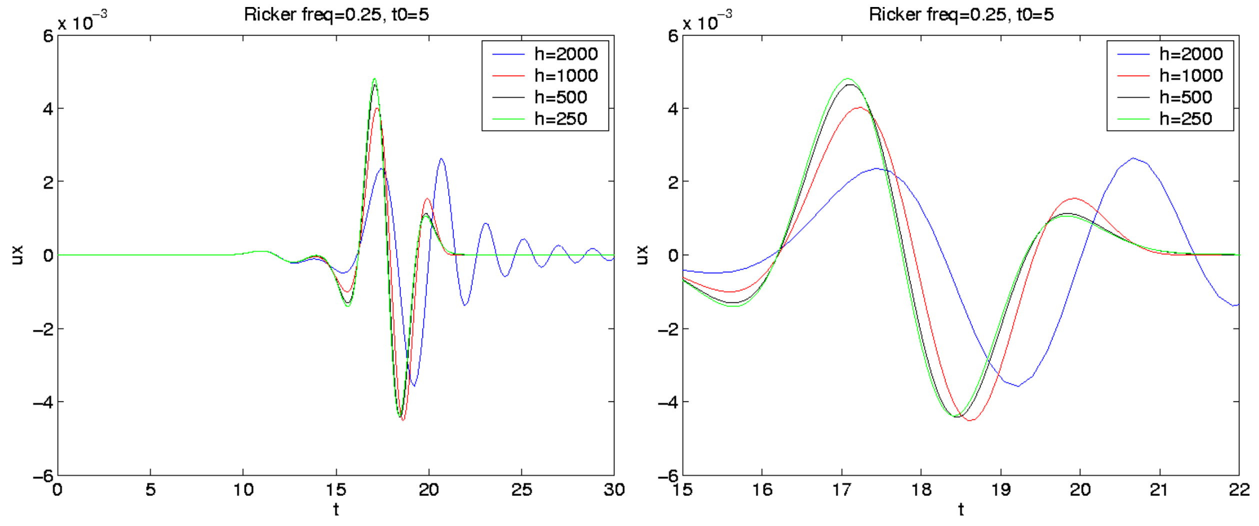


Figure 4.10: The x-component of the displacement due to a source with a Ricker time function with frequency parameter `freq`=0.25 on four different grids. Enlarged view to the right.

Figure 4.10 shows the results from computing with the Ricker (Sec. 4.1.3) time function instead of the RickerInt time function. The accuracy is similar to the accuracy in the computation with RickerInt. This is expected because the frequency content in the two computations are similar, which leads to a similar value for the number of points per wave length.

Figure 4.11 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. To estimate the frequency content, we use the approximate formulae:

$$f \approx \begin{cases} \texttt{freq}, & \text{for Ricker and RickerInt}, \\ \texttt{freq}/(2\pi), & \text{for Brune, BruneSmoothed, Gaussian, and GaussianInt.} \end{cases} \tag{4.2}$$

The plots in Figure 4.11 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. Furthermore, Figure 4.11 illustrates that the Fourier transform of the Ricker, RickerInt and Gaussian functions decay exponentially for large frequencies, while the Fourier transform of the Brune function decays much slower for high frequencies. The slow decay of the high modes arise from the lack of smoothness in the Brune function at $t = 0$. As a result, simulations using the Brune function are harder to resolve on the grid.

We next study solutions obtained with the Gaussian (Sec. 4.1.1), GaussianInt (Sec. 4.1.2) and the Brune (Sec. 4.1.5) time functions. To obtain a similar frequency content as for the Ricker and the RickerInt functions, we use the frequency parameter `freq`=1.5 in these computations. The source command in the input file then becomes

```
source x=100e3 y=100e3 z=3.5e3 m0=1.75e17 Mxy=-1 freq=1.5 \
       type=GaussianInt t0=5.0
```

Computations with the Gaussian, GaussianInt, and Brune functions are shown in Figures 4.12, 4.13, and 4.14 respectively. Even though there is more ringing in the calculation using the Brune function, the accuracy in these calculations is seen to be similar to the previous computations with the Ricker and RickerInt time

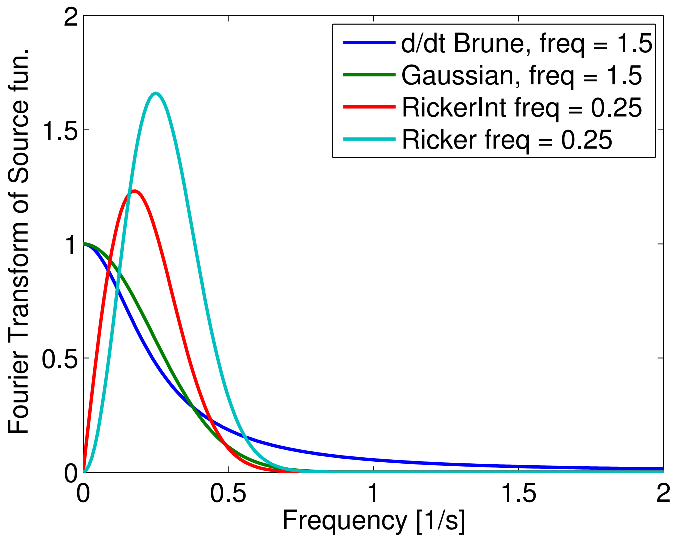


Figure 4.11: Absolute values of the Fourier transforms of the derivative of the Brune (dark blue), the Gaussian (green), the RickerInt (red), and the Ricker (light blue) time-functions. Here `freq`=1.5 for the Gaussian and the derivative of the Brune function, and `freq`=0.25 for Ricker and RickerInt.

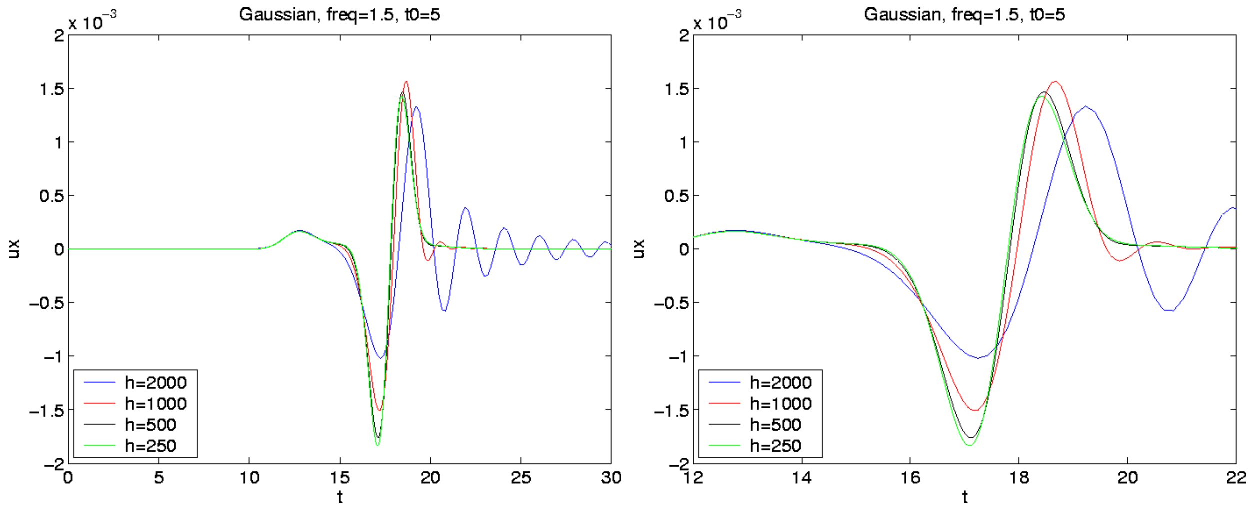


Figure 4.12: The x-component of the displacement due to a source with a Gaussian time function with frequency parameter `freq`=1.5 on four different grids. Enlarged view to the right.

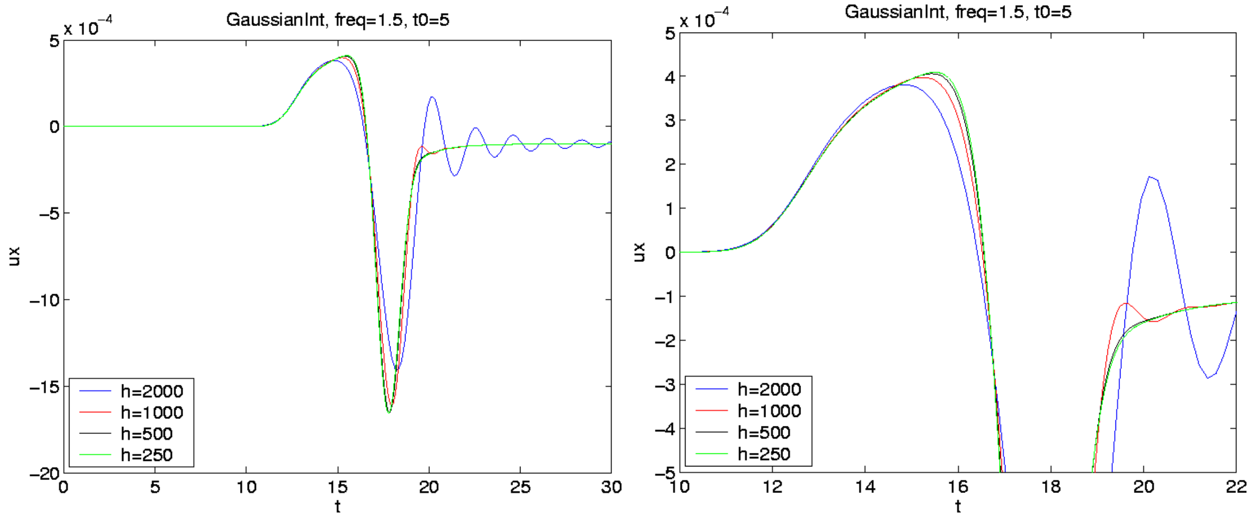


Figure 4.13: The x-component of the displacement due to a source with a GaussianInt time function with frequency parameter `freq`=1.5 on four different grids. Enlarged view to the right.

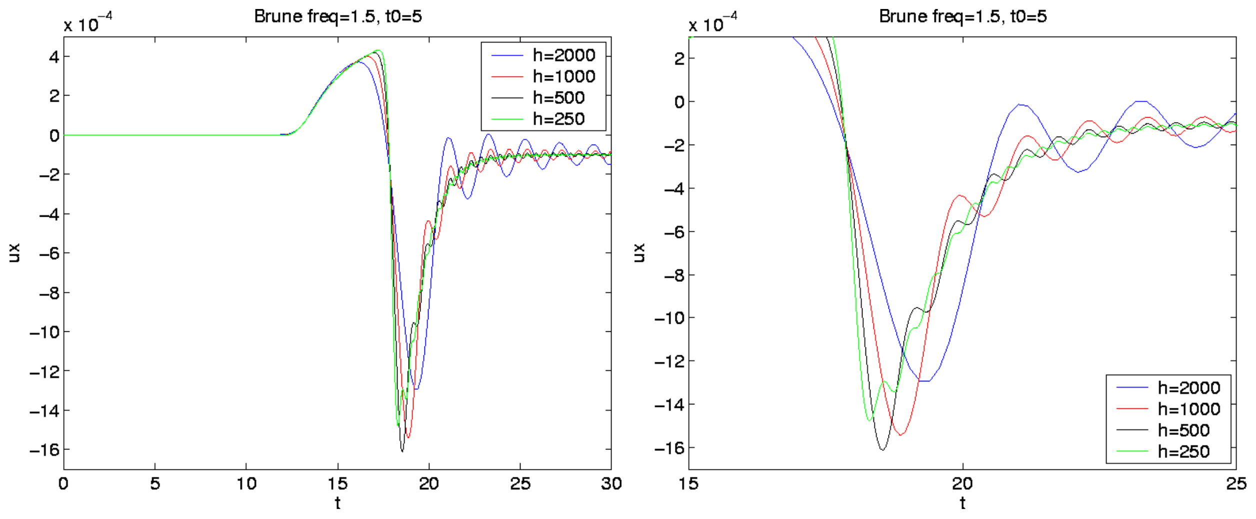


Figure 4.14: The x-component of the displacement due to a source with a Brune time function with frequency parameter `freq`=1.5 on four different grids. Enlarged view to the right.

functions. We conclude that the grid size $h = 1000$ is adequate for representing the solution on the grid. If we use formula (4.1) with the frequency scaling (4.2) and $h = 1000$, we obtain

$$P = 3949/(1000 * 1.5/(2\pi)) = 16.5.$$

Hence, $P \approx 15$ gives a solution with acceptable accuracy also for the Gaussian, GaussianInt, and Brune time functions.

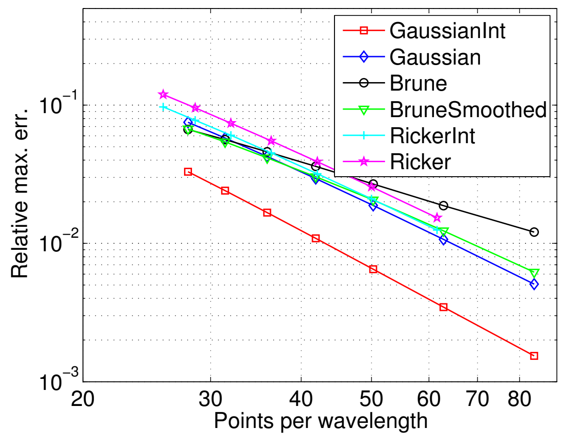
4.2.1 Lamb's problem revisited

We now compute solutions to Lamb's problem in a material with $V_p = \sqrt{3}\ km/s$, $V_s = 1\ km/s$ and the density $1000kg/m^3$. The solution is forced downward with amplitude `fz=5e13 N` and with a time function centered at time `t0=25 s`. For various time functions the solution is recorded at receivers 10 and 50 km from the source. At the recievers the relative error

$$\frac{\|u_{exact}(t) - u_{computed}(t)\|_\infty}{\|u_{exact}(t)\|_\infty},$$

in the horizontal component is computed and plotted in Figure 4.15. In these calculations, the grid size was held constant and the frequency parameter `freq` was varied. The number of points per wavelength was computed by (4.1) Hence, a lower number of points per wave length corresponds to a higher value of `freq`.

From Figure 4.15 we see that for all of the time functions, except the Brune function, there is a decrease in error inversely proportional to the square of the number of points per wavelength. The errors are larger for the Brune function since its spectrum decays much slower due to its discontinuous second derivative at $t = t_0$. The difference in the error levels between the left and the right sub-figures are due to the fact that errors in the numerical solution accumulate as the solution propagates away from the source. For a single harmonic wave, and a second order accurate finite difference method, the number of points per wavelength required to achieve a certain error is proportional to the square root of the number of wavelengths the wave propagates (see Chapter 3 in [4] for a detailed discussion). Thus, to get the same accuracy at five times the distance from the source, we need to use about $\sqrt{5} \approx 2.24$ times more points per wave length. This could be achived by reducing the grid size by a factor 2.24 in each direction, resulting in a factor of 11.1 times more grid points and an increase in CPU time by a factor 25.



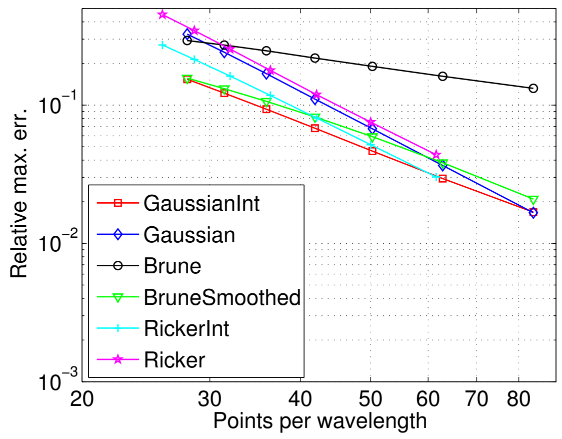


Figure 4.15: Relative errors for different source functions 10 (left) and 50 km (right) from the source. For the Brune time function the error decays much slower than for the other time functions.

Chapter 5

The material model

The basic material model in *WPP* is defined by the values of the density, ρ, the compressional velocity, V_p, and the shear velocity, V_s, at each grid point. These values can be specified by the block command (see, Section 5.1), the vfile command (see Section 5.2), the efile command (see Section 5.3), the pfile command (see Section 5.4) or by a combination of them. By default, these commands apply to the entire computational domain (as defined by the grid command), but can be restricted to a portion of the domain by the sub-region options described in Table 9.1.

It is also possible to add more advanced modeling of dissipative materials (visco-elastic damping), see section 6.

Note: The order within the material commands (block, vfile, pfile, and efile) *does* matter (unlike all other commands) in that the priority of the material command increase 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.

