

User's guide to the Wave Propagation Program (WPP) version 1.0

Daniel Appelö, Stefan Nilsson, N. Anders Petersson, Björn Sjögreen^{1 2},
Kathleen McCandless³
Arthur J. Rodgers⁴
Lawrence Livermore National Laboratory

June 25, 2007

¹Lawrence Livermore National Laboratory technical report UCRL-SM-230257. This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory, under Contract W-7405-Eng-48.

²Center for Applied Scientific Computing, Computations Directorate

³Biology, Chemistry, Atmosphere and Earth Division, Computations Directorate

⁴Atmosphere, Earth and Energy Department, Energy and Environment Directorate

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

Contents

1	Getting started	4
1.1	Running <i>WPP</i>	4
1.2	Introductory example: Lamb's problem	5
2	Coordinate system, units and the grid	10
2.1	Geographic coordinates	11
3	Sources, time-functions and grid sizes	13
3.1	Sources and time-functions in <i>WPP</i>	13
3.1.1	Gaussian	14
3.1.2	GaussianInt	15
3.1.3	Ricker	15
3.1.4	RickerInt	15
3.1.5	Triangle	15
3.1.6	Sawtooth	16
3.1.7	Ramp	16
3.1.8	Smoothwave	16
3.1.9	Brune	18
3.1.10	BruneSmoothed	18
3.1.11	VerySmoothBump	18
3.2	How fine does the grid need to be?	19
3.2.1	Lamb's problem revisited	24
4	The material model	26
4.1	The block command	26
4.1.1	Modeling water with blocks	27
4.2	The vfile command	28
4.3	The efile command	29
4.3.1	Modeling water with Etree models	30
5	Output options	31
5.1	Setting the output directory	31
5.2	Time-history at a point: the sac command	31
5.3	2-D cross-sectional data: the image command	32
5.4	Volumetric data: the image3d command	33
5.5	Restart files	34

6	Keywords in the input file	35
6.1	Specifying the grid parameters (grid) [required]	35
6.2	Specifying the time parameters (time) [required]	36
6.3	Specifying the model [required]	36
6.3.1	specifying a box shaped sub-region	36
6.3.2	basic block command (block)	36
6.3.3	velocity file (vfile)	37
6.3.4	etree database files (efile)	37
6.4	Specifying the source (source)	38
6.5	Specifying output	39
6.5.1	commands to control stdout (fileio)	39
6.5.2	SAC files (sac)	40
6.5.3	2D slices of data (image)	40
6.5.4	3D volumetric data (image3d)	41
6.5.5	saving and restoring the simulation (restart)	42
6.6	Specifying numerical simulation controls	42
6.6.1	curl-curl damping command options (in water only)	42
6.6.2	Controlling solid body motion (projection)	42
7	Building WPP	43
7.1	Supported platforms	43
7.2	Build tools	43
7.3	Directory structure	44
7.4	Third party library (TPL) build instructions	44
7.4.1	Message Passing Interface (MPI) library (REQUIRED)	45
7.4.2	BLITZ++ array library (REQUIRED)	46
7.5	Configuring and building WPP	46
7.5.1	Invoking scons to build WPP	46
7.6	Additional libraries required for the efile command (OPTIONAL)	47
7.6.1	Euclid etree database query library (OPTIONAL)	47
7.6.2	PROJ4 Projection Library (OPTIONAL)	47
7.6.3	Central California velocity model query library (OPTIONAL)	48
7.7	Additional library for image3d command (OPTIONAL)	48
7.7.1	Building BOW	48
7.8	Detailed build example for Mac OS X	48

Chapter 1

Getting started

WPP is a parallel computer program for simulating time-dependent elastic and acoustic wave propagation. *WPP* solves the elastic wave equation using a finite difference approach on a Cartesian grid, see [6] for details. Version 1.0 of *WPP* implements substantial capabilities for 3-D seismic modeling, with a free surface condition on the top boundary, Clayton-Engquist far-field boundary conditions on all other boundaries, point force and point moment tensor source terms with many time dependencies, fully 3-D material model specification, output of synthetic seismograms in SAC [2] format, output of 2-D slices of the solution field as well as the material model, and output of 3-D solution fields.

WPP comes with some examples, which can be found in the tests/examples subdirectory of the downloaded source code.

1.1 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 one 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* will read files with any extension.

Running on the Livermore Computing Linux Machines under *srun*:

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

The above command runs *wpp* on 16 nodes utilizing 32 processors on the debug partition using *xxx.in* as the input file. Note that the debug partition time limit is limited to 30 minutes. Larger or longer jobs must be submitted through the batch system using the *psub* command. Refer to the Livermore Computing web pages for detailed information (<http://www.llnl.gov/computing>).

Running on other platforms (Linux desktop/laptop):

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

This command will run 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 will output version information for the `wpp` executable. This information is by default printed at the beginning of every run.

```
shell> srun -p pdebug wpp -v
```

```
-----  
                        WPP Version 1.0  
-----  
Compiled: Wed Dec 14 10:31:54 2005  
By:      kmccandl  
Machine: mcr39  
Compiler: /usr/local/bin/mpiicpc  
Lib:      /usr/gapps/wpp/chaos_3_x86_elan3/tools/lib9.0.1/lib  
-----
```

1.2 Introductory example: Lamb's problem

The version of Lamb's problem [4] 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 1.1.

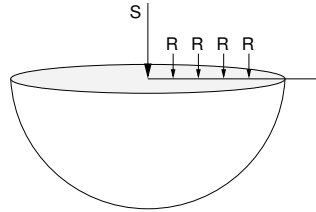


Figure 1.1: A schematic picture of Lamb's problem. A time-dependent source pointing downward into an elastic half-space. There are four hypothetical seismic stations located on the surface recording the vertical and radial displacements.