5.1 The block command

The block command can be used to specify material properties in rectangular volumes (sub-regions), 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 r=2000
 block vp=6000 vs=3500 r=2700 z1=15000 z2=35000
 block vp=8000 vs=4500 r=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 the above 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 r=1000 \
            x1=4000 x2=8000 y1=3000 y2=7000 z1=10000 z2=70000
```

To the left in Figure 5.1 an image slice of V_p through $x = 50,000$ is displayed.

The following example combines several block commands used to generate the material model displayed to the right in Figure 5.1:

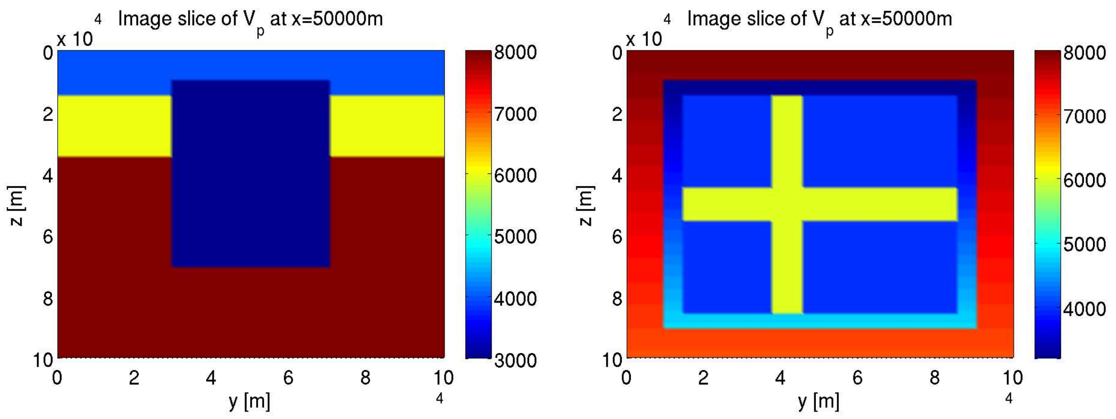


Figure 5.1: Examples of material models specified with the block command.

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

5.1.1 Modeling water with blocks

Water has density $\rho \approx 1000\ kg/m^3$, $V_p \approx 1500\ m/s$ and $V_s = 0\ m/s$. With the block command it is easy to include a layer of water near the top of the domain, for example:

```
block vp=6000 vs=3500 r=2700
block vp=1500 vs=0 r=1000 z1=0 z2=500
```

Even though no physical shear waves can propagate in water there may still be spurious numerical shear waves in the water. To remove such spurious waves the damping command can be used with the curlcurl option:

```
damping curlcurl=0.1
```

The curl-curl damping adds the term

$$-d_2 \nabla \times (\nabla \times u_t),$$

to the elastic wave equation in grid points where $V_s = \mu = 0$, i.e., in the parts of the computational domain where water or some other fluid is present. Note that no damping is added in the solid parts of the model, i.e., where $V_s > 0$.

The curl-curl damping coefficient d_2 is not taken into account when the time step is computed. The simulation can therefore go unstable if the coefficient is too large. The value of d_2 is defined as the fraction of the maximum value allowed by the CFL stability condition when only the dissipation term is taken into account and all other terms of the basic scheme are ignored. In most cases the simulation stays stable when $d_2 \leq 0.1$ but the exact limit depends on the details of the material model, the grid size, extent of the domain, etc. We recommend using $d_2 = 0.05$.

5.2 The vfile command

The vfile command can be used to read a binary raster file (vfile) that contains the values of a given model feature (P-velocity, S-velocity, or density) at each point or a subset of the points in the grid. Note that these files only contain the raw binary floating point values (4 bytes) for each grid point, and do not contain any header metadata information (such as the number of grid points in each direction or the grid size). Assuming that the files `vp.flt`, `vs.flt` and `r.flt` contain values for the P-velocity, the S-velocity and the density at all grid points, the material model can be specified by the following lines in the input file:

```
vfile type=vp file=vp.flt
vfile type=vs file=vs.flt
vfile type=r file=r.flt
```

The values in each binary file are read in the following order:

```
for j = j2, j1
   for k = k1, k2
      for i = i1, i2
         read vp(i,j,k);
```

Note that the values are traversed in increasing order in the x-direction (i) and in the z-direction (k), but in decreasing order in the y-direction (j). This is because vfiles are specified (for historical reasons) in a left handed coordinate system, where the y-direction is reversed compared to the right handed system used by *WPP*. If i1, j1, k1, i2, j2 and k2 are not explicitly stated they default to the entire grid: i1$=$j1$=$k1$=$1, i2$=$nx, j2$=$ny, k2$=$nz. A box shaped sub-region of the computational grid can be specified by

```
vfile i1=51 i2=61 k1=1 k2=11 type=vp file=vp.flt
vfile i1=51 i2=61 k1=1 k2=11 type=vs file=vs.flt
vfile i1=51 i2=61 k1=1 k2=11 type=r file=r.flt
```

The above lines specifies the material model in the sub-domain $51 \leq i \leq 61$ in the x-direction, $1 \leq k \leq 11$ in the z-direction, and $1 \leq j \leq$ny (entire range) in the y-direction.

There is a one-to-one mapping between the points in the vfile and the computational grid within the specified region. Thus, there must at least be $(\text{i2}-\text{i1}+1)\times(\text{j2}-\text{j1}+1)\times(\text{k2}-\text{k1}+1)$ values in the vfile. Since a floating point number is 4 bytes long, the size of the vfile must be at least $4 \times (\text{i2}-\text{i1}+1) \times (\text{j2}-\text{j1}+1) \times (\text{k2}-\text{k1}+1)$ bytes. If the vfile is smaller than that, *WPP* exits with an error message. It is however allowed for the vfile to contain more values than the computational grid. For example, suppose that the computational grid has nx=100 grid points in the x-direction. If the vfile exactly maps to the computational grid then i1=1 and i2=100. However, if the vfile only contains 31 points in the x-direction, we need to use i1 and i2 such that i2-i1+1=30, e.g. i1=20 and i2=50. The vfile is also allowed to represent a region which is larger than the computational grid. If the vfile contains 200 values in the x-direction, we could use i1=-49 and i2=150. Then the data in the vfile corresponding to $-49 \leq i \leq 0$ and $101 \leq i \leq 150$ are ignored. Extreme caution is required whenever there is a missmatch between the number of data points in the vfile and the number of grid points in the computational grid.

Note that it is *not* allowed to specify the sub-domain boundaries using the x1, x2, y1, y2, z1, z2 options, because roundoff is likely to induce problems when physical coordinates are converted to grid point indices, which sometimes cause the number of points to be ”off by one”.

As a final note, we strongly recommend verifying the material model before starting a simulation. For example, the image command (see Section 8.4) can be used to output a couple of cross sections through the computational domain to verify that the material model has reasonable values throughout the domain. The range of material values is also reported by *WPP* before the time-stepping starts in a simulation.

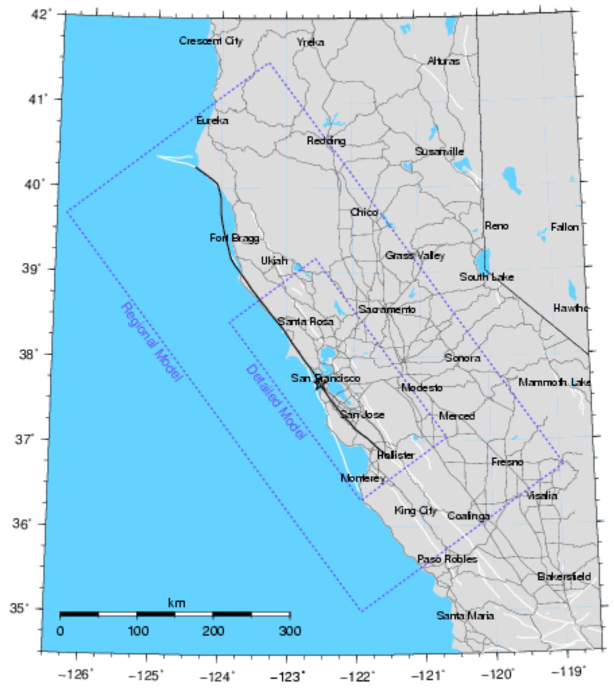


Figure 5.2: The geographical location where the efile model=SF is valid.

5.3 The efile command

The efile command is used to read in material properties from an etree database file. It is similar to the vfile command in that it can either specify the material properties in the entire computational domain, or within a box shaped sub-region. One major advantage compared to the vfile command is that the same etree database file can be used independently of the grid size, so there is no need to have a one-to-one mapping between the etree model and the computational grid.

We support reading efile data from two different sources:

- `[model=SF]` Central California Velocity Model Etree provided by the USGS (see Figure 5.2)
- `[model=ETREEGEN]` Etree database files generated by the EtreeGen tool (developed internally at LLNL)

The Central California Velocity Model etree does not explicitly know where its geographic bounds are. Instead, this information has been hardcoded into the *cencalvm* query software library which is used by *WPP* to extract the material information. The EtreeGen generated etree files contain the spatial location and elevation ranges of the model in the header section of the Etree database file.

Internally in *WPP* we use the formula (3.4)-(3.5) to determine the geographic coordinates for the grid points where material properties are requested from the etree database. Given the latitude, longitude and depth, the *cencalvm* software library is used to query the etree database to populate the computational grid with material properties. It is important to note the bounds of the geographical region in the database. If it is queried outside that region, the *cencalvm* software returns an error code and the corresponding values are not used by *WPP*.

Assuming the computational domain is contained within the bounds of the database, it is easy to set up the material model in the input file:

Model	SF		
Description	see: http://www.sf06simulation.org/geology/velocitymodel/		
Detailed Model Coordinates (NAD27)	SE	-120.64040	37.04718
	SW	-121.91833	36.31746
	NW	-123.85736	38.42426
	NE	-122.56127	39.17461
Regional Model Coordinates (NAD27)	SE	-118.944514	36.702176
	SW	-121.930857	35.009018
	NW	-126.353173	39.680558
	NE	-123.273199	41.48486
Depth	40,000 m		

Table 5.1: Data for the central California velocity model.

```
grid x=14000.0 y=14000.0 z=3000.0 lat=38. lon=-121.5 az=135 nx=100
efile model=SF etree=USGSBayAreaVM-05.1.0.etree
```

In the case when the computational domain is larger than the region covered by the efile, a block command can be used to assign material properties to grid points outside of the efile region:

```
grid x=300000.0 y=300000.0 z=60000.0 lat=38. lon=-121.5 az=135 nx=100
block vp=8000.0 vs=4000.0 r=1000.0 rhograd=0.5
efile model=SF etree=USGSBayAreaVM-05.1.0.etree
```

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

It is also possible to only query the efile in a sub-region of the grid:

```
grid x=14000.0 y=140000.0 z=30000.0 lat=38. lon=-121.5 az=135 nx=100
block vp=8000.0 vs=4000.0 r=1000.0 rhograd=0.5
efile model=SF i1=10 i2=90 etree=USGSBayAreaVM-05.1.0.etree
```

In this case, the detailed model only spans the region between grid points 10 and 90 in the x-direction; everywhere else the material properties are taken from the block command.

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

```
efile model=SF etree=USGSBayAreaVM.etree xetree=USGSBayAreaVMExt.etree
```

5.3.1 Modeling water with Etree models

When water is present in an etree database, the efile command defaults to the `water=noshear` option, which assigns a zero shear velocity to the grid points in the water. Other water options are available for historical reasons, see Section 9.3.5. As was mentioned before, spurious shear waves in the water are suppressed using the curl-curl damping:

```
grid x=14000.0 y=140000.0 z=30000.0 lat=38. lon=-121.5 az=135 nx=100
time steps=1
block vp=8000.0 vs=4000.0 r=1000.0 rhograd=0.5
efile model=SF water=noshear query=FIXEDRES etree=USGSBayAreaVM-05.1.0.etree
damping curlcurl=0.05
```

For numerical stability reasons it is required that the water is at least one grid size deep (two grid points with water properties). If the grid size is larger than the depth of the water, *WPP* assigns water properties to the first grid point below the water surface. However, all regions of shallow water (less than 25 m deep) are modeled by the properties of the solid just below the water.

5.3.2 Evaluating Etrees near the surface

The current version of WPP enforces the free surface boundary condition on a flat surface. To approximately account for topographic effects there are two ways of assigning material properties near the free surface:

trunacte and fill Truncate mountains above a user specified elevation and fill valleys with surface properties up to that elevation. This approach can change the material properties at the free surface.

compress and expand Assign an elevation above which regions are pushed down into the domain, and below which regions are pulled up. This approach preserves the material properties in the vicinity of the free surface.

In the truncate and fill method, we provide the ability to change the offset between the z coordinate in the computational domain and the elevation in the Etree, as well as optionally filling in valleys by extrapolating the material properties at the base of the valley to the top of the domain. For example:

```
efile model=ETREEGEN etree=My.etree elev=1050 fillBasins=1
```

Here the `elev=1050` keyword is used to raise the $z = 0$ level in *WPP* to elevation 1050 meters in the Etree. It is also possible to assign a negative value to `elev` to lower the computational domain. By default *WPP* uses the material properties that are provided in the Etree, including the properties of air which would be assigned if the Etree is evaluated at an elevation above the topography. Since air has much lower density than the earth, using such a material model would lead to numerical difficulties (very large displacements in the air and large errors near the air/soil interface). To avoid having air in the material model, you can use the keyword `fillBasins=1` to extrapolate the material properties from the topographic surface in the Etree to the top of the computational domain $z = 0$. By combining the two keywords `elev=1050` and `fillBasin=1`, *WPP* ignores all material properties above 1050 meters and fills all regions between the topography and elevation 1050 meters with the surface properties in the Etree. Two consequences of the truncate and fill method are that hard rock sites can be exposed to the top of the domain (e.g.,truncated mountains exceeding the truncated elevation), and sedimentary basins can get artificially deepened leading to artificial ringing in the computed motion in those basins.

In order to improve the handing of topography, we provide an alternate method of assigning material properties near the free surface. In this approach, we modify a region near the free surface by compressing and expanding the material to match the flat free surface of the computational domain. The input line might read as follows:

```
efile model=ETREEGEN etree=My.etree depressTopo=-4000 elev=1050
```

In this example we offset the $z = 0$ boundary in *WPP* to elevation `elev=1050` meters in the Etree. We trigger the compression and expansion option by setting `depressTopo` to be the (negative) thickness of

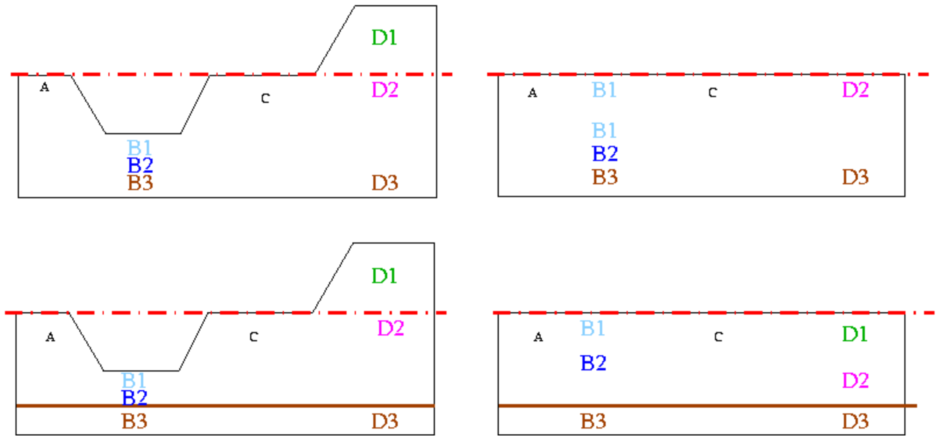


Figure 5.3: The mapping of material properties from an Etree (left) to the computational domain in *WPP* (right), using the truncate and fill approach (top) and the compress and expand technique (bottom).

the region to be compressed or expanded. The part of the Etree model with an elevation $e \leq 1050 - 4000 = -2950$ meters is mapped directly to $z = 1050 - e$ in the *WPP* computational domain. Hence, elevation $e = -2950$ corresponds to $z = 4000$ and all points below that level are mapped to *WPP* with a fixed offset of 1050 meters. Above that level, the actual region in the Etree model is compressed or expanded to be 4000 meters thick, so that the material properties at the surface in the Etree are assigned to the $z = 0$ boundary in *WPP*. A linear mapping is used to assign elevations for $0 \leq z \leq 4000$.

The two approaches are illustrated in Fig. 5.3. The original topography is shown to the left where the reference elevation is indicated with a red dash-dotted line, and the depth for the compression and expansion is shown with a brown line in the lower left figure. In the truncate and fill method, everything above the red line is ignored and the surface properties are extrapolated from the topography to the red line. In the compress and expand approach, the region between the brown line and the topography is deformed to fit between the red and brown lines in the lower right figure. Comparing the two approaches, we see that light blue material `B1` is added on top of the original blue materials `B1` and `B2` in the truncate and fill approach, which also ignores the green material `D1`. In contrast, the compress and expand technique stretches out the original blue materials and pushes down the topography on the right to make it fit in the computational domain.

To illustrate the difference in simulated results, we compare the truncate and fill approach to the compress and expand technique for the Barnwell event, see Fig 5.4. An example of the problem setup script is as follows:

```
# barnwell-125m resolution
grid x=180000 y=170000 z=50000 h=125 lat=37.38 lon=-116.9 az=90
time t=200
source type=Gaussian t0=25.0 lat=37.231 lon=-116.410 \
       depth=601 m0=1e15 freq=1.0 Mxx=1.0 Myy=1.0 Mzz=1.0
efile model=ETREEGEN etree=nts.etree depressTopo=-4000 elev=+1050
sac lat=36.264167 lon=-115.309722 depth=0.0 file=ANNR
sac lat=36.111389 lon=-115.049722 depth=0.0 file=EFLA
```

For sites with less path effects (closer to the source) and minimal alluvium basin deposits, the difference between the fill and truncate approach and the compress and expand technique is small (see Fig 5.5). However, for sites situated in basins which get artificially deepened by the fill and truncate method, we observe ex-

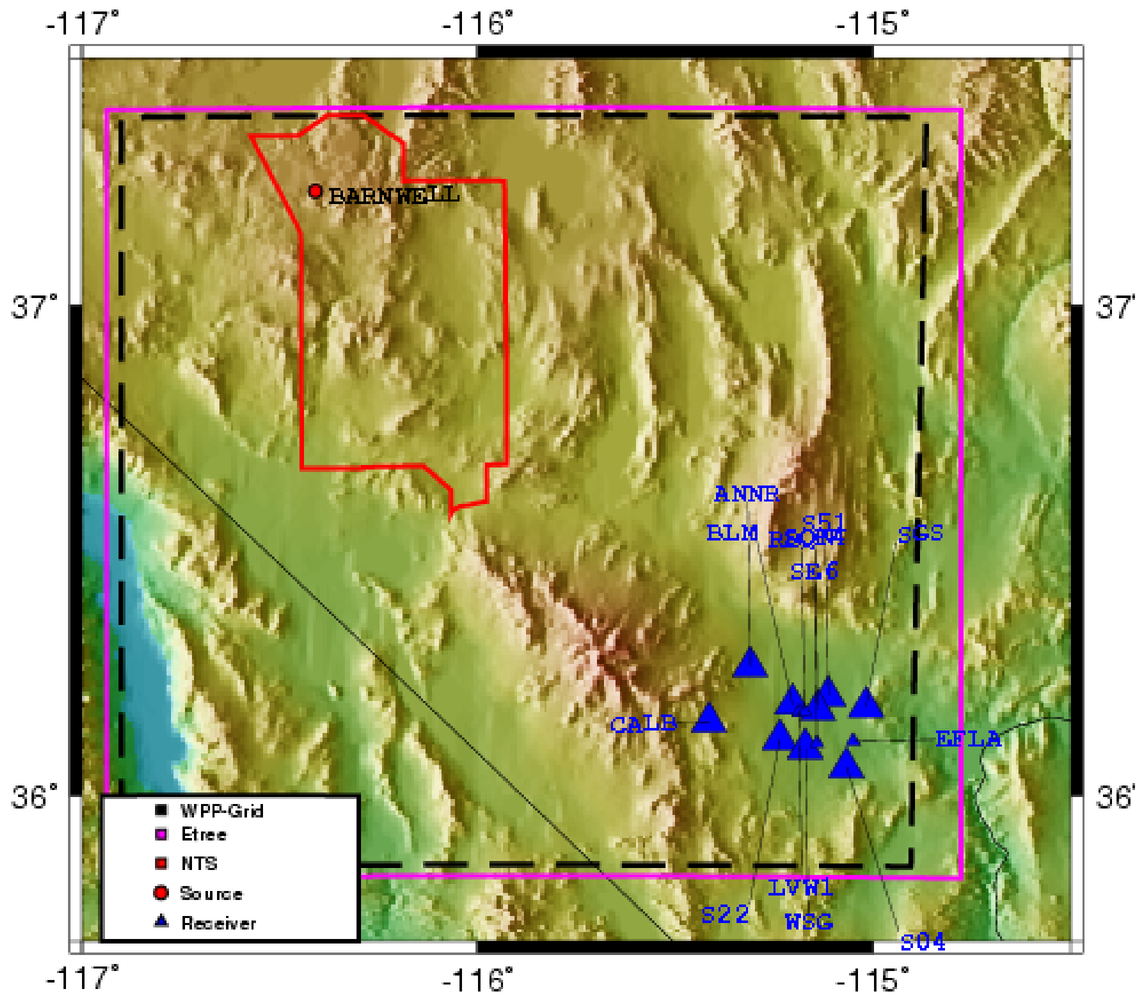


Figure 5.4: Location of the source and stations for the Barnwell simulation. This figure was generated using the GMT command, see Section 9.5.1 for details.

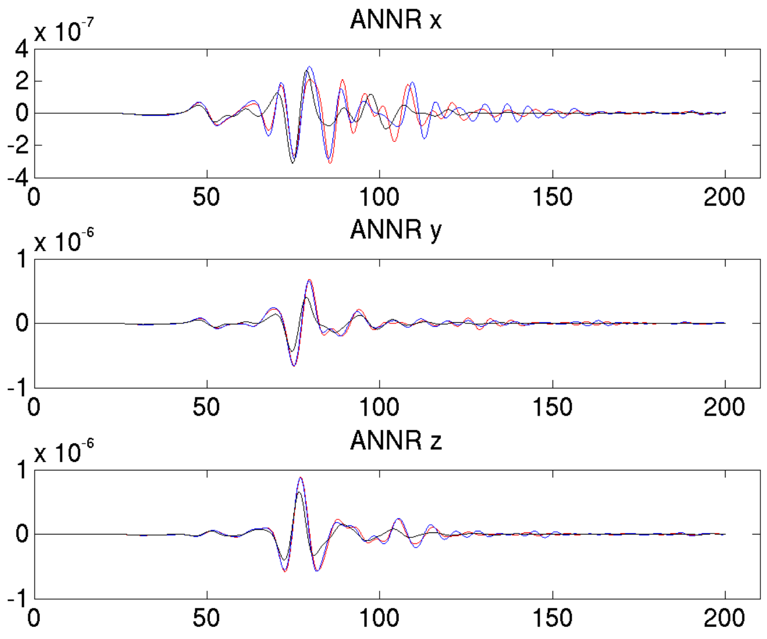


Figure 5.5: Site response at station ANNR using a 1-D plane layered material model (black), a 3-D material model with the fill and truncate approach (blue), and the same 3-D material model with the compress and expand technique (red). Displacements in meters versus time in seconds.

cessive artificial ringing at later times. This ringing does not occur with the compress and expand technique (see Fig 5.6).

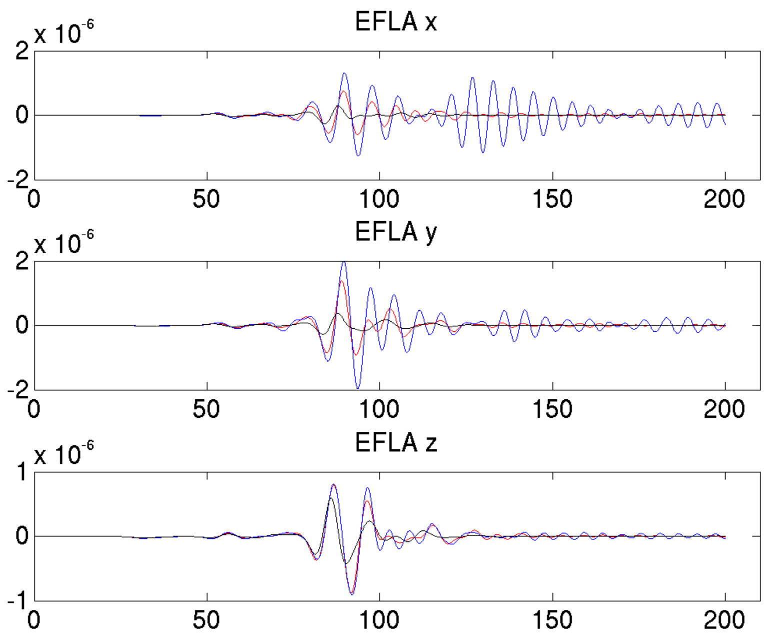


Figure 5.6: Site response at station EFLA using a 1-D plane layered material model (black), a 3-D material model with the fill and truncate approach (blue), and the same 3-D material model with the compress and expand technique (red). Displacements in meters versus time in seconds.

5.4 The pfile command

The pfile command can be used to assign material properties based on a lattice of depth profiles (pfile). A pfile contains the values of the model features (P-velocity, S-velocity, density, and Q-factors) as function of depth at points on an equally spaced latitude-longitude lattice. 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. Furthermore, layers which only occur in a subset of the profiles can be tapered to 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 *WPP* and is often much coarser. The material properties at the grid points of the computational mesh in *WPP* are assigned values using Gaussian averaging between the nearest $P \times P$ 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 profile and then averaged in the latitude-longitude plane. The number of points in the Guassian averaging, P, is assigned by the user in the `pfile` command. For example, the following line in the input file makes *WPP* 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. `smoothingsize=4` means that $P = 4$ in the Gaussian averaging. A larger value of P (≥ 5) is particularily useful to avoid staircasing imprints when the computational grid is much

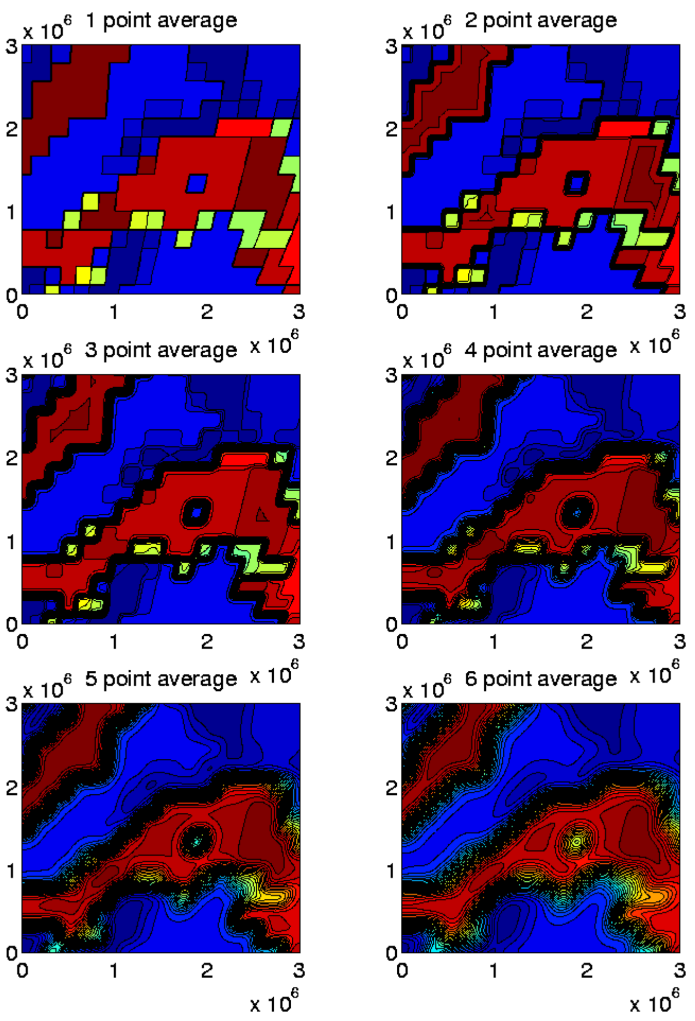


Figure 5.7: The `smoothingsize` parameter can be used to average out abrupt variations in the horizontal plane (constant depth) in a coarse pfile material model.

finer than the pfile lattice, see Figure 5.7. `smoothingsize` can be set to any number greater than or equal to one.

When P 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, etc.) is assigned by the formula

$$c(\phi,\theta) = \frac{\sum_{m=i-Q}^{i+Q} \sum_{n=j-Q}^{j+Q} c_{m,n}\omega_{m,n}}{\sum_{m=i-Q}^{i+Q} \sum_{n=j-Q}^{j+Q} \omega_{m,n}}, \quad Q = \frac{P-1}{2}, \tag{5.1}$$

where the weights are given by

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

and the grid size in the latitude-longitude lattice is Δ_{lat}. This choice of α makes the weights $\omega_{m,n} < 10^{-6}$ for points that are further from (ϕ_m, θ_n) than $P\Delta_{lat}/2$, which justifies the truncation of the series in (5.1). A similar procedure is used for even values of P, but in this case the averaging formula (5.1) is centered around the nearest cell center on the latitude-longitude lattice.

The pfiles are in ASCII text format, and consist of a header followed by data. The header has 7 lines and follows the following format:

Line	Column 1	Column 2	Column 3	Column 4
1	Name (string)			
2	Δ_{lat} [deg] (real)			
3	N_{lat} (integer)	θ_{min} [deg] (real)	θ_{max} [deg] (real)	
4	N_{lon} (integer)	ϕ_{min} [deg] (real)	ϕ_{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)			

Lines 3 and 4 contain the number of lattice points in the latitude and longitude direction, respectively, as well as the starting and ending angles. Line 5 contains the number of depth values in each profile, followed by the minimum and maximum depth measured in km. Line 6 supplies information about the location of the discontinuities in each depth profile. 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 if the subsequent data contains quality factors (Q_P and Q_S); otherwise it should contain the single letter F.

The header is directly followed by $N_{lat} \times N_{lon}$ depth profiles, according to the pseudo-code

```
for $i = 0, 1, \ldots, N_{lat} - 1$
        for $j = 0, 1, \ldots, N_{lon} - 1$
                $\theta_i = \theta_{min} + i\Delta_{lat}$;
                $\phi_j = \phi_{min} + j\Delta_{lat}$;
                (save depth profile for $\theta_i$, $\phi_j$)
        end
end
```

The first line of each depth profile contains its latitude and longitude (in degrees as real numbers). The subsequent N_{dep} lines have the following format:

Index (integer)	depth [km] (real)	V_p [km/s] (real)	V_s [km/s] (real)	ρ [g/cm^3] (real)	Q_P (real)	Q_S (real)

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.

Chapter 6

Modeling dissipative materials (attenuation)

Attenuation models are obtained by adding viscoelastic components to the stress tensor. Such viscoelastic behavior is especially significant in sedimentary materials near the surface, whereas viscoelastic effects in solid rock are smaller. Viscoelastic models describe the stress tensor as frequency dependent in Fourier space. The frequency dependency is usually modeled by a rational expression, which leads to a number of extra differential equations when transformed to the physical space. The attenuation models damps the waves.

The attenuation properties of the material are described by the quality factors Q_p and Q_s where Q_p models the P-wave attenuation and Q_s models the S-wave attenuation. The quality factor is large for small attenuation and small for large attenuation. The equations without attenuation are obtained as the limit $Q_p, Q_s \to \infty$.

WPP uses the attenuation model described in [6]. The model of the viscoelastic modulus has 8 terms leading to 24 additional variables per grid point. Therefore, running *WPP* with attenuation doubles the memory requirement from the purely elastic case. There are also 24 additional partial differential equations which need to be solved leading to about ten times more arithmetic operations compared to the purely elastic case. However, *WPP* simulations with attenuation only seem to require about three times more CPU time.

The attenuation model is designed for the interval $5 \leq Q \leq 5000$, if Q_p or Q_s are outside this interval, the computed results might not be reliable.

To activate the attenuation model, place the line

```
attenuation
```

in the input file. Once the attenuation is activated, the quality factors are given in the same way as the other material properties, i.e., through the `block` command, the `vfile` command, or the `efile` command. For example,

```
block vs=3464 vp=6000 r=2700 z1=1000 qs=69.3 qp=155.9
```

sets $Q_s = 69.3$ and $Q_p = 155.9$ in the entire computational domain. The command

```
block x1=0 x2=1000 y1=0 y2=4000 z1=0 z2=1500 qs=40.3
```

sets $Q_s = 40.3$ in the domain $0 \leq x \leq 1000$, $0 \leq y \leq 4000$, $0 \leq z \leq 1500$. The vfile command

```
vfile type=qp file=qp.flt
```

reads Q_p from a file named `qp.flt`. No extra option is needed for the `efile` command, because if the etree database contains the quality factors, *WPP* will read them automatically when attenuation is activated.

In the attenuation model, the material velocities V_p and V_s are frequency dependent. Therefore, *WPP* needs to know at which frequency the input material velocities were measured. The attenuation command takes this frequency as a parameter. The command

```
attenuation frequency=2.5
```

means that the material velocities V_p and V_s given in the input script were measured at the frequency 2.5 Hz. Only one frequency, valid for all material velocities in the input script, can be specified. The default value of `frequency` is 1 Hz, which according to [6] is a reasonable value to use when the frequency is not known.

Chapter 7

Mesh refinement

The mesh refinement command in *WPP* enables the user to locally refine the computational mesh in areas where finer resolution is needed. The resolution of the solution is often charaterized in terms of the number of grid points per wave length. For a signal of frequency f, the wave length is V_s/f, where V_s is the wave speed in the material. Hence the grid size needs to be small where the wave speed is small, and vice versa. In order to maintain a constant resolution in terms of the number of grid points per wavelength for a given frequency (see Equation (4.1)), the grid size should ideally be adjusted such that ratio V_s/h becomes constant over the computational domain, that is, the grid spacing h should be made proportional to the local value of V_s. However, significant savings in computational effort can still be realized by limiting the variations in this ratio to two or maybe three over the computational domain, which is a more realistic goal for the strongly heterogeneous materials that often occur in seismic applications.

When using mesh refinement, the extent of the computational domain is determined by the base grid which is specified by the grid command, and the mesh sizes in the refinement patches are specified relative to the spacing in the base grid. The command