There are closed form solutions available for Lamb's problem (see e.g. [5]) which can be used to test that *WPP* is working properly by measuring the error for successively refined 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 $1000 \times \sqrt{3} \text{ m/s}$ 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 3.2) with $\omega = 1 \text{ Hz}$, $t_0 = 1 \text{ s}$ 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 $h = 40 \text{ m}$):

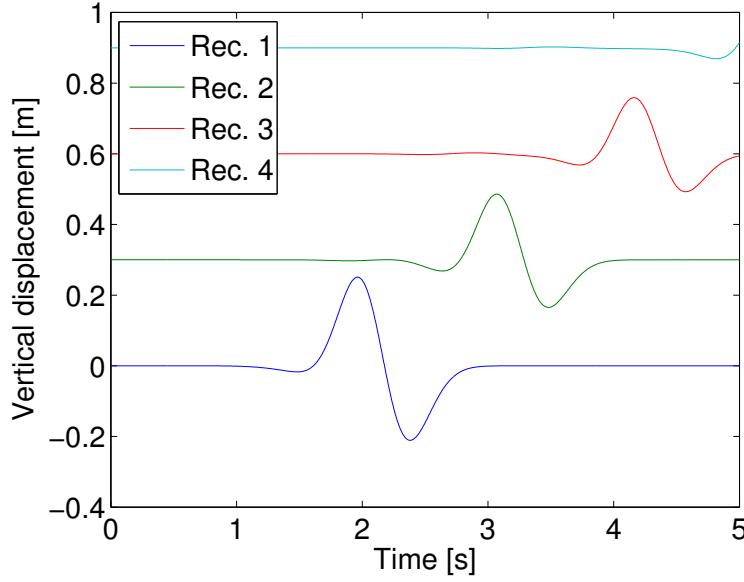


Figure 1.2: The vertical displacements as a function of time. The displacement recorded at receivers 2, 3 and 4 have been offset in the vertical direction to improve readability.

```
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 \
      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 3.1.

For the grid spacing $h = 40$ the vertical displacement for the different receivers can be found in Figure 1.2. The waveforms are all smooth and the problem appears to be well resolved. In Figure 1.3 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 1.3 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:

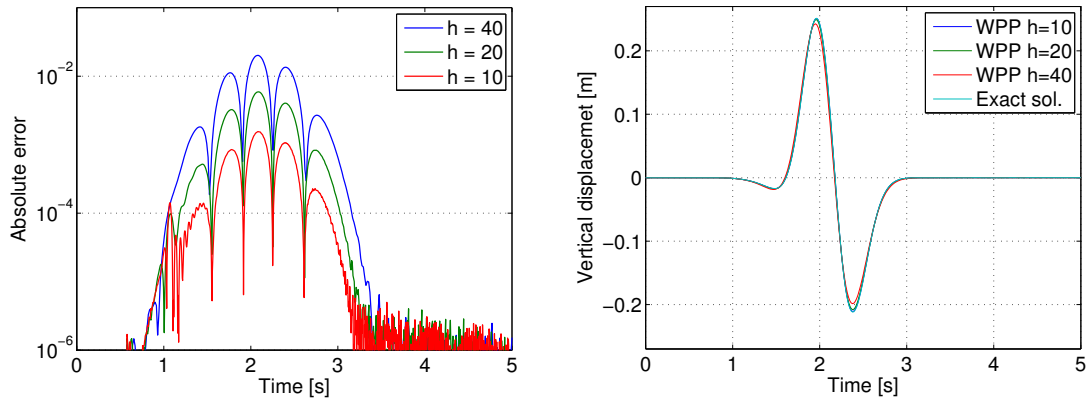


Figure 1.3: To the left: Absolute error for three different grid spacings. To the right: *WPP* solutions for grid spacings $h = 10, 20, 40$ together with the exact solution from [5].

- * Setting `nx` to 251 to be consistent with $h=40$
- * Setting `ny` to 251 to be consistent with $h=40$
- * Setting `nz` to 126 to be consistent with $h=40$

```
-----
Max no of pts in one processor is 574592
Estimated maximum memory usage per process is 99 Mbytes
-----
```

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

```
Block 1 will cover region: I(1, 251) J(1,251) K(1,126)
```

```
Running WPP on 16 processors...
Writing output to directory: .
    With name: wpp
```

```
-----
Making Output Directory: .
```

```
... Done!
```

```
=====
Starting program wpp
```

```
-----
WPP Version 1.0
Copyright (C) 2006 The Regents of the University of California
```

```
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 Apr 17 11:37:47 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/lib  
-----
```

=====

Using the following data :

Start Time = 0 Goal Time = 5
Number Steps = 328 dt: 0.0152439
Internal order of accuracy: 2
Time discr. is 2nd order central method

```
-----  
Total seismic moment (M0): 0 Nm  
-----
```

```
----- Material properties ranges -----  
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  
1e+09 Pa <= mu <= 1e+09 Pa  
1e+09 Pa <= lambda <= 1e+09 Pa  
-----
```

dt/h = 0.000381098

Time step 1 t = 0

Time step 101 t = 1.52439

Time step 201 t = 3.04878

Time step 301 t = 4.57317

Writing BINARY SAC Files, of size 329:

s3.201.126.1.x

s3.201.126.1.y

s3.201.126.1.z

Writing BINARY SAC Files, of size 329:

s1.151.126.1.x

s1.151.126.1.y

s1.151.126.1.z

Writing BINARY SAC Files, of size 329:

s4.226.126.1.x

s4.226.126.1.y

s4.226.126.1.z

Writing BINARY SAC Files, of size 329:

s2.176.126.1.x