```
refinement x1=5000 x2=15000 y1=1000 y2=7000 z1=0 z2=2500 rr=4
```

builds a refinement patch that covers the region $5000 \leq x \leq 15000$, $1000 \leq y \leq 7000$, and $0 \leq z \leq 2500$ with refinement ratio four, i.e., the grid spacing in the refinement patch will be four times smaller than in the base grid. Note that the refinement ratio must be a positive integer power of two. Figure 7.1 shows a slice through the grid at $x = 7000$ with the above refinement when the base grid was specified by

```
grid x=30000 y=30000 z=10000 h=1000
```

Unspecified bounds are interpreted as the whole domain. A refinement given by

```
refinement z1=0 z2=3500 rr=2
```

is limited in the z-direction but extends over the entire domain (as specified by the grid command) in the x- and y-directions.

Refinement grids are aligned in the sense that every `rr`:th point coincides with a grid point in the base grid. However, refinement patches don't necessarily have to start or end on a base grid point. For example, if the refinement ratio equals two, the refined grid can either start/end on a base grid point, or half-way in between two points in the base grid. If the refinement ratio is four, the refined mesh can start/end on a base grid point or be offset by quarters of the base grid size, and so on for higher refinement ratios. When x, y, or z coordinates specify the extent of a refinement patch, *WPP* rounds the specified coordinates for the start/end locations such that the specified region always gets covered by the refined grid. For example, if the

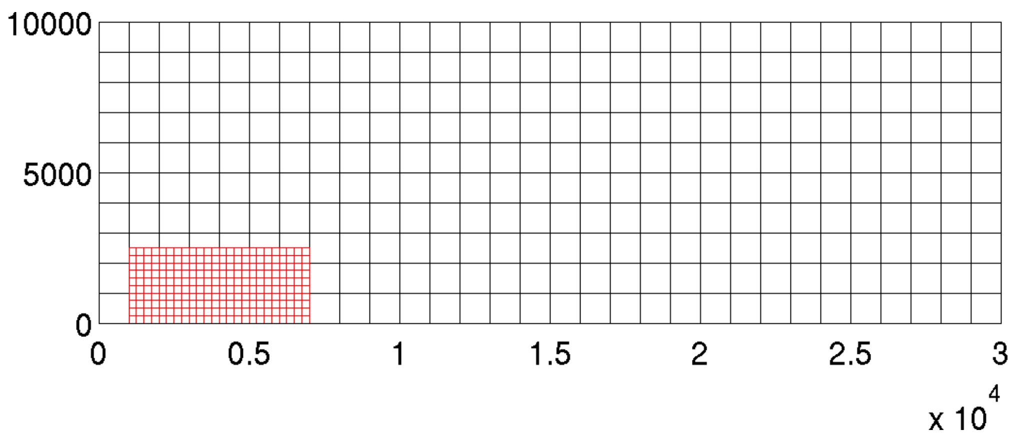


Figure 7.1: Base grid with one refinement patch.

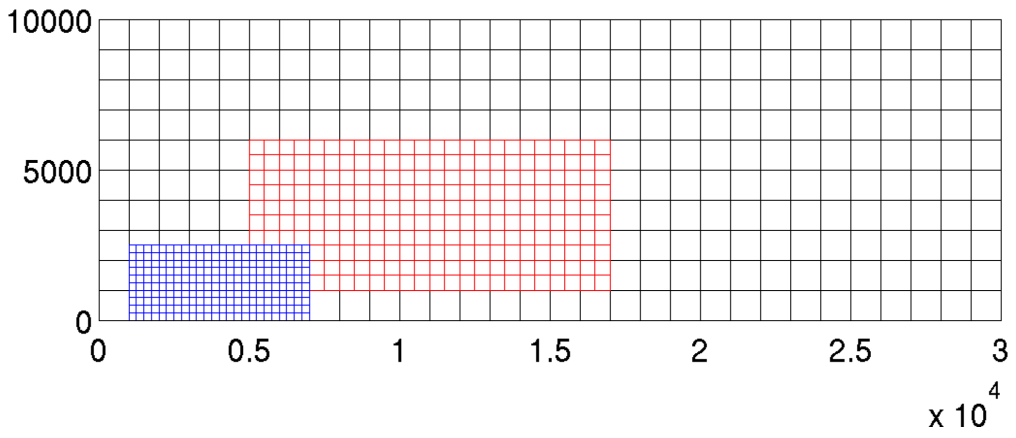


Figure 7.2: Base grid with two refinement patches.

grid points in the base grid have coordinates x_i, the refinement ratio is two, and the start location `x1`$= x_1$ satisfies $x_i + h/2 < x_1 < x_{i+1}$, the refinement patch will start at $x_i + h/2$.

There are no restrictions of how different refinement grids can overlap. The two refinement commands

```
refinement x1=5000 x2=15000 y1=1000 y2=7000 z1=0 z2=2500 rr=4
refinement x1=5000 x2=15000 y1=5000 y2=17000 z1=1000 z2=6000 rr=2
```

give the grid shown in Figure 7.2, where an slice through the grid at $x = 7000$ is plotted. It is instructive to examine the individual patches in the grid, see Figure 7.3. Note that there are some holes in the coarser grids. The grid points inside the holes have been removed because they are not used during the simulation. Grid points on the fringe of the hole are called interpolation points since they get their value by interpolation from another grid, so only the grid points away from the hole boundary are used to solve the elastic wave equation. When interpolation points are located in between grid points in the other grid, a fourth order accurate interpolation formula is used to update these points. This means that two refinement grid patches need to overlap by at least the grid size of the coarser of the two patches. Otherwise, an underlying coarser grid will be used to transfer solution values between the refinement patches, which can severely reduce the accuracy of the solution.

It is also possible to give the bounds on the refinement grids in terms of the indices of the base grid. For example,

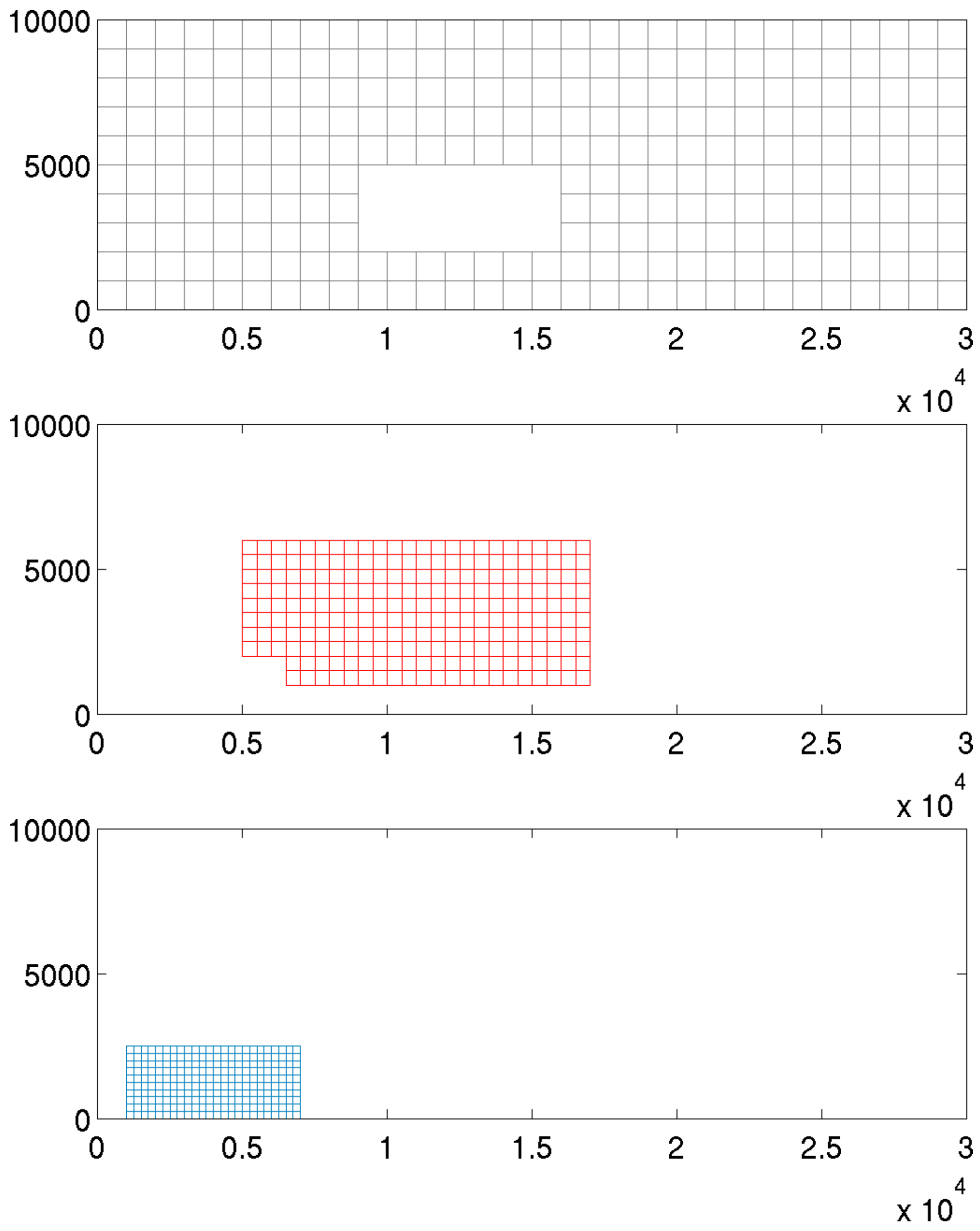


Figure 7.3: The three individual grid patches in Figure 7.2.

```
refinement i1=5 i2=50 j1=1 j2=7 k1=1 k2=15 rr=4
```

specifies a refinement grid over the region $5 \le i \le 50$, $1 \le j \le 7$, and $1 \le k \le 15$ in the base grid indices, with a refinement ratio of 4. Note that this approach is limited to specifying refinements which start and end at grid points in the base grid.

Refinement patches should overlap! Note that the first and last grid points in each direction of a mesh refinement patch are used to interpolate values from another grid, unless the patch is aligned with the boundary of the computational domain. Hence, it is important that successive mesh refinement patches are made to overlap because otherwise the underlying coarser mesh will be used to interpolate solution values which can degrade the accuracy of the solution. An example without overlap is shown in Figure 7.4, where we show a one dimensional cut along the z-axis of the active grid points in three different grids. The grid and refinement commands used to create these grids were

```
grid nx=30 x=1.0 y=1.0 z=1.0
refinement z1=0 z2=0.1725 rr=4
refinement z1=0.1725 z2=0.6 rr=2
```

The finest (red) grid ends at the same point ($z = 0.1725$) as where the intermediate grid (green) starts. There is no overlap between these two grids, and therefore some points from the coarsest grid (blue) are needed to define the interpolation. If we instead let the two refinement patches overlap,

```
grid nx=30 x=1.0 y=1.0 z=1.0
refinement z1=0 z2=0.1725 rr=4
refinement z1=0.1553 z2=0.6 rr=2
```

we obtain the grid hierarchy displayed in Figure 7.5. Here the overlap between the two grids is approximately equal to the spacing of the intermediate (green) grid. There is enough overlap to let the two grids exchange values without using the coarsest (blue) grid. Therefore, the accuracy will be better. **In general, if two grids meet they should be given an overlap which is equal to the spacing of the coarsest of the two grids.**

The `image` command saves the solution on the base grid as well as all refined meshes that intersect the given image plane. We recommend saving the mask array together with any solution to aid in postprocessing the results, see Section 9.5.4. Receivers defined by the `sac` command that are positioned in a location covered by a fine grid are moved to the finest grid.

7.1 Artificial dissipation

At mesh refinement boundaries, values are transfered between the grids by fourth order accurate interpolation. These interpolation boundary conditions at the mesh refinement boundaries make the non-dissipative numerical scheme in *WPP* unstable. Artificial dissipation (damping) can be used to suppress this instability. The artificial dissipation operator implemented in *WPP* is called D_4 dissipation. D_4 dissipation adds a discretized version of

$$-d_4 h^3 \left(\frac{\partial}{\partial x^2}(\rho \mathbf{u}_{txx}) + \frac{\partial}{\partial y^2}(\rho \mathbf{u}_{tyy}) + \frac{\partial}{\partial z^2}(\rho \mathbf{u}_{tzz}) \right),$$

to the right hand side of the elastic wave equation, where $\mathbf{u}_t$ is the time derivative of the displacement vector. The factor h^3 shows that the dissipation operator only has a third order effect on the total truncation error, i.e., the D_4 dissipation is one order smaller (in h) than the truncation errors of the numerical scheme. Hence,

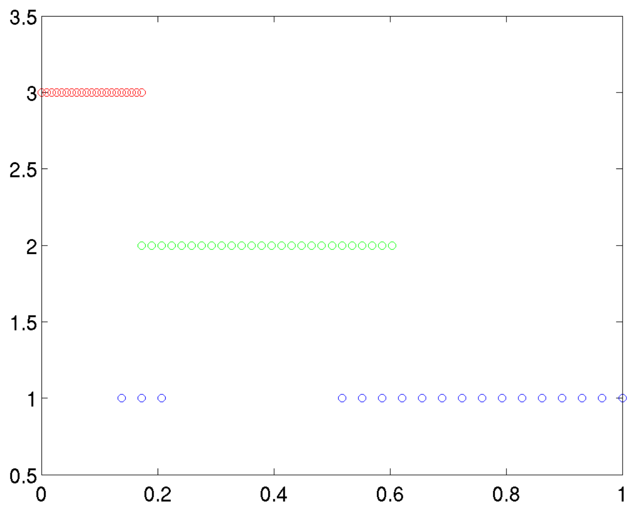


Figure 7.4: Active grid points in three grid hierarchy. No overlap between the finest (red) and intermediate (green) grids.

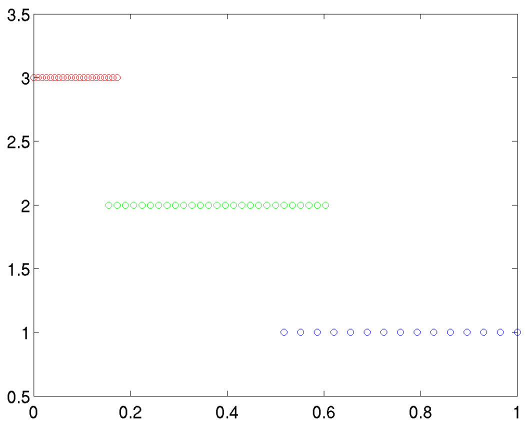


Figure 7.5: Active grid points in three grid hierarchy. Overlap between the finest (red) and intermediate (green) grids is equal to the spacing of the intermediate grid.

the D_4 dissipation does not degrade the formal order of accuracy of the method. The D_4 dissipation has a tunable parameter d_4. Numerical experiments indicate that $d_4 = 0.2$ works well in practice.

The purpose of the D_4 damping is to control numerical instabilities due to interpolation errors in between mesh refinement patches, and is therefore not necessary if the calculation is performed on a singe mesh. When mesh refinement is used, *WPP* automatically activates the D_4 dissipation with the default strength that works well in most cases. The `damping` command can be used to change this value, but most users should never need to change it. If a single grid is used, the D_4 dissipation is by default not activated.

7.2 Sources and mesh refinement

The standard *WPP* discretization of the source term breaks down if the source is located at an interface between two grids. Therefore *WPP* uses a special source discretization near grid interfaces. This special source discretization also gives second order accuracy, but has a slightly larger error constant than the source discretization on a single mesh. It is therefore advisable to avoid placing mesh refinement boundaries near sources, if possible.

Chapter 8

Output options

8.1 Setting the output directory

The fileio command can be used to specify a directory where *WPP* writes all its output, as well as a name to prepend to the output files. If the directory does not exist, *WPP* 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,

```
fileio path=wpp_dir verbose=0 printcycle=10
```

causes all output to be written to the directory "./wpp_dir", turns off extra diagnostic messages, and prints the time step information every 10 time steps.

8.2 Generating a bird's eye view of the problem domain: the gmt command

The Generic Mapping Toolkit (*GMT*) [11] is a popular postscript image generation toolkit for geophysical applications which can be used to make plots like Figure 5.4. 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 file which is output from *WPP* is a UNIX C-shell script, which often needs to be fine-tuned to suit the needs of a particular application. To have *WPP* write a *GMT* file, you can say

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

8.3 Time-history at a point: the sac command

The Seismic Analysis Code (*SAC[3]*) is a widely used program for displaying and analyzing digital seismograms (time-series). Because of the broad use of *SAC*, the file format is commonly used to exchange digital seismogram data. *WPP* can save the time-history of its solution at one point in the computational domain using the command:

```
sac x=100000 y=50000 z=0 file=s1
```

This command makes *WPP* sample the displacement time-history at the grid point closest to the location x=100000 m (100 km), y=50000 m (50 km), z=0. *WPP* saves the data to the files named:

- s1.I.J.K.x

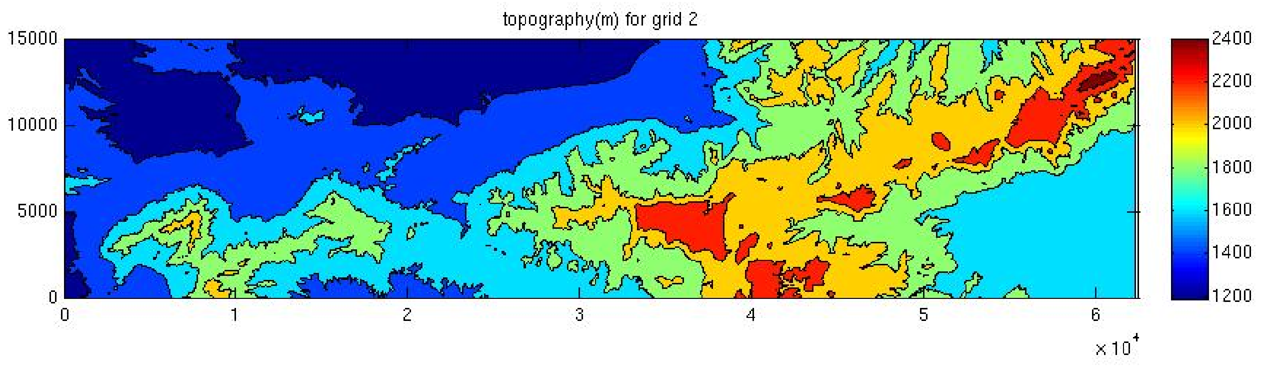


Figure 8.1: A sample 2D image slice looking at surface topography near the Nevada Test Site

- s1.I.J.K.y
- s1.I.J.K.z (note: positive displacement values correspond to downward motion)

Here I, J and K are the indices of the grid point closest to the specified (x, y, z) coordinate. The x,y,z files correspond to the displacement history in the corresponding coordinate direction.

SAC files contain date information which can be assigned using the date option:

```
sac i=10 j=10 k=5 eventDate=2003/11/22 eventTime=16:17:00 sta=EKM
```

Here the date is set to November 22nd, 2003, at 4:17 PM. By default the date is set to the date and time at the start of the simulation. In this example we also set the station name to ”EKM”.

The sample option can be used to only save the displacements every other time step

```
sac i=10 j=10 k=5 sample=2 writeEvery=100
```

The sample option refers to how often the *SAC* file will record a displacement, in this case every 2 time steps. We also set the writeEvery option to 100, which means that the *SAC* files are flushed to disk after every 100 time steps. By default, the files are flushed to disk every 1000 time steps, and at the end of the simulation.

8.4 2-D cross-sectional data: the image command

The image command may be used to save files holding 2D horizontal or vertical cross-sectional data for many different variables, as an example rendered in MATLAB see Fig. 8.1. This command is useful for visualizing the solution, making movies, and checking the material properties. The example

```
image mode=ux j=5 file=picturefile cycle=1
```

outputs the x-displacement component (`mode=ux`) on the xz-plane along the fifth grid point (`j=5`) in the y-direction. The data is written to a file named `picturefile.cycle=1.j=5.ux` after the first time step (`cycle=1`). The example

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

outputs the divergence of the displacement 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
picturefile.cycle=200.x=1000.div
...
```

With this setup, one image file is output every 100 time steps.

WPP can output twenty different quantities, see Section 9.5.4. The image plane can be specified as a grid point index or as a Cartesian coordinate (x, y, or z). In the latter case, the image data is taken from the grid plane closest to the specified coordinate. The image can be written at a specific time step or a specified time. Images can also be output at time step intervals (as `cycleInterval` in the example above) or at time intervals.

When mesh refinement is used, the image plane might intersect more than one grid. If this occurs, all grid patches that intersect the image plane are stored on file. It is possible that some of the stored grid patches are covered by finer grids and therefore consist entirely of points that are inactive. It is the responsability of the user to plot whatever grid information is relevant from the output image file. To help with the plotting, it can be useful to output an image with `mode=mask`. The mask has the value 1 at active (used) points, and 0 at unused points.

The images files are written in a binary format. *WPP* stores the grid size h (a double precision floating point number) and the index bounds of the image (four integers) for each patch first in the file. Then follows the image patches as single or double precision floating point numbers stored in column order. The exact format follows from the Matlab function

```
% Returns image patch nr. 'inr' on file 'fil' in 'im',
% corresponding grid returned in 'x' and 'y'
function [im,x,y]=readimagepatch( fil, inr )

fd=fopen(fil,'r');

% Precision of image data (4-float, 8-double)
pr=fread(fd,1,'int');

% Number of image patches on file
ni=fread(fd,1,'int');
if inr > ni
   disp( 'Error image nr too large');
else
% For each patch read grid spacing and index bounds.
% For patch nr. p:  ib(p) <= i <= ie(p) and jb(p) <= j <= je(p)
   for i=1:ni
      h(i)  = fread(fd,1,'double');
      ib(i) = fread(fd,1,'int');
      ie(i) = fread(fd,1,'int');
      jb(i) = fread(fd,1,'int');
      je(i) = fread(fd,1,'int');
   end;
% Want patch nr. inr, skip the first inr-1 image patches.
   for i=1:inr-1
      fseek(fd,(ie(i)-ib(i)+1)*(je(i)-jb(i)+1)*pr,0);
   end;
% Read wanted image patch, single or double precision.
```

```
   if pr == 4
      im = fread(fd,[ie(inr)-ib(inr)+1 je(inr)-jb(inr)+1],'float');
   else
      im = fread(fd,[ie(inr)-ib(inr)+1 je(inr)-jb(inr)+1],'double');
   end;
% Corresponding Cartesian grid
   x  = ((ib(inr):ie(inr))-1)*h(inr);
   y  = ((jb(inr):je(inr))-1)*h(inr);
   fclose(fd);
end;
```

This function reads one image patch from an image file into the Matlab matrix `im`. The corresponding Cartesian coordinates are returned in the Matlab vectors `x` and `y`. Here, `fd` is a file descriptor variable. The Matlab functions `fopen` and `fread` perform binary I/O similarly to the C functions with the same names. This function is provided in the tools/readimagepatch.m file in the source distribution of *WPP*.

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

The options hvelmax, vvelmax, haccmax, and vaccmax, compute the maximum of the respective quantity over time at each point in the image plane. The horizontal velocity is $\sqrt{u_t^2 + v_t^2}$ where u and v are the displacement components in the x- and y-directions, respectively, The vertical velocity is $|w_t|$, where w is the displacement component in the z-direction. For these modes, the image cycle option only determines how often the maxima are written to image files; the actual computation and accumulation of the maximuma are performed after each time step.

8.5 Restart files

Warning: Restart files can be quite large. Aside from header information, a restart file contains six double precision numbers per grid point. For a moderately sized simulation with 80 million grid points, the size of the restart file exceeds 3.84 GB.

WPP can save the current state of a simulation in a restart file and initialize a simulation from an existing restart file. This functionality can be useful during longer simulations to protect against data loss in case of hardware failures. To request a restart file to be written, use the restart command line:

```
restart dumpInterval=1000 file=myRestart
```

This command generates a restart file every 1000 time steps with the perpended name myRestart. After time step 3000, three restart files are available:

```
myRestart_step001000.rst
myRestart_step002000.rst
myRestart_step003000.rst
```

To initialize a simulation from an existing restart file, use the command line

```
restart fromCycle=2000 file=myRestart
```

This time, we use the fromCycle keyword to specify the time step. In this case *WPP* searches the working directory (as specified by the fileio command) for the restart file myRestart_step002000.rst.

WPP requires that material properties be setup using the same command lines on both the initial and restarted runs. It also requires the grid to be the same. Some header information is stored in the restart file to ensure basic compatibility. Changing the input file between the initial and restarted runs may result in unexpected results.

SAC file data is also restarted. When a restart file is written, the relevant *SAC* header information and displacement history is saved in the restart file. Upon restart, all *SAC* options specified in the *WPP* input file for the restarted run are overridden by the *SAC* information contained in the restart file. Hence, it is not possible to add or delete sampling stations in restarted runs.

Chapter 9

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 ... parameterN=valueN
...
```

Each command starts at the beginning of the line and ends at the end of the same line. Blank lines are disregarded and comments can be placed on lines starting with a # character. The order of the parameters within each command is arbitrary. The material commands (block, vfile, pfile, and efile) are applied in the order they appear, but the ordering of all other commands is inconsequential. Also note that the entire input file is read before the simulation starts.

9.1 Specifying the grid parameters (grid) [required]

Required parameters:

- number of grid points in all three dimensions and the grid size, or
- spatial extents in all three dimensions and the grid size, or
- spatial extents in all three dimensions and the number of grid points in one direction.

grid options				
Option	**Description**	**Type**	**Units**	**Default**
x	physical dimension of grid in the x-direction	float	m	none
y	physical dimension of grid in the y-direction	float	m	none
z	physical dimension of grid in the z-direction	float	m	none
h	grid spacing	float	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
az	clockwise angle from north to the x-axis	float	degrees	135.0
lat	latitude geographic coordinate of the origin	float	degrees	37.0
lon	longitude geographic coordinate of the origin	float	degrees	-118.0

9.2 Specifying the time parameters (time) [required]

The time command line specifies the duration of the simulation in seconds or the number of time-steps. The size of the time step is computed internally by *WPP*.
Required parameters:

- Either t or steps must be specified.

time options				
Option	**Description**	**Type**	**Units**	**Default**
t	duration of simulation	float	s	none
steps	number of cycles (time-steps) to advance	int	none	none

9.3 Specifying the material model [required]

9.3.1 Enabling viscoelastic modeling (attenuation)

Required parameters: none

attenuation options				
Option	**Description**	**Type**	**Units**	**Default**
frequency	frequency at which material velocities were measured	float	Hz	1.0

The attenuation model uses the quality factors Q_p and Q_s. These factors are given as a part of the material model, but are only taken into account if the `attenuation` command is present in the input file.

9.3.2 specifying a box shaped sub-region

A sub-region can be used in conjunction with the block, vfile and efile commands to signal that a command should only apply to parts of the computational domain. A box shaped sub-region can be specified using either i,j,k grid points or x,y,z domains, see Table 9.1. Note that the i,j,k indices refer to the base grid when mesh refinement is used.

box shaped sub-region options				
Option	**Description**	**Type**	**Units**	**Default**
x1	minimum x-dim for the box shaped sub-region	float	m	0.0
x2	maximum x-dim for the box shaped sub-region	float	m	max x
x	sets x1 and x2 to the same value, corresponds to yz plane	float	m	none
y1	minimum y-dim for the box shaped sub-region	float	m	0.0
y2	maximum y-dim for the box shaped sub-region	float	m	max y
y	sets y1 and y2 to the same value, corresponds to xz plane	float	m	none
z1	minimum z-dim for the box shaped sub-region	float	m	0.0
z2	maximum z-dim for the box shaped sub-region	float	m	max z
z	sets z1 and z2 to the same value, corresponds to xy plane	float	m	none
i1	minimum index in x-dim for the box shaped sub-region	int	none	1
i2	maximum index in x-dim for the box shaped sub-region	int	none	Nx
i	sets i1 and i2 to the same value	int	none	none
j1	minimum index in y-dim for the box shaped sub-region	int	none	1
j2	maximum index in y-dim for the box shaped sub-region	int	none	Ny
j	sets j1 and j2 to the same value	int	none	none
k1	minimum index in z-dim for the box shaped sub-region	int	none	1
k2	maximum index in z-dim for the box shaped sub-region	int	none	Nz
k	sets k1 and k2 to the same value	int	none	none

Table 9.1: A box shaped sub-region can be specified on the block, vfile, and efile command lines.

9.3.3 block command (block)

Required parameters: none

The commands below can be used with or without the box shaped sub-region options, depending on whether the block spans the entire grid or just a subvolume.

block options			
Option	**Description**	**Type**	**Units**
vp	P-wave velocity	float	m/s
vs	S-wave velocity	float	m/s
r	density	float	kg/m^3
ps	p/s ratio	float	none
vpgrad	vertical gradient for vp	float	s^{-1}
vsgrad	vertical gradient for vs	float	s^{-1}
rhograd	vertical gradient for rho	float	kg/m^4
qp	P-wave quality factor	float	none
qs	S-wave quality factor	float	none

The gradient options `vpgrad`, `vsgrad`, and `rhograd` are ways to specify linear profiles in the z-direction (downward). Note that the unit for `vpgrad` and `vsgrad` is 1/seconds, which can be interpreted as m/s per m, or km/s per km.

9.3.4 velocity file (vfile)

Required parameters:

- type of the input file (vp, vs, r, qp, qs)
- name of the file to be read

The vfile command options may be used in conjunction with the (i,j,k), but not the (x,y,z), style box shaped sub-region options (see Table 9.1).

vfile options			
Option	**Description**	**Type**	**Units**
type	input file type (vp, vs, qp, qs, or r)	string	none
file	name of input vfile	string	none
scaling	factor multiplying vfile data when read	float	none
float	machine type where the vfile was written	string	none

Values for the float option can be either `linux`, `sun`, or `native`. The default is `native`.

9.3.5 etree database files (efile)

Required parameters:

- model: name of the model to initialize materials from
- etree: full path to the etree database

These command options may be used in conjunction with the box shaped sub-region options (see Table in 9.1).

efile options				
Option	**Description**	**Type**	**Units**	**Default**
model	name of the model to be read in	string	none	none
logfile	name of file where output from etree file read will go	string	none	none
vsmin	minimum threshold for the s velocity in solids	float	m/s	none
vpmin	minimum threshold for the p velocity in solids	float	m/s	none
query	type of query to perform	string	none	MAXRES
resolution	for FIXEDRES, the resolution to sample the data	float	m	h
water	set to noshear, poisson, or seafloor	string	none	noshear
elev	max etree elevation to map WPP's z=0 plane	float	m	0.0
fillBasins	set to 1 or 0	int	none	0
depressTopo	negative thickness of deformed region	float	m	none
etree	full path to the etree database	string	none	none
xetree	full path to the extended etree database	string	none	none

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

Query Option	Description
MAXRES	This will sample the data at the maximum resolution available in the database, which is the default query type for the efile option.
FIXEDRES	This will sample the database at the requested resolution, even if the database contains values at a higher resolution. This option defaults to the grid spacing h, or can be specified with the resolution option.

For example, to set the data to be fixed at a 1km sampling:

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

Note: If you would like to find out the locations of grid points which are found to be outside the etree database domain, the logfile option can be used to track which points were not found. It will report points it found outside the domain, as well as points it did not have data for (i.e., air just above water or a material).

9.3.6 pfile (pfile)

Required parameters:

- name of the file to be read

pfile options				
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	int	none	5
vpmin	minimum threshold value for V_P	float	m/s	0
vsmin	minimum threshold value for V_S	float	m/s	0
rhomin	minimum threshold value for density	float	m/s	0
flatten	??? (T or F)	string	none	F

9.4 Specifying the source (source)

The source location can be specified in terms of Cartesian coordinates (x, y, z), grid indices (i, j, k), or geographic coordinates (lon,lat,depth). There can be multiple source terms where each source can have different spatial location and be centered at different time levels.
Required parameters:

- Location of the source specified by Cartesian location, grid indices, or geographical coordinates
- Type of the source which can be specified in two ways:
  1. Specifying a point force by the amplitude, F_0, and at least one component of the force vector (F_x, F_y, F_z)
  2. Specifying a point moment tensor, which there are two ways:
     (a) Seismic moment, M_0, and at least one component of the moment tensor (M_{xx}, M_{xy}, etc)
     (b) Seismic moment, M_0, and double couple focal mechanism, strike/dip/rake angles (see [1]).

source options				
Option	**Description**	**Type**	**Units**	**Default**
x	x position of the source	float	m	none
y	y position of the source	float	m	none
z	z position of the source	float	m	none
i	x grid point of the source	int	none	none
j	y grid point of the source	int	none	none
k	z grid point of the source	int	none	none
depth	depth of the source	double	m	none
lat	latitude geographic coordinate of the source	double	degrees	none
lon	longitude geographic coordinate of the source	double	degrees	none
m0	moment amplitude	float	Newton-meter	1.0
Mxx	xx-component of the moment tensor	float	none	0.0
Myy	yy-component of the moment tensor	float	none	0.0
Mzz	zz-component of the moment tensor	float	none	0.0
Mxy	xy-component of the moment tensor	float	none	0.0
Mxz	xz-component of the moment tensor	float	none	0.0
Myz	yz-component of the moment tensor	float	none	0.0
f0	point force amplitude	float	Newton	1.0
Fx	forcing function in the x direction	float	none	0.0
Fy	forcing function in the y direction	float	none	0.0
Fz	forcing function in the z direction	float	none	0.0
strike	Aki and Richards strike angle	float	degrees	none
dip	Aki and Richards dip angle	float	degrees	none
rake	Aki and Richards rake angle	float	degrees	none
t0	offset in time	float	s	0.0
freq	frequency	float	none	none
type	selects a particular time dependent function	string	none	RickerInt

Options for the time dependent function (`type`) include: `GaussianInt, Gaussian, RickerInt, Ricker, Ramp, Triangle, Sawtooth, Smoothwave, Brune, BruneSmoothed`, and `Liu`.

9.5 Specifying output

9.5.1 Generic Mapping Toolkit output (gmt)

Required parameters: none

gmt options			
Option	**Description**	**Type**	**Default**
file	name of file to write out gmt csh commands	string	wpp.gmt.csh

9.5.2 commands to control stdout (fileio)

Required parameters: none

fileio options				
Option	**Description**	**Type**	**Units**	**Default**
name	file header name to be prepended on all output files	string	none	wpp
path	path to a directory where all output will be written	string	none	.
verbose	sets the level of diagnostic messages written to standard out	int	none	1
printcycle	sets the interval for printing the cycle, time, dt info	int	none	100

9.5.3 SAC files (sac)

Required parameters:

- Location of the receiver in Cartesian, grid or geographic coordinates

sac options				
Option	**Description**	**Type**	**Units**	**Default**
x	x position of the receiver	float	m	none
y	y position of the receiver	float	m	none
z	z position of the receiver	float	m	none
i	x grid point of the receiver	int	none	none
j	y grid point of the receiver	int	none	none
k	z grid point of the receiver	int	none	none
depth	depth of the receiver	float	m	none
lat	latitude geographic coordinate of the receiver	float	degrees	none
lon	longitude geographic coordinate of the receiver	float	degrees	none
sta	name of the station	string	none	file
sample	sample factor or cycle interval for the seismogram	int	none	1
file	file name header of the SAC file	string	none	sac
type	write out a binary or ascii SAC file	string	none	binary
writeEvery	cycle interval to write out the SAC file to disk	int	none	1000
eventDate	date the event occured: YYYY/MM/DD	int/int/int	none	date of run
eventTime	time the event occured: hours:minutes:seconds	int:int:int	none	time of run

9.5.4 2D slices of data (image)

Required parameters:

- Location of the image slice (x, y, z, i, j or k)
- Timing interval (time, timeInterval, cycle or cycleInterval)

image options				
Option	**Description**	**Type**	**Units**	**Default**
x	x location of visual plane	float	m	none
y	y location of visual plane	float	m	none
z	z location of visual plane	float	m	none
i	x direction node location of visual plane	int	none	none
j	y direction node location of visual plane	int	none	none
k	z direction node location of visual plane	int	none	none
time	simulation time to output image, will be closest depending on dt taken	float	s	none
timeInterval	simulation time interval to output series of images	float	s	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
file	file name header of image	string	none	image
mode	specifies which field is written to the image file	string	none	rho
precision	precision of image data on file (float or double)	string	none	float

Options for mode include:

image mode sub-options	
Value	**Description**
ux	displacement in the x-direction
uy	displacement in the y-direction
uz	displacement in the z-direction
rho	density
lambda	lambda
mu	mu
p	p velocity
s	s velocity
div	divergence (div) of the displacement
curl	magnitude of the rotation (curl) of the displacement
veldiv	divergence (div) of the velocity
velcurl	magnitude of the rotation (curl) of the velocity
mask	the mask array (1 at used points, 0 otherwise)
lat	latitude (in degrees)
lon	longitude (in degrees)
hvelmax	maximum in time of the horizontal velocity
vvelmax	maximum in time of the vertical velocity
haccmax	maximum in time of the horizontal acceleration
vaccmax	maximum in time of the vertical acceleration
topo	topography or elevation [*only available with efile input*]

9.5.5 saving and restoring the simulation (restart)

Required parameters: none

restart options			
Option	**Description**	**Type**	**Default**
file	file header name to be prepended on restart files	string	wpp_restart
dumpInterval	sets the interval for writing out restart files	int	0
fromCycle	cycle from which to restart the code	int	0

9.6 Specifying numerical simulation controls

Artificial dissipation (damping) is used to improve stability of the numerical method in the presence of mesh refinement, and to suppress spurious shear waves in materials with zero shear velocity such as water or air. We call these two dissipation models:

- D_4 dissipation (or damping), described in 7.1
- Curl-curl dissipation (or damping), described in 5.1.1.

9.6.1 Damping command options (damping)

Required parameters: none

damping options			
Option	**Description**	**Type**	**Default**
d4	Normalized coefficient of 4th order dissipation	float	0 or 0.2
curlcurl	Normalized coefficient for curl-curl water damping	float	0.05

The default value for the `d4` parameter is 0.2??? if mesh refinements are present, otherwise it is zero. Note that the dissipation term are not taken into account when the time step is computed. Therefore, a value which is too large can lead to time-stepping instabilities. The dissipation coefficient d4 is given as fractions of the maximum allowed value by the CFL stability condition when only the dissipation term is present (in the absence of the spatial derivatives in the elastic wave equation). Therefore the practical stability limit is closer to 0.1 than 1.0. The same restrictions apply to the coefficient of the curl-curl damping, which is used to remove artificial shear wave in materials with zero shear velocity (see Section 5.1.1).

You can turn on damping by adding a line like