```
s2.176.126.1.y
s2.176.126.1.z
Time step 328 t = 5
=====
Wave Propagation Program (WPP) Finished!
=====
```

Chapter 2

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 2.1. *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.

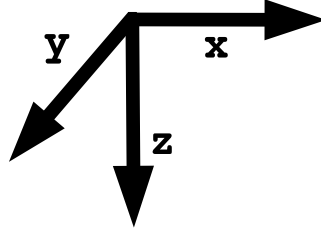


Figure 2.1: *WPP* uses a right handed coordinate system with the z -axis pointing downwards.

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

$$0 \leq x \leq x_{max}, \quad 0 \leq y \leq y_{max}, \quad 0 \leq z \leq z_{max}. \quad (2.1)$$

The grid command line in the input file specifies the extent of the computational domain. This release of *WPP* uses a uniform grid spacing h in all three coordinate directions. 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: $n_x=61$, $n_y=41$, and $n_z=21$, 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

$$n_x = (\text{int})1.5 + x/h, \quad (2.2)$$

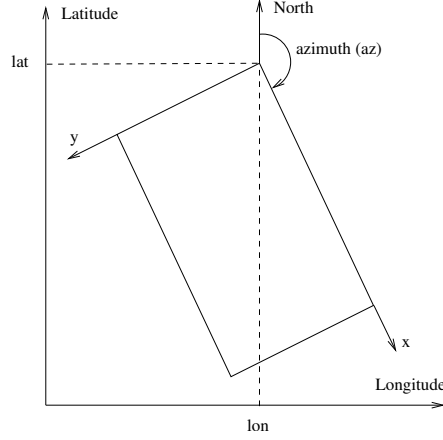


Figure 2.2: Geographical coordinates in *WPP*.

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

$$x = (n_x - 1)h. \quad (2.3)$$

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
```

Once again, same size grid spatially, but here the grid spacing will be computed as

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

Note that no rounding needs to take place in this case, since h is real. Given this value of h , n_y and n_z are computed using formulas corresponding to (2.2) giving $n_y=34$ and $n_z=67$, for a total of 227,800 grid points. Again, the extents in the y and z -directions are adjusted corresponding to (2.3). The syntax for the grid command is given in Section 6.1.

2.1 Geographic coordinates

WPP supports geographic coordinates as an alternative way of specifying spatial locations, see Figure 2.2. The location of the Cartesian coordinate 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)

$$\phi = lat + \frac{x \cos(\alpha) - y \sin(\alpha)}{M}, \quad \alpha = az \frac{\pi}{180}, \quad (2.4)$$

$$\theta = lon + \frac{x \sin(\alpha) + y \cos(\alpha)}{M \cos(\phi\pi/180)}, \quad (2.5)$$

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 3

Sources, time-functions and grid sizes

3.1 Sources and time-functions in *WPP*

WPP solves the elastic wave equation in displacement formulation,

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

where ρ is the density, $\mathbf{u}(\mathbf{x}, t)$ is the displacement vector, \mathcal{T} is the stress tensor and \mathbf{n} is the normal vector to the $z = 0$ plane. The computational domain Ω is the box shaped region (2.1). A stress-free boundary condition is imposed on the $z = 0$ boundary and first order Clayton-Engquist non-reflecting boundary conditions are imposed on the other five sides of the computational domain.

There are six linearly independent invariants in the solution of the elastic wave equation: translations along the three coordinate directions and rotation along the three coordinate axes. The projection command (Section 6.6.2) 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 can be disabled by setting the projection interval to 0, but this may cause the solution to spuriously drift after long times. 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
```

The forcing term \mathbf{F} consists of a sum of point force and point moment terms. For a point forcing we have

$$\mathbf{F}(\mathbf{r}, t) = g(t, t_0, \omega) F_0 \begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix} \delta(\mathbf{r} - \mathbf{r}_0),$$

where F_0 is the amplitude, $\mathbf{r}_0 = (x_0, y_0, z_0)$ 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 ($F_0 * F_x, F_0 * F_y$, etc.).

For a point moment tensor we have

$$\mathbf{F}(\mathbf{r}, t) = g(t, t_0, \omega) M_0 \nabla \cdot (\mathcal{M} \delta(\mathbf{r} - \mathbf{r}_0)), \quad \mathcal{M} = \begin{pmatrix} M_{xx} & M_{xy} & M_{xz} \\ M_{xy} & M_{yy} & M_{yz} \\ M_{xz} & M_{yz} & M_{zz} \end{pmatrix}.$$

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 using any (finite) number of source commands. A simple example 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 (their default values).

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 ($\lim_{t \rightarrow -\infty} g(t, t_0, \omega) = 0$). 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, and BruneSmoothed functions tend to one ($\lim_{t \rightarrow \infty} g(t, t_0, \omega) = 1$). The time derivative of $g(t, t_0, \omega)$, is often referred 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 used 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 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 $t = t_0$ with exponentially decaying tails for $t < t_0$. Hence t_0 must be positive and of the order $\mathcal{O}(1/\omega)$ to avoid incompatibility problems with the initial conditions. We recommend choosing t_0 such that $g(0, t_0, \omega) \leq 10^{-8}$ for these functions.

3.1.1 Gaussian

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