```
damping d4=0.05 curlcurl=0.1
```

Here the fourth derivative damping gets a strength of 0.05 and the curlcurl damping in regions with zero shear velocity (water or air) is set to 0.1.

9.6.2 Controlling solid body motion (projection)

Required parameters: none

projection options			
Option	**Description**	**Type**	**Default**
projectionInterval	Remove invariants each projectionInterval:th time step	int	1000

Projection should only be used if all sources are of point moment type. If point forces are present the solution can develop a solid body translation or rotation unless all point forces sum to zero and the torque of all point forces sum to zero. Projection can be disabled by adding the command

```
projection projectionInterval=0
```

9.6.3 Mesh refinement control options (refinement)

Required parameters: none

refinement options				
Option	**Description**	**Type**	**Unit**	**Default**
x1	minimum x-dim for the refinement regin	float	m	0.0
x2	maximum x-dim for the refinement region	float	m	max x
y1	minimum y-dim for the refinement region	float	m	0.0
y2	maximum y-dim for the refinement region	float	m	max y
z1	minimum z-dim for the refinement region	float	m	0.0
z2	maximum z-dim for the refinement region	float	m	max z
i1	minimum index in x-dim for the refinement region	int	none	1
i2	maximum index in x-dim for the refinement region	int	none	Nx
j1	minimum index in y-dim for the refinement region	int	none	1
j2	maximum index in y-dim for the refinement region	int	none	Ny
k1	minimum index in z-dim for the refinement region	int	none	1
k2	maximum index in z-dim for the refinement region	int	none	Nz
rr	refinement ratio	int	none	2

The index bounds `i1`, `i2`, etc. refer to indices in the coarsest grid. The refinement ratio should be given as a power of two. For example,

```
refinement x1=0 x2=10000 y1=1000 y2=8000 z1=0 z2=3000 rr=8
```

inserts a refinement grid over the region $0 \leq x \leq 10000$, $1000 \leq y \leq 8000$, and $0 \leq z \leq 3000$, with a grid spacing that is a factor 8 finer than the coarsest grid in the computation.

Chapter 10

Examples

This chapter describes the examples defined by the input files in the directory `/tests/examples`.

10.1 A layered problem with mesh refinement

As stated in Chapter 7, the ratio V_s/h should be kept constant over the computational domain to get the same number of points per wavelength everywhere. This is important as the accuracy of the solution is closely connected to the number of points per wavelength. For layered models with different V_p, V_s the use of a uniform grid leads to either under resolved (if h is chosen based on the maximal V_s) or over resolved (if h is chosen based on the minimal V_s) solutions. With mesh refinement the grid size is adjusted to V_s locally, resulting in better use of computational resources.

The input files `/tests/examples/Layer.in` and `/tests/examples/LayerRefine.in` describe a layered model in the domain $(x, y, z) \in [0\,m, 10000\,m]^2 \times [0\,m, 5000\,m]$. The model consists of four layers with the velocities

$$\begin{aligned}
&V_p = 75 m/s, && V_s = 50 m/s, && z \in [0\,m, 2001, m),\\
&V_p = 150 m/s, && V_s = 100 m/s, && z \in [2001\,m, 3001, m),\\
&V_p = 300 m/s, && V_s = 200 m/s, && z \in [3001\,m, 4000, m),\\
&V_p = 600 m/s, && V_s = 400 m/s, && z \in [4001\,m, 5000, m],
\end{aligned}$$

described in the input files by the lines:

```
block vp=75 vs=50 r=200
block vp=150 vs=100 r=200 z1=2001 z2=3001
block vp=300 vs=200 r=200 z1=3001 z2=4001
block vp=600 vs=400 r=200 z1=4001 z2=5000
```

The solution is computed until

```
time t=150
```

and an explosive source with a RickerInt time dependence is placed in the uppermost layer. The frequency is $\omega_0 = 0.025$ Hz and the offset in time is $t_0 = 35\,s$. The source is realized by the line:

```
source x=5000 y=5000 z=500 m0=1e5 mxx=1 myy=1 mzz=1 \
type=RickerInt freq=0.025 t0=35.0
```

The solution is recorded in synthetic seismograms using the sac command

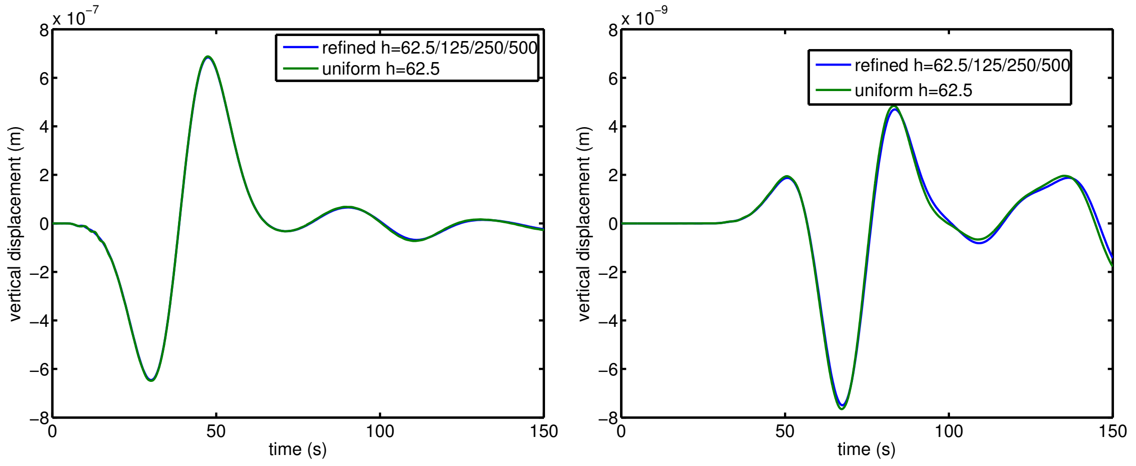


Figure 10.1: The vertical displacements from receiver five $(x, y, z) = (5000, 5000, 0)$ (to the left) and two $(x, y, z) = (5000, 5000, 4500)$ (to the right) are plotted. As expected at 32 points per wavelength both solutions agree well.

```
sac x=5000 y=5000 z=500 file=Layer_sac1 sta=apple writeEvery=50
sac x=5000 y=5000 z=4500 file=Layer_sac2 sta=apple writeEvery=50
sac x=5000 y=5000 z=2000 file=Layer_sac3 sta=apple writeEvery=50
sac x=5000 y=5000 z=3000 file=Layer_sac4 sta=apple writeEvery=50
sac x=5000 y=5000 z=00 file=Layer_sac5 sta=apple writeEvery=50
```

Images of the displacement are saved with five second intervals using the image command

```
image mode=ux x=5000.0 timeInterval=5 file=IM
image mode=uy x=5000.0 timeInterval=5 file=IM
image mode=uz x=5000.0 timeInterval=5 file=IM
```

The output is redirected to the directory `./Layer` (or `./Layer_Refine`) using the fileio command

```
fileio path=Layer
```

For the uniform grid, a suitable grid size is chosen using formula (4.1). The smallest shear velocity in the model is $50\,m/s$ and the frequency content is approximated by (4.2) to be $f = 0.025$ Hz. Thus, a grid size of $62.5\,m$ gives 32 points per wavelength. The computational domain and the grid size are setup by the line

```
grid h=62.5 x=10000 y=10000 z=5000
```

In this example V_s increases with depth by a factor of two between each layer, thus making the shear velocity at the surface eight times lower than the shear velocity in the deepest layer. Therefore, in the deepest layer the solution is highly over resolved with a resolution of 256 points per wavelength.

The situation is ideally suited for mesh refinement. In order to keep the points per wavelength constant the mesh is coarsened 8 times (see `/tests/examples/LayerRefine.in`)

```
grid h=500 x=10000 y=10000 z=5000
```

and mesh refinement patches covering the uppermost layers are added according to:

```
refinement z1=0000 z2=2000 rr=8
refinement z1=1875 z2=3000 rr=4
refinement z1=2750 z2=4000 rr=2
```

NOTE the overlap of the refinement patches (see Figure 7.4 and the discussion in Chapter 7).

Adding mesh refinement reduces both memory and cpu usage. The uniform grid consists of $\sim$ 2.1 million grid points while the grid with mesh refinement consists of $\sim$ 0.97 million grid points. In addition, as the points per wavelength is held constant, the number of time steps for the refined computation is 309 while the uniform computation need to take 2473 steps to reach time 150 s.

In Figure 10.1, the vertical displacements from receiver five and two are plotted. As expected at 32 points per wavelength both solutions agree well.

10.2 Examples from Lifelines project 1A01: Validation of basin response codes

The following examples are taken form the Lifelines project 1A01: Validation of basin response codes, [2]. A detailed description of the physical setup of the LOH problems is also found in [2]. Note that most of the results here are displayed in terms of displacement while in the report they are displayed as velocities.

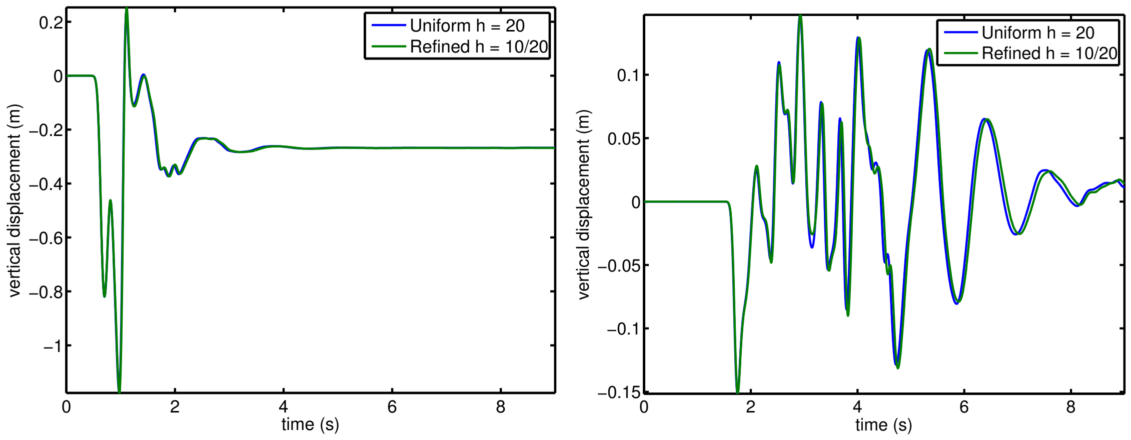


Figure 10.2: LOH.1: The vertical displacement in receiver one and eight for an uniform grid and a grid with mesh refinement.

10.2.1 The LOH.1 problem

The LOH.1 problem, defined in the input script `/tests/examples/LOH.1.in` (the version with mesh refinement is found in `/tests/examples/LOH.1.REFINE.in`) consists of a layer ($z \in [0, 1000m]$) over halfspace problem. The halfspace is represented by the domain $(x, y, z) \in [0, 30000]^2 \times [0, 17000]$. The discretization (here with $h = 20\,m$ or $h = 10\,m\,/20\,m$ for the setup with mesh refinement) of the domain and the material properties in the different layers are described by the lines

```
grid h=50 x=30000 y=30000 z=17000
block vp=4000 vs=2000 r=2600
block vp=6000 vs=3464 r=2700 z1=1000 z2=17000
```

The problem is forced for $9s$ by a point moment source, positioned in the lower half-space. The time function in this problem is a GaussianInt (if setup as in the input file, the GaussianInt source is equivalent to using a Brune time function combined with a post processing step, as is described in [2]). The advantage of using the GaussianInt is that no post processing is necessary. The lines to setup the source and the time duration of the simulation are:

```
time t=9
source x=15000 y=15000 z=2000 Mxy=1 m0=1e18\
t0=0.2 freq=20 type=GaussianInt
```

The solution is recorded in an array of receivers:

```
sac x=15600 y=15800 z=0 file=sac_01 sta=apple writeEvery=50
sac x=16200 y=16600 z=0 file=sac_02 sta=apple writeEvery=50
sac x=16800 y=17400 z=0 file=sac_03 sta=apple writeEvery=50
sac x=17400 y=18200 z=0 file=sac_04 sta=apple writeEvery=50
sac x=18000 y=19000 z=0 file=sac_05 sta=apple writeEvery=50
sac x=18600 y=19800 z=0 file=sac_06 sta=apple writeEvery=50
sac x=19200 y=20600 z=0 file=sac_07 sta=apple writeEvery=50
sac x=19800 y=21400 z=0 file=sac_08 sta=apple writeEvery=50
sac x=20400 y=22200 z=0 file=sac_09 sta=apple writeEvery=50
sac x=21000 y=23000 z=0 file=sac_10 sta=apple writeEvery=50
```

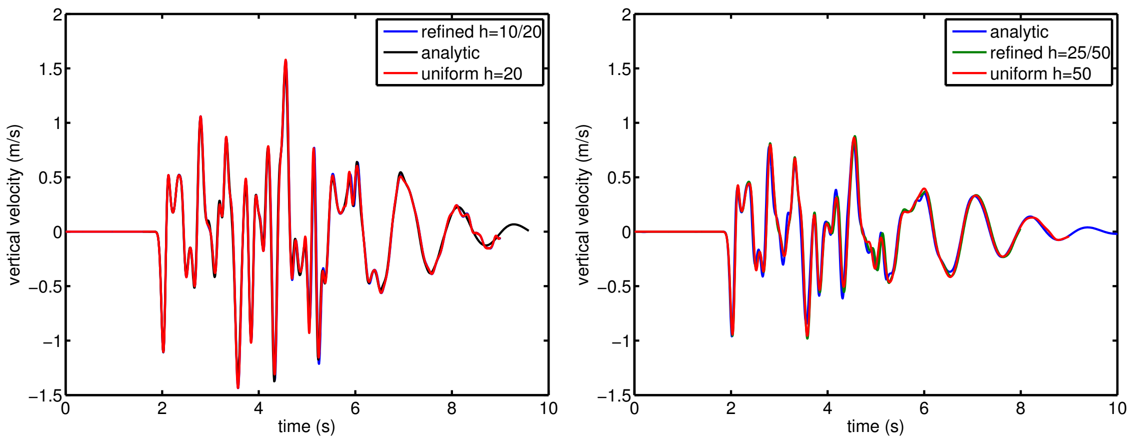


Figure 10.3: To the left: A comparison of the vertical velocities recorded in station 10 for LOH.1 and the analytic solution. To the right: LOH.3 includes attenuation and the computed solutions (recorded in station 10) oscillate less than the solution to the perfectly elastic LOH.1 problem. Here a comparison of the vertical velocities recorded in station 10 for LOH.3 and the analytic solution are compared.

Mesh refinement

Here, the difference in shear velocity between the layer and the halfspace is close to two, making the problem well suited for mesh refinement. In order to increase the resolution close to the surface a mesh refinement patch, conforming with the upper layer, is added:

```
refinement z1=0 z2=1000 rr=2
```

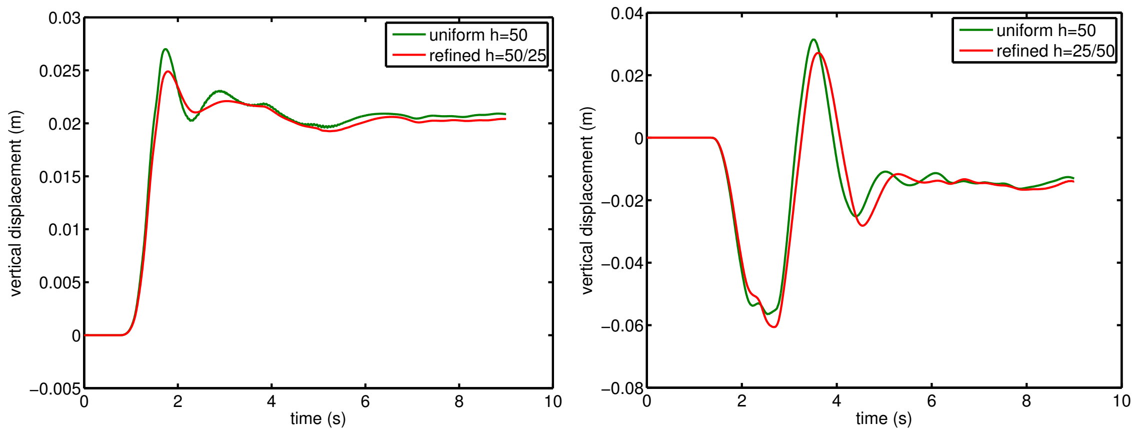


Figure 10.4: LOH.2: The vertical displacement in receiver one and seven for an uniform grid and a grid with mesh refinement.

The uniform grid (with $h = 20\,m$) consists of $\sim$ 2 billion grid points and is approaching the limit of what is tractable on the computational resources available to the authors. In this case mesh refinement allow for twice the resolution at the surface at the manageable cost of increasing the number of grid points to $\sim$ 2.7 billion and the number of time steps by a factor of 1.27. In contrast, refining the uniform grid by a factor of two everywhere increases the memory requirements eightfold and the total computational time increases by a factor of 16.

In Figure 10.2 the vertical displacements, recorded in stations one and eight, are plotted. The results for the computation on the uniform and the refined grid agree well, indicating that a computation on a coarser refined grid, say $h = 20\,m/40\,m$, would be sufficient to resolve the solution accurately. Figure 10.3 contains a plot of the vertical velocity at station 10. The Figure shows the analytic solution as well as the solutions computed using the uniform and the refined grid. The computed solutions agree well with the analytic solution.

10.2.2 The LOH.2 problem

The geometrical setup of the LOH.2 problem is identical to that of LOH.1, the difference between the two problems is that LOH.2 models an earthquake along a fault plane. The fault is modeled as a large number of moment point sources with the time dependency given by the Brune function with different offsets in time (depending on location). Here a fault plane in the y-z-plane is modeled. The input files for LOH.2 with and without mesh refinement are found in `/tests/examples/LOH.2.REFINE.in` and `/tests/examples/LOH.2.in`.

The solutions at receivers one and seven are displayed in Figure 10.4. Due to the complex source mechanism the results differ a bit more than for LOH.1.

10.2.3 The LOH.3 problem

The geometrical setup of the LOH.3 problem is identical to that of LOH.1. The difference between the two problems is that LOH.3 includes attenuation. The attenuation is added by the lines:

```
attenuation frequency=2.5
block vs=2000 vp=4000 r=2600             Qs=40   Qp=120
block vs=3464 vp=6000 r=2700 z1=1000     Qs=69.3 Qp=155.9
```

Attenuation predominantly damps waves with short wavelengths. This is illustrated in Figure 10.3, were the computed solutions are compared to the analytic solutions to the perfectly elastic LOH.1 problem and the LOH.3 problem. The input files for LOH.3 are found in `/tests/examples/LOH.3.REFINE.in` and `/tests/examples/LOH.3.in`.

10.3 Simulation of real seismic events

10.3.1 The September 3, 2000 Yountville earthquake

A brief description of the magnitude 5.2 September 3, 2000 Yountville earthquake is found at the website `http://quake.usgs.gov/recent/reports/napa/`

In the directory `/tests/examples` there are three different input files that can be used to model this event:

`yountville-1000m.in`	elastic model on a uniform grid with grid size 1000 m
`yountville-1000m-R.in`	elastic model on a refined grid with grid sizes 1000/500/250 m
`yountville-1000m-RA.in`	model with attenuation on a refined grid with grid sizes 1000/500/250 m

The simulation domain (see also Figure 5.2) is specified to be the rotated 360 km $\times$ 180 km $\times$ 45 km domain:

```
grid x=360e3 y=180e3 z=45e3 h=1000. lat=39.60 lon=-122.60 az=144
```

All the the Yountville input files use the USGS bay area velocity model, see Chapter 5.3. However, as the USGS model does not cover the entire computational domain a one dimensional model from Stidham et al. [10] is used to specified the velocities throughout the volume:

```
block vp=3200. vs=1500. r=2280. z1=0. z2=1000.
block vp=4500. vs=2400. r=2280. z1=1000. z2=3000.
block vp=4800. vs=2780. r=2580. z1=3000. z2=4000.
block vp=5510. vs=3180. r=2580. z1=4000. z2=5000.
block vp=6210. vs=3400. r=2680. z1=5000. z2=17000.
block vp=6890. vs=3980. r=3000. z1=17000. z2=25000.
block vp=7830. vs=4520. r=3260. z1=25000.
```

The detailed USGS model is then used to specify the velocities wherever available using the etree command:

```
efile model=SF query=FIXEDRES vsmin=500 vpmin=765 \
etree=USGSBayAreaVM-05.1.0.etree xetree=USGSBayAreaVMExt-05.1.0.etree
```

The earthquake is modeled as a moment source with a GaussianInt time dependence. Its orientation is specified using strike, dip and rake:

```
source type=GaussianInt t0=5.0 lat=38.377 lon=-122.414 depth=11.0e3 freq=0.5612\
m0=3.74e16 strike=60 dip=75 rake=18
```

In the models using mesh refinement the lines

```
refinement rr=4 z1=0 z2=2400
refinement rr=2 z1=0 z2=27200
damping d4=0.1
```

are added. Note that the damping command is used to reduce the artificial damping slightly. In the model using both attenuation and mesh refinement the specification of the one dimensional model is adjusted to include attenuation (via quality factors)

```
attenuation
block vp=3200. vs=1500. r=2280. z1=0. z2=1000.    qs=0.30e5 qp=0.45e5
block vp=4500. vs=2400. r=2280. z1=1000. z2=3000. qs=2.4e5 qp=3.6e5
block vp=4800. vs=2780. r=2580. z1=3000. z2=4000. qs=2.78e5 qp=4.17e5
block vp=5510. vs=3180. r=2580. z1=4000. z2=5000. qs=3.18e5 qp=4.77e5
block vp=6210. vs=3400. r=2680. z1=5000. z2=17000. qs=3.4e5 qp=5.1e5
block vp=6890. vs=3980. r=3000. z1=17000. z2=25000. qs=3.98e5 qp=5.97e5
block vp=7830. vs=4520. r=3260. z1=25000. qs=4.52e5 qp=6.78e5
```

The ETREE command automatically reads the stored values for the quality factors and does not have to be modified.

The solution is recorded in the stations specified on the map in Figure 10.5 and 2D images of the accumulated maximal velocities in the vertical and horizontal directions are saved.

10.3.2 The August 17, 1999 Bolinas Lagoon earthquake

The files `/tests/examples/bolinas.in` and `/tests/examples/bolinas.in` use the same computational model as the Yountville example but implements a different source, modeling the Bolinas Lagoon Earthquake. Both the Bolinas models use mesh refinement and the latter use attenuation. A description of the event is found at `http://quake.wr.usgs.gov/recent/reports/bolinas/index.html`.

In Figure 10.6 the vertical displacements recorded in the station POTR (see Figure 10.5) is plotted. The POTR station is located relatively far away form the event and the signals travel through domains with significant attenuation, smoothing the waveform.

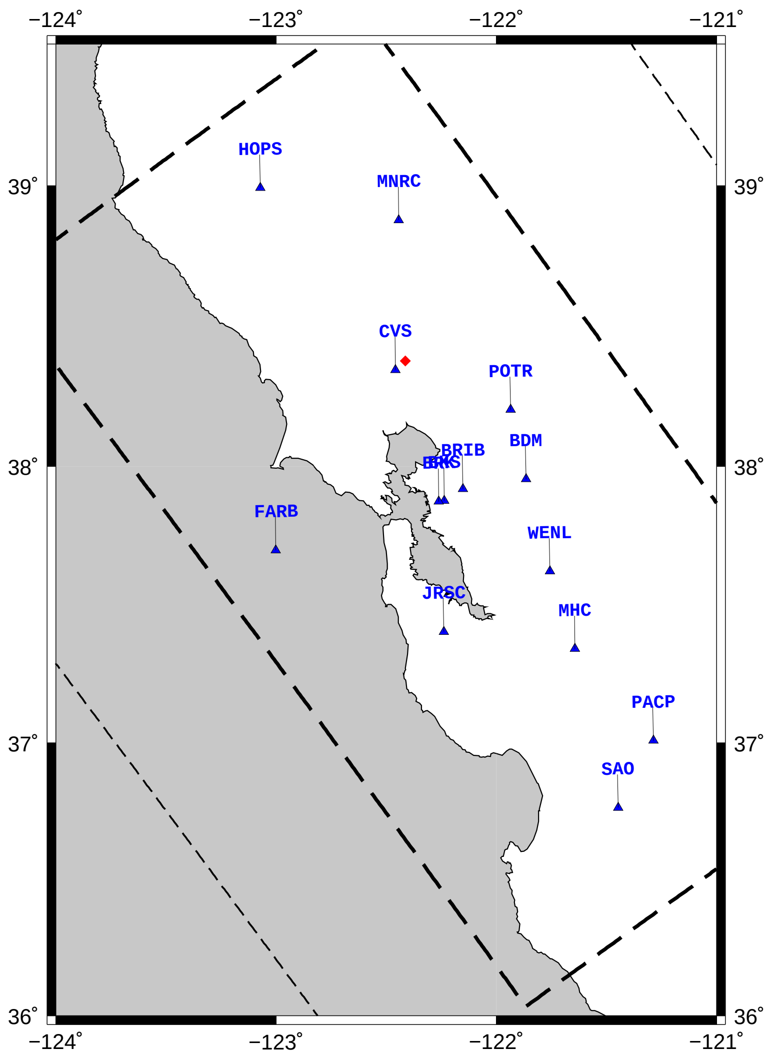


Figure 10.5: Stations recording the September 3, 2000 Yountville Earthquake. The event location is marked in red.

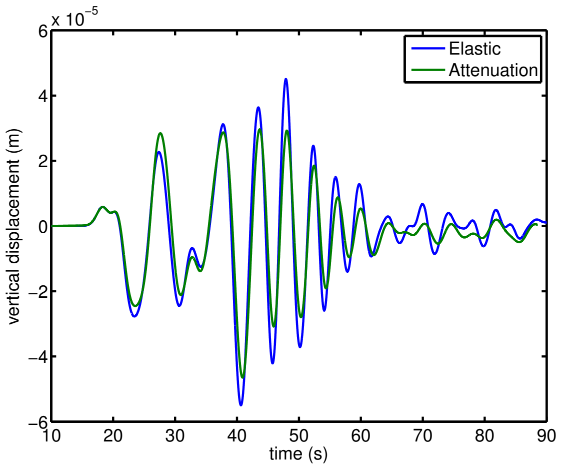


Figure 10.6: The vertical displacement caused by the Bolinas event recorded in the POTR station. The attenuated model is less oscillatory.

Chapter 11

Building WPP

WPP source can be downloaded from:

```
https://computation.llnl.gov/casc/serpentine/software.html
```

Note: A detailed example for building and installing *WPP* on a Linux workstation can be found in section 11.8.

11.1 Supported platforms

WPP and its supporting libraries have been built on LINUX based desktops and supercomputers. We have built WPP using both the Gnu and Intel compilers. If the third party libraries can be built successfully on a platform, WPP will most likely also work.

```
Gnu:   g++/gcc/g77    version 4.0.2
Intel: icpc/icc/ifort version 9.1
```

11.2 Build tools

WPP does not use autoconf and automake (i.e., `configure` and `make`), but instead SCons (`scons`) which is a software construction tool intended to replace autoconf and automake. Unlike the two step process with running `configure` and then `make`, with SCons both the configuration and build steps occur during the `scons` command. To use SCons as a build system, there are two tools required:

- `python` A widely used scripting language
- `scons` A python based software construction tool

Assuming `python` is already installed (`v2.3.5` or higher from `www.python.org`), it is straightforward to install `scons`. You can test to see if python is installed by typing `which python` at the command line. SCons can be downloaded from the website `www.scons.org` (`v0.96.1`), to install it, simply invoke the following command from the top level directory:

```
shell> python setup.py install --prefix=/dir/of/your/choice
```

Note: `scons` and `python` can be installed anywhere in your path. Typically people have them installed into `/usr/local/bin`, however if you don't have root access any other directory reachable by your UNIX path will work.

11.3 Directory structure

For more detailed information about the files and directories in the WPP source, please read `wpp-v1.1/README.txt` after you've untared your tarball.