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 3.1.

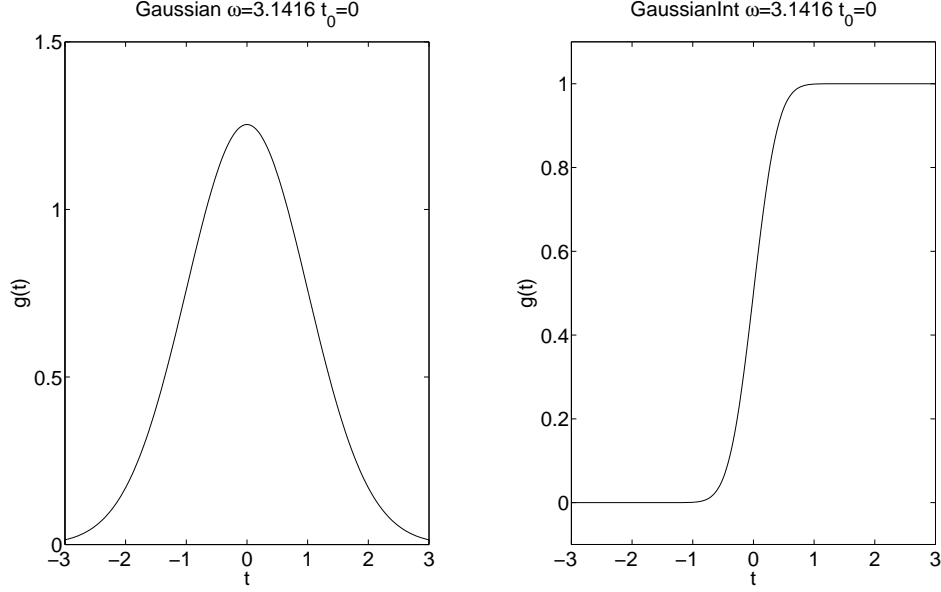


Figure 3.1: Gaussian (left) and GaussianInt (right) with $\omega = \pi$ and $t_0 = 0$.

3.1.2 GaussianInt

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

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

GaussianInt is the time-integral of the Gaussian. A plot of the GaussianInt time-function is shown in Figure 3.1.

3.1.3 Ricker

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

A plot of the Ricker time-function is shown in Figure 3.2.

3.1.4 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. The RickerInt function is often used in seismic exploration applications, since it does not lead to any permanent displacement. A plot of the RickerInt time-function is shown in Figure 3.2.

3.1.5 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 3.3.

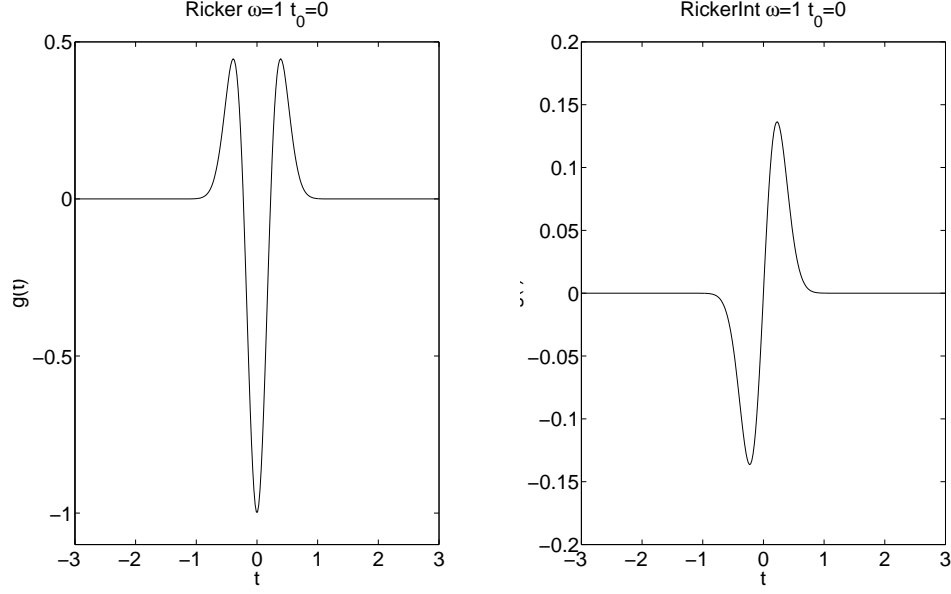


Figure 3.2: Ricker (left) and RickerInt (right) with $\omega = 1$ and $t_0 = 0$.

3.1.6 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 3.3.

3.1.7 Ramp

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

A plot of the Ramp time-function is shown in Figure 3.4.

3.1.8 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 3.4.

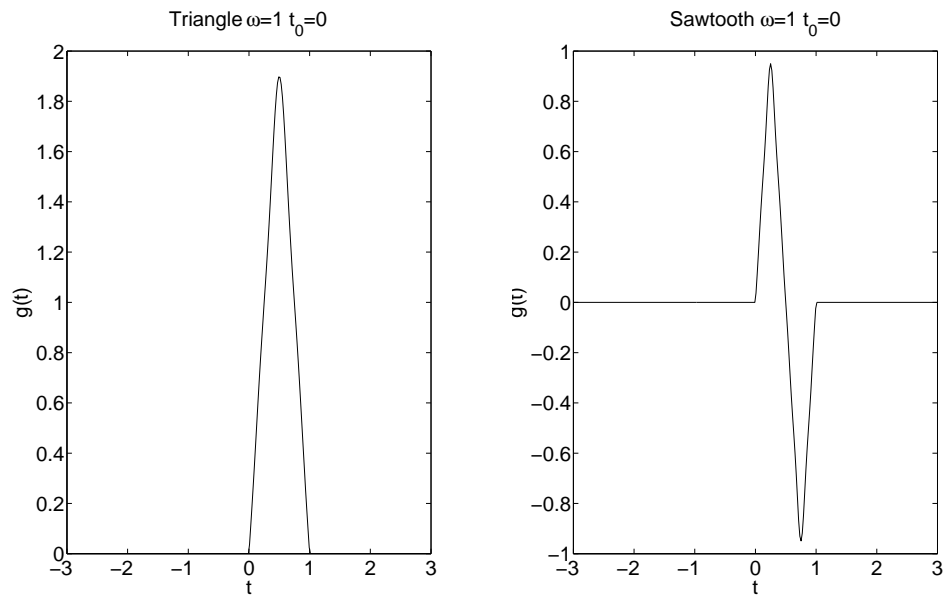


Figure 3.3: Triangle (left) and Sawtooth (right) with $\omega = 1$ and $t_0 = 0$.

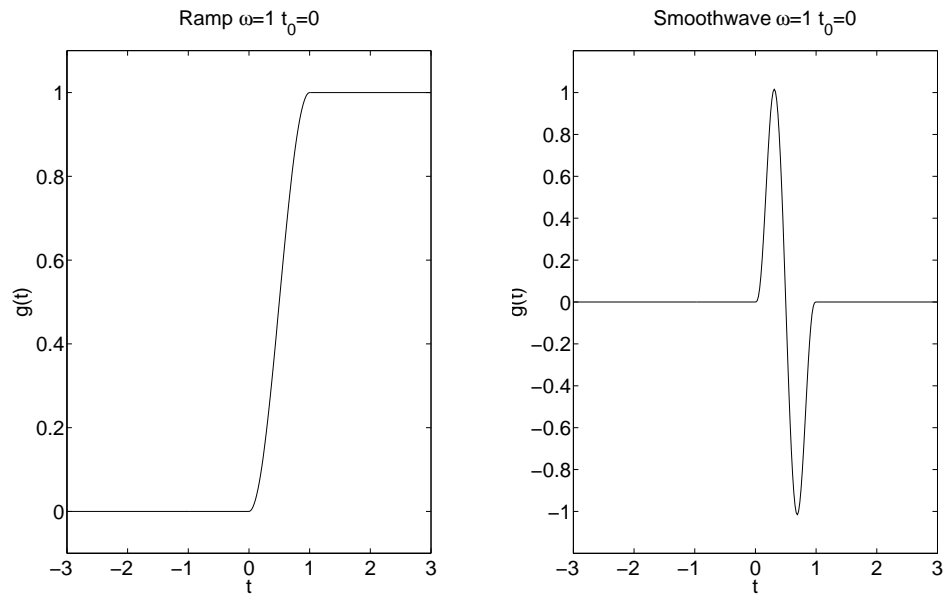


Figure 3.4: Ramp (left) and Smoothwave (right) with $\omega = 1$ and $t_0 = 0$.

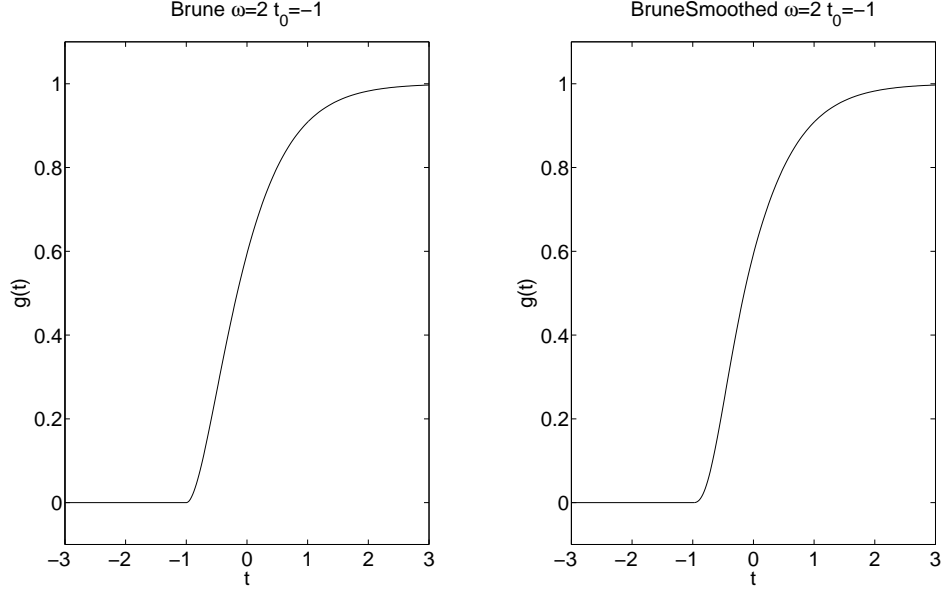


Figure 3.5: Brune (left) and BruneSmoothed (right) with $\omega = 2$ and $t_0 = -1$.

3.1.9 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.

3.1.10 BruneSmoothed

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

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

The parameter is fixed to $x_0 = 2.31$. Plots of the Brune and BruneSmoothed time-functions are shown in Figure 3.5.

3.1.11 VerySmoothBump

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

A plot of the VerySmoothBump time-function is shown in Figure 3.6.

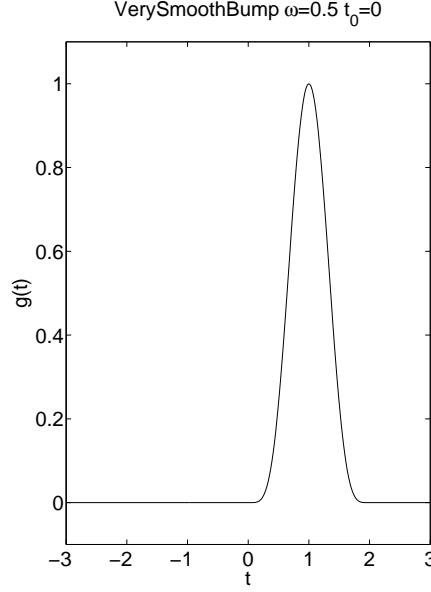


Figure 3.6: VerySmoothBump with $\omega = 0.5$ and $t_0 = 0$.

3.2 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 either, because both the computational demand and the memory requirements increase with decreasing grid size.

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 temporal frequency of the wave. Hence the number of grid points per wave length equals $\min \lambda/h$, which is given by

$$P = \frac{\min V_s}{h f}. \quad (3.1)$$

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 receiver.

In formula (3.1), $\min V_s$ 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 $200 \times 200 \times 100$ km, with a free surface on the top and non-reflecting boundary conditions on all other sides. The source is lo-

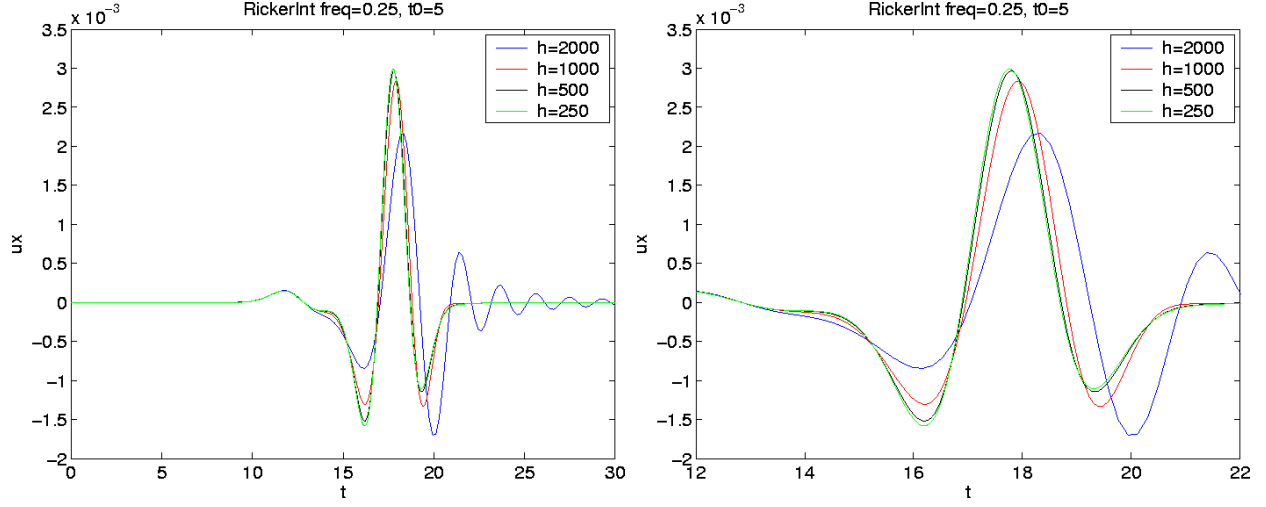


Figure 3.7: The x -component of the displacement due to a source with a RickerInt time function with frequency parameter $\text{freq}=0.25$ on four different grids. Enlarged view to the right.

cated 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/m³. 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 3.7 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 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 $h = 2000$ m 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 $f = \text{freq}$ and $h = 1000$ in formula (3.1),

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

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 3.8. 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 ($\text{freq}=0.25$) in the RickerInt time function.

Figure 3.9 shows the results from computing with the Ricker (3.1.3) 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.

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

Table 3.1: Approximate grid points per wavelength (P) for a given grid spacing (h)

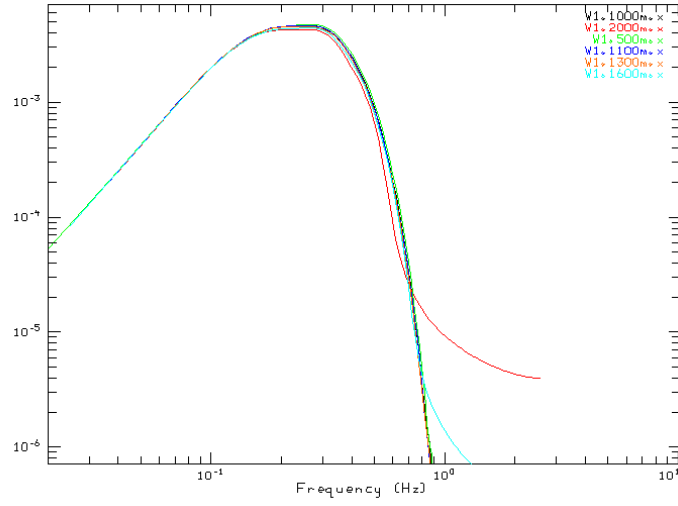


Figure 3.8: The Fourier transform of the time signal for different grid spacings.

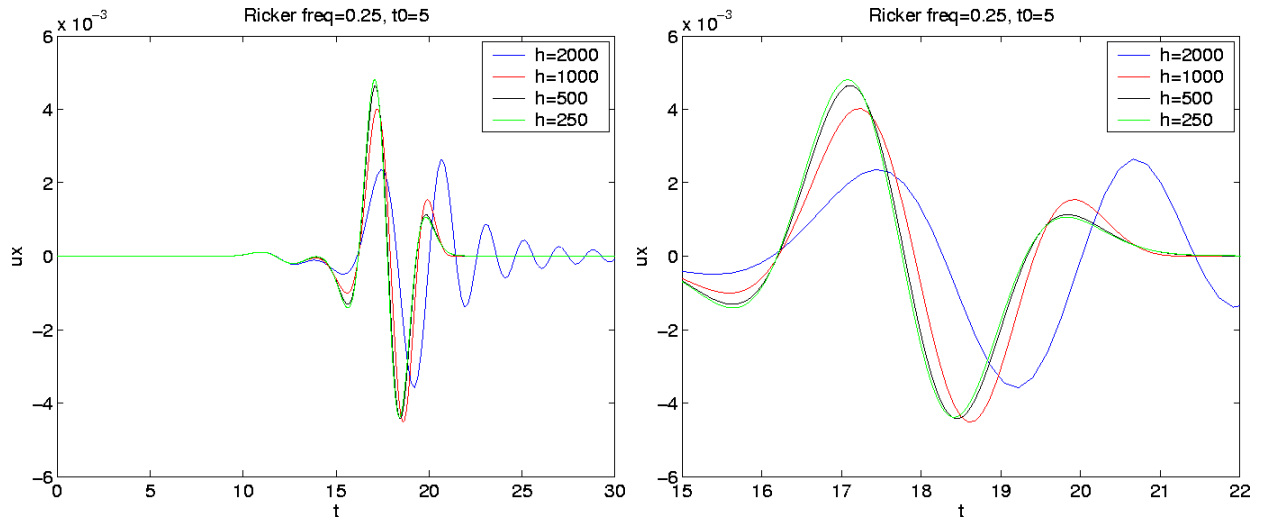


Figure 3.9: The x -component of the displacement due to a source with a Ricker time function with frequency parameter $\text{freq}=0.25$ on four different grids. Enlarged view to the right.

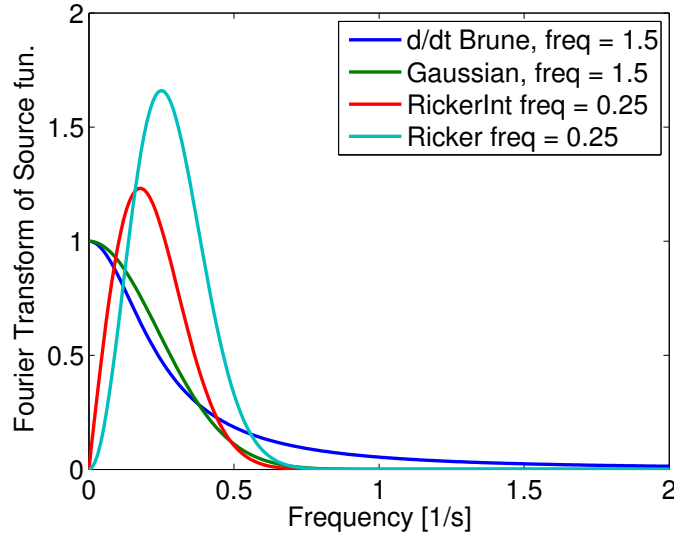