```
shell> gunzip wpp-v1.1.tar.gz; tar xfv wpp-v1.1.tar
```

This will create a directory named `wpp-v1.1` which will contain several files and subdirectories:

- `INSTALL.txt` Information about how to build WPP
- `KNOWN-BUGS.txt` List of known problems or issues with porting or other bugs
- `README.txt`
- `TPL.txt` Details about building the third party libraries WPP requires
- `configs` Directory which contains build configuration files
- `src1.1` C++ and Fortran source of WPP
- `tools` Matlab scripts which may be helpful for post processing and analysis
- `tests/examples` Referenced and documented examples from the users guide
- `utilities` Auxiliary code used to interface with the cencalvm library
- `SConstruct` A SCons "makefile"
- `wave.py python` script used to print "WPP Lives" at end of build

11.4 Third party library (TPL) build instructions

In addition to requiring the two build tools, there is a minimal set of third party libraries which WPP depends upon:

- `blitz` An array class library
- `mpi/mpio` An implementation of the Message Passing Interface (MPI) which WPP uses for its parallelism.

The WPP build system expects the third party libraries to all be collocated in the same directory. This can be accomplished by actually building them all into the same install directory - or creating a directory and symlinking to already built third party libraries. In addition it is required that the TPLDIR is named lib; if not it may be necessary to change the `wpp_v1.1/SConstruct` file. In any case, we will refer to this common directory as TPLDIR.

```
                           ____________
                          |  wpp_v1.1  |
                          |____________|
                           /          \
                          /            \
                         /              \
                    ________               ________
   ...  README.txt | src1.1 | SConstruct  | TPLDIR | ...
                   |________|             |________|
                     [...]               /     |     \
                                    _____   _______   _______
                              ...  | bin | |include| | src   |  ...
                                   |_____| |_______| |_______|
                                                    /    |    \
                                             _________  ________   _____
                                     ...    |blitz-0.9||cencalvm| | ... |
                                            |_________||________| |_____|
```

Make an install directory for the third party library sets and a source directory where all the tarfiles will be located:

```
shell> mkdir TPLDIR
shell> mkdir TPLDIR/src
```

Setup the environment to point to the C++/C/Fortran compilers of your choice. On LINUX installations we normally will use the Gnu compilers. These are used by the TPL build processes. We will also use an environment variable to refer to the third party library install location:

```
shell> setenv WPP_TPL TPLDIR
shell> setenv CC gcc
shell> setenv CXX g++
shell> setenv F77 g77
```

NOTE *For third party libraries installed at LLNL, please see specific instructions found in TPL.txt. This includes platform and OS specific gotchas we have encountered porting these libraries.*

11.4.1 Message Passing Interface (MPI) library (REQUIRED)

WPP should be compatible with any standardized version of MPI. It requires the MPI-IO library, from the MPI-2 Standard. If you don't already have a vendor version of MPI for your cluster/machine, we recommend using the mpich implementation from Argonne Labs (v1.0.3 from www-unix.mcs.anl.gov/mpi/mpich) To build this implementation:

```
shell> ./configure --prefix=${WPP_TPL} --disable-f77 --disable-f90 \
            |& tee configure.log
shell> make |& tee make.log
shell> make install |& tee install.log
shell> setenv PATH ${WPP_TPL}/bin:$PATH
```

To make sure you are going to pick up the right daemon and `mpirun` script, execute:

```
shell> which mpd; which mpirun; which mpiexec
```

These should refer to the ones where you just built it. Now, start an `mpd` daemon.

```
shell> mpd &
```

Note: if one is already running, kill it with

```
shell> mpdallexit
```

Ensure it works:

```
shell> mpdtrace
```

This should return your host. To test the implementation works:

```
shell> make testing
```

11.4.2 BLITZ++ array library (REQUIRED)

WPP builds successfully with version 0.9 from `http://www.oonumerics.org/blitz`. To build this library:

```
shell> cd blitz-0.9
shell> ./configure --prefix=${WPP_TPL}
shell> make lib
shell> make install
shell> make check-testsuite
```

11.5 Configuring and building WPP

For a minimal installation you are now set to configure and build WPP. If you want support for the efile command and the image3d command you should first install the libraries as described in Sections 11.6 and 11.7 below.

To build WPP on your local system, you will need to create a configuration file. Examples are shown in the configs directory, a detailed example can be found in configs/template.py. Once you have created a configuration file, say configs/myconfig.py, you will need to set the WPPCONFIG env variable to locate it.

```
shell> setenv WPPCONFIG configs/myconfig.py
```

11.5.1 Invoking scons to build WPP

From the top level directory, where the SConstruct file is located:

```
shell> scons -h
```

This will give you the build options (debug/opt/etc.) For building an optimized executable:

```
shell> scons
```

This will read in your configuration file, and build WPP. If you have questions, please email Kathleen McCandless at mccandless2@llnl.gov

11.6 Additional libraries required for the efile command (OPTIONAL)

To utilize the efile material command, WPP requires three additional libraries. Note that these libraries are only needed if you want to use the efile command.

Efile Option Libraries	**Version**	**Location**
etree	3.0	www.cs.cmu.edu/ euclid/
proj4	4.4.9	proj.maptools.org
cencalvm	0.4.1	ftp://ehzftp.wr.usgs.gov/baagaard/cencalvm/software

To enable the use of the efile command, configure WPP with **-DENABLE_ETREE**. This can be set in the CXXFLAGS (see configs/tux-all-libs.py).

11.6.1 Euclid etree database query library (OPTIONAL)

See `http://www.sf06simulation.org/geology/velocitymodel/querydoc/INSTALL.html` for more details

```
shell> cd euclid3-1.2/libsrc
shell> make
```

There is no install target so the files have to be copied to the install directories:

```
shell> cp *.h ../../../include/.
shell> cp libetree.* ../../../lib/.
```

Set environment variables which are needed during the cencalvm build. These are not needed during a WPP compile, just building the cencalvm library

```
shell> setenv ETREE_LIBDIR ${WPP_TPL}/src/euclid3-1.2/libsrc
shell> setenv ETREE_INCDIR ${ETREE_LIBDIR}
shell> setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${ETREE_LIBDIR}
```

11.6.2 PROJ4 Projection Library (OPTIONAL)

```
shell> ./configure --prefix=${WPP_TPL}
shell> make
shell> make install
shell> setenv PROJ4_LIBDIR ${WPP_TPL}/lib
shell> setenv PROJ4_INCDIR ${WPP_TPL}/include
shell> setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${PROJ4_LIBDIR}
```

Note: Environment variables above are used in cencalvm library build

11.6.3 Central California velocity model query library (OPTIONAL)

```
shell> ./configure --prefix=${WPP_TPL} \
            CPPFLAGS="-I${ETREE_INCDIR} -I${PROJ4_INCDIR}" \
            LDFLAGS="-L${ETREE_LIBDIR} -L${PROJ4_LIBDIR}"  \
            CC=${CC} CXX=${CXX} F77=${F77}
shell> make
shell> make install
```

11.7 Additional library for image3d command (OPTIONAL)

To use the `image3d` command, WPP must be built with the **-DENABLE_BOW** directive. This can be set in the `CXXFLAGS` (see `configs/tux-all-libs.py`). The brick of wavelet (BOW) library is a part of LibGen, we used a version from October 2006. See `www.cognigraph.com/LibGen/` to download it.

11.7.1 Building BOW

Edit the makescript to unset ZINCLUDE from ../../Gzlib/zlib to .

```
    set ZINCLUDE = .

shell> ./makescript
shell> cp *.h ../../include/.
shell> cp *.a ../../lib/.
```

11.8 Detailed build example for Linux

This section describes how to install *WPP* and its required libraries in a directory `/home/andersp/src/wpp-1.2` using C shell (`tcsh`). To make this work on your machine, you will have to replace `/home/andersp/src` by the actual installation directory on your machine.

To make a minimal installation of *WPP* version 1.2 you go through the follwoing steps

1. Download wpp tar ball from https://computation.llnl.gov/casc/serpentine. Unpack in `/home/andersp/src` which will put all the wpp files in a new directory `/home/andersp/src/wpp-1.2`.

2. Build the third party libraries

   (a) scons: Download the scons compressed tarball from www.scons.org. Unpack in `/home/andersp/src`. Go to the `/home/andersp/src/scons-0.98.5` directory and install the package by executing:

   ```
   python setup.py install --prefix=/home/andersp
   ```

   (b) blitz: Download the blitz compressed tarball from `www.oonumerics.org/blitz`. Unpack in `/home/andersp/src` Go to the `/home/andersp/src/blitz-0.9` directory. Setup the library by executing the commands:

   ```
   configure --prefix=/home/andersp
   make lib
   make install
   ```

(c) mpich: Download the compressed tarball from `www-unix.mcs.anl.gov/mpi/mpich1`. Unpack in `/home/andersp/src`. Go to the directory `/home/andersp/src/mpich-1.2.7p1` and execute the following commands:

```
configure --prefix=/home/andersp
make
make install
```

3. Modify your .cshrc file to include the statement

```
setenv WPPCONFIG /home/andersp/src/wpp-v1.2/configs/rallyxlin.py
```

4. Go to the `/home/andersp/src/wpp-1.2/configs` directory and copy the file template.py to rallyxlin.py.

5. Edit the file `/home/andersp/src/wpp-1.2/configs/rallyxlin.py` to suit your environment. Here are the changes I made:

```
...
# To locate your third party libraries built for wpp, set the "tool" directory
env.Replace(WPPLIBDIR = os.path.join(os.sep, ’home’, ’andersp’, ’lib’))

# Set a different C++, C and Fortran compiler.
env.Replace(CXX     = os.path.join(os.sep, ’home’, ’andersp’, ’bin’, ’mpicxx’)
env.Replace(CC      = os.path.join(os.sep, ’home’, ’andersp’, ’bin’, ’mpicc’))
env.Replace(FORTRAN = os.path.join(os.sep, ’home’, ’andersp’, ’bin’, ’mpif77’)
...
```

6. Build wpp: Go to the directory `/home/andersp/src/wpp-1.2` and execute the “scons” command:

```
shell> scons
```

If all the previous setup steps were successful, scons should end with the printout:

```
‘‘’-.,_,.-’‘‘’-.,_,.=’‘‘’-.,_,.-’‘‘’-.,_,.=’‘‘‘‘’-.,_,.-’‘‘’-.,_,.=’‘‘

     ____    __    ____ .______   .______
     \   \  /  \  /   / |   _  \  |   _  \
      \   \/    \/   /  |  |_)  | |  |_)  |
       \            /   |   ___/  |   ___/
        \    /\    /    |  |      |  |
         \__/  \__/     | _|      | _|

      __       __  ____    ____  _______     _______. __
     |  |     |  | \   \  /   / |   ____|   /       ||  |
     |  |     |  |  \   \/   /  |  |__     |   (----‘|  |
     |  |     |  |   \      /   |   __|     \   \    |  |
     |  ‘----.|  |    \    /    |  |____.----)   |   |__|
     |_______||__|     \__/     |_______|_______/    (__)

     in /home/andersp/src/wpp-v1.2/optimize_v1.2

‘‘’-.,_,.-’‘‘’-.,_,.=’‘‘’-.,_,.-’‘‘’-.,_,.=’‘‘‘‘’-.,_,.-’‘‘’-.,_,.=’‘‘
```

7. Test wpp: Go to the directory `/home/andersp/src/wpp-1.2/tests/TwilightTest`. Run wpp in test mode by executing

   ```
   shell> mpirun -np 1 ../../optimize_v1.2/wpp twilight1.in
   ```

   The output at the end of the run should read

   ```
   ...
    Time is 0.8
    u[0]:  max err = 0.000716908 l2 err = 7.64858e-05
    u[1]:  max err = 0.000716908 l2 err = 7.64858e-05
    u[2]:  max err = 0.000838446 l2 err = 7.65661e-05
   ============================================================
    Wave Propagation Program (WPP) Finished!
   ============================================================
   ```

Chapter 12

Testing WPP

After the executable has been built it is a good idea to verify that the program works correctly. In principle these tests should give the same answer regardless of the number of processors, but in practice we have noticed slight differences in the reported errors. It can also be expected that you get slightly different answers on different types of architectures, due to different handling of round-off in flating opint arithmetic.

12.1 Method of manufactured solutions

The first kind of test checks the order of accuracy by using the method of manufactured solution. This technique starts out by choosing a smooth analytic solution $\mathbf{u}(\mathbf{x}, t)$. In WPP we use the following time-dependent displacement field $\mathbf{u} = (U, V, W)^T$,

$$\begin{aligned} U(x,y,z,t) &= \sin(\omega(x-ct))\sin(\omega y)\sin(\omega z), \\ V(x,y,z,t) &= \sin(\omega x)\sin(\omega(y-ct))\sin(\omega z), \\ W(x,y,z,t) &= \sin(\omega x)\sin(\omega y)\sin(\omega(z-ct)), \end{aligned}$$

where $c = 1.3$. The wave number ω can be set in the input file, but has the default value $\omega = 1$. In this tet mode, the material has particular properties,

Internal and boundary forcing functions are computed to match the above exact solution. The forced problem is then solved numerically and since the exact solution is know, we can evaluate the error in the numerical solution at all points in space and time. The order of accuracy can be tested by running the same calculation with different grid sizes. WPP uses a second order accurate method, so the error in the solution should be reduced by a factor of four when the grid size is reduced by a factor of two, assuming that the solution is properly resolved on both grids.

The source code distribution contains the input file `twilight1.in` in the directory `tests/TwilightTest`. This case can be executed on a single processor workstation and should only take a couple of seconds to run. To execute this case, you go to that directory and issue the commands

```
shell> mpirun -np 1 ../../optimize_v1.2/wpp twilight1.in
```

At the end of the run, *WPP* prints out the error in max and L_2 norms:

```
...
Time step 1 t = 0
Time step 90 t = 0.791111
```

```
 Time is 0.8
 u[0]:  max err = 0.000716908 l2 err = 7.64858e-05
 u[1]:  max err = 0.000716908 l2 err = 7.64858e-05
 u[2]:  max err = 0.000838446 l2 err = 7.65661e-05
============================================================
 Wave Propagation Program (WPP) Finished!
============================================================
```

There is another input file, `twiligt2.in`, which sets up the same case but with half the grid size. This case has eight times more grid points and needs to take twice as many time steps, so it will take about 16 times longer to execute. It is still small enough to run on a single processor workstation. At the end of this run, *WPP* outputs:

```
...
Time step 1 t = 0
Time step 101 t = 0.444444
Time step 180 t = 0.795556

 Time is 0.8
 u[0]:  max err = 0.00021135 l2 err = 1.82593e-05
 u[1]:  max err = 0.00021135 l2 err = 1.82593e-05
 u[2]:  max err = 0.000246531 l2 err = 1.83094e-05
============================================================
 Wave Propagation Program (WPP) Finished!
============================================================
```

We can now check the ratio of the errors in max and L_2 norm:

$$\frac{8.38446}{2.46531} \approx 3.40, \quad \frac{7.65661}{1.83094} \approx 4.18.$$

There is a third inputfile, `twilight3.in`, which refines the grid by another factor of two. This case will fit in memory of most workstations, but you might want to run this case in parallel since it will take about 16 times longer to run than the previous case. This run gives the output:

```
...
Time step 1 t = 0
Time step 101 t = 0.222222
Time step 201 t = 0.444444
Time step 301 t = 0.666667
Time step 360 t = 0.797778

 Time is 0.8
 u[0]:  max err = 6.19938e-05 l2 err = 4.44841e-06
 u[1]:  max err = 6.19937e-05 l2 err = 4.44841e-06
 u[2]:  max err = 7.15218e-05 l2 err = 4.46357e-06
============================================================
 Wave Propagation Program (WPP) Finished!
============================================================
```

The ratios between the errors in max and L_2 norm are now

$$\frac{2.46531}{0.715218} \approx 3.44, \quad \frac{1.83094}{0.446357} \approx 4.10.$$

We remark that in the limit as the grid size goes to zero, the ratios should approach 4 for a second order accurate method.

There are additional input files in the same directory for testing the convergence rate when meshrefinement is used.

Index

Bibliography

[1] K. Aki and P.G. Richards. *Quantitative Seismology*. University Science Books, second edition, 2002.

[2] Steven M. Day et al. Test of 3D elastodynamic codes: Final report for lifelines project 1A01. Technical report, Pacific Earthquake Engineering Center, 2001.

[3] 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.

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

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

[6] Pengcheng Liu and Ralph J. Archuleta. Efficient modeling of Q for 3d numerical simulation of wave propagation. *Bull. Seism. Soc. Am.*, 96:1352–1358, 2006.

[7] Pengcheng Liu, Ralph J. Archuleta, and Stephen H. Hartzell. Perdiction 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.

[8] Harold M. Mooney. Some numerical solutions for Lamb's problem. *Bulletin of the Seismological Society of America*, 64, 1974.

[9] S. Nilsson, N.A. Petersson, B. Sjögreen, and H.-O. Kreiss. Stable difference approximations for the elastic wave equation in second order formulation. *SIAM J. Numer. Anal.*, 45:1902–1936, 2007.

[10] C. Stidham, M. Antolik, D. Dreger, S. Larsen, and B. Romanowicz. Three-dimensional structure influences on the strong-motion wavefield of the 1989 loma prieta earthquake. *Bulletin of the Seismological Society of America*, 89(5):1184–1202, 1999.

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