Figure 3.10: Absolute values of the Fourier transforms of the Gaussian (blue), Ricker (red), and the derivative of the Brune (black) time-functions. Here `freq=1.5` for the Gaussian and the derivative of the Brune function, and `freq=0.25` for the Ricker and RickerInt.

Figure 3.10 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:

$$f \approx \begin{cases} \text{freq}, & \text{for Ricker and RickerInt,} \\ \text{freq}/(2\pi), & \text{for Brune, BruneSmoothed, Gaussian, and GaussianInt.} \end{cases} \quad (3.2)$$

The plots in Figure 3.10 were made with frequency parameter `freq=0.25` for the Ricker and RickerInt functions and frequency parameter `freq=1.5` for the Gaussian and $d/dt(\text{Brune})$ functions. Furthermore, Figure 3.10 illustrates that the Fourier transform of the Ricker, RickerInt and Gaussian functions decay exponentially for large frequencies, but that 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 (3.1.1), GaussianInt (3.1.2) and the Brune (3.1.9) 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 3.11, 3.12, and 3.13 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 functions. We conclude that the grid size $h = 1000$ is adequate for representing the solution on the grid. If we use formula (3.1) with the frequency scaling (3.2) and $h = 1000$, we obtain

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

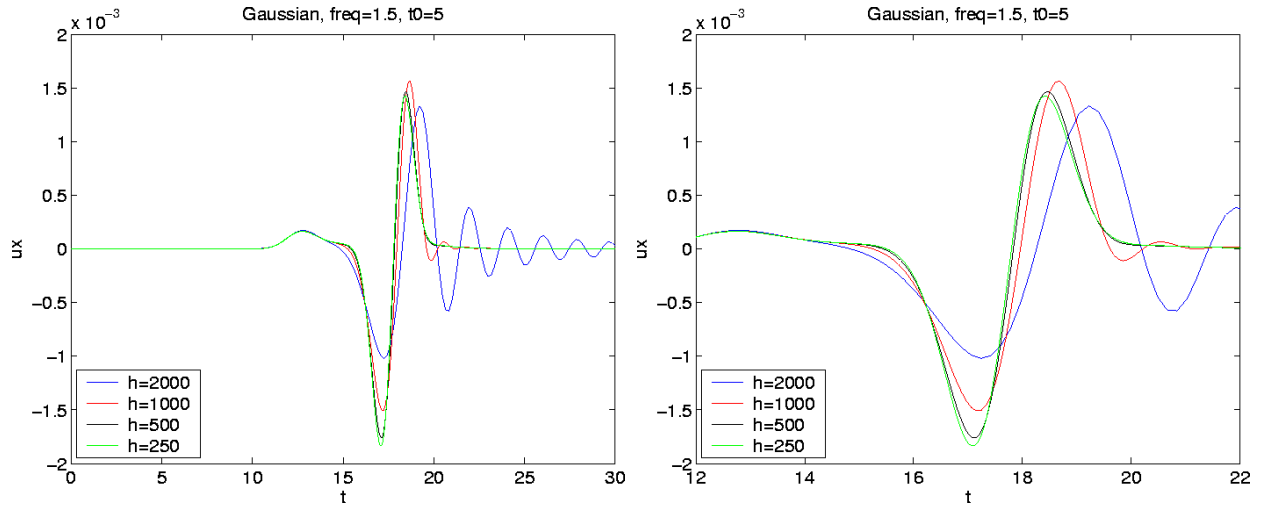


Figure 3.11: The x -component of the displacement due to a source with a Gaussian time function with frequency parameter $\text{freq}=1.5$ on four different grids. Enlarged view to the right.

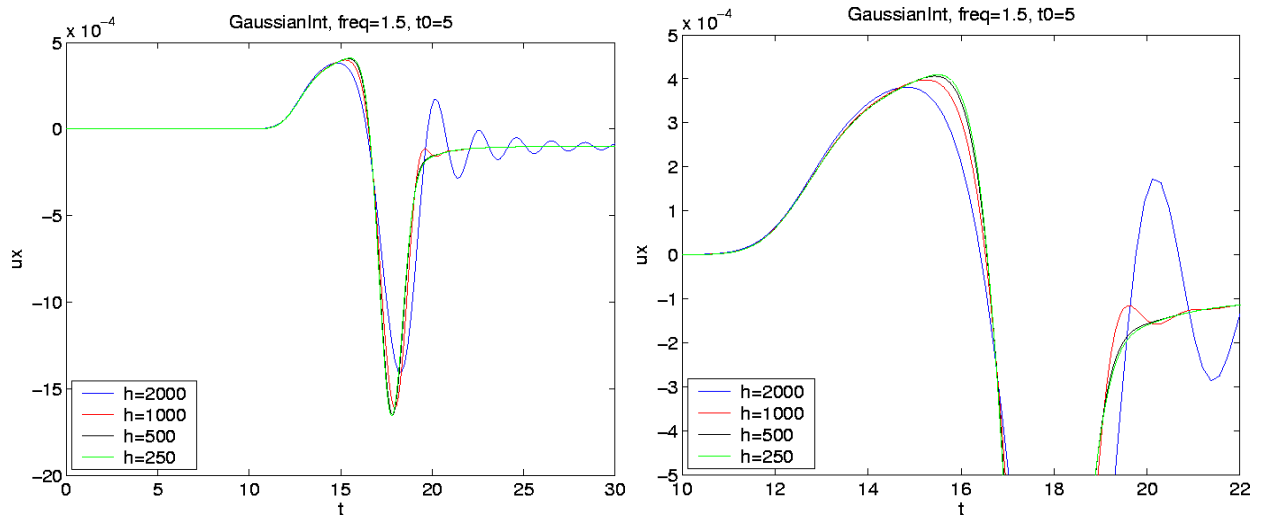


Figure 3.12: The x -component of the displacement due to a source with a GaussianInt time function with frequency parameter $\text{freq}=1.5$ on four different grids. Enlarged view to the right.

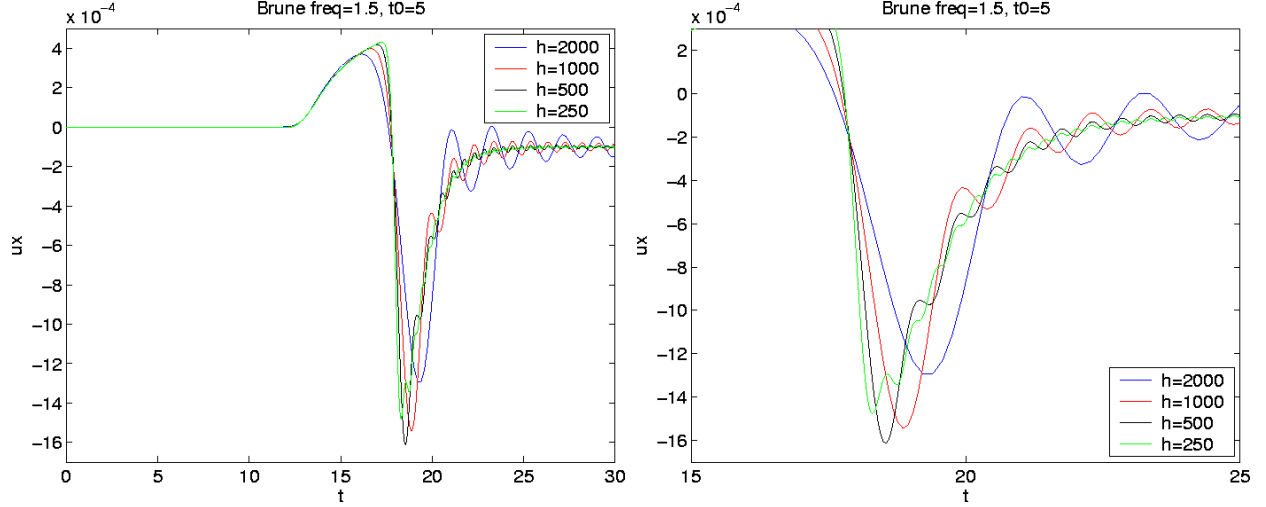


Figure 3.13: The x -component of the displacement due to a source with a Brune time function with frequency parameter $\text{freq}=1.5$ on four different grids. Enlarged view to the right.

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

3.2.1 Lamb's problem revisited

We now compute solutions to Lamb's problem in a material with $V_p = \sqrt{3} \text{ km/s}$, $V_s = 1 \text{ km/s}$ and the density 1000 kg/m^3 . The solution is forced downward with amplitude $\text{fz}=5 \times 10^{13} \text{ N}$ and with a time function centered at time $t_0=25 \text{ s}$. For various time functions the solution is recorded at receivers 10 and 50 km from the source. At the receivers the relative error

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

in the horizontal component is computed and plotted in Figure 3.14. 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 (3.1) Hence, a lower number of points per wave length corresponds to a higher value of freq .

From Figure 3.14 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 the numerical solution accumulates errors as it 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 [3] 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 achieved 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.

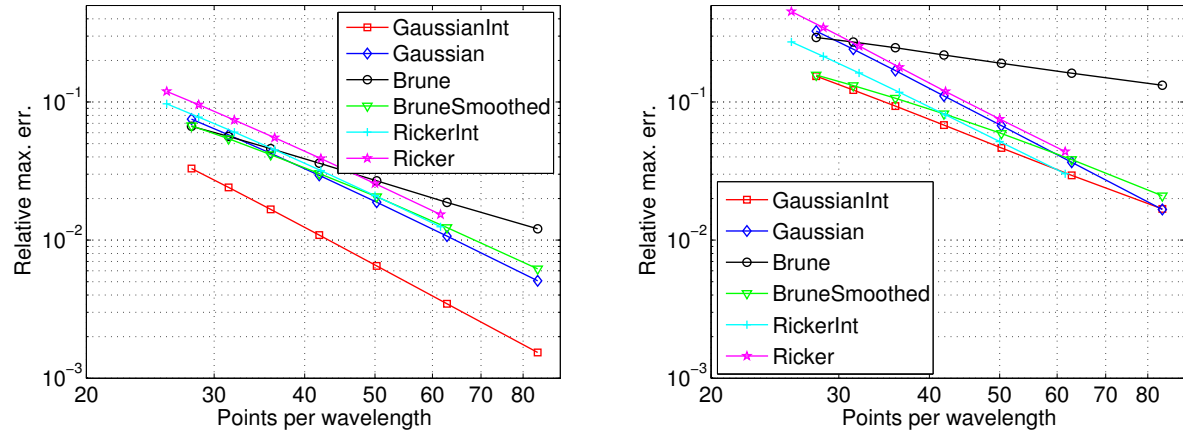


Figure 3.14: Relative errors for different source functions 10 (left) and 50 km (right) from the source. For the Brune time function the error decays much slower than for the other time functions.

Chapter 4

The material model

In *WPP* the material model 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 6.3.2), the vfile command (see Section 6.3.3), the etree command (see Section 6.3.4) 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 6.3.1.

Note: The order within the material commands (block, vfile, and etree) does matter (unlike all other commands) in that the priority of the command / material increase towards the end of the input file. Hence, a material command in the input file is overridden by subsequent material commands.

4.1 The block command

The block command can be used to specify properties in rectangular volumes (sub-regions), either with constant properties or with linear vertical gradients. 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=25000 r=1000 \
      x1=4000 x2=8000 y1=4000 y2=10000 z1=10000 z2=70000
```

To the left in Figure 4.1 an image slice of V_p at $x = 50\text{km}$ is displayed.

The following example combines several block commands used to generate the material model displayed to the right in Figure 4.1:

```
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
```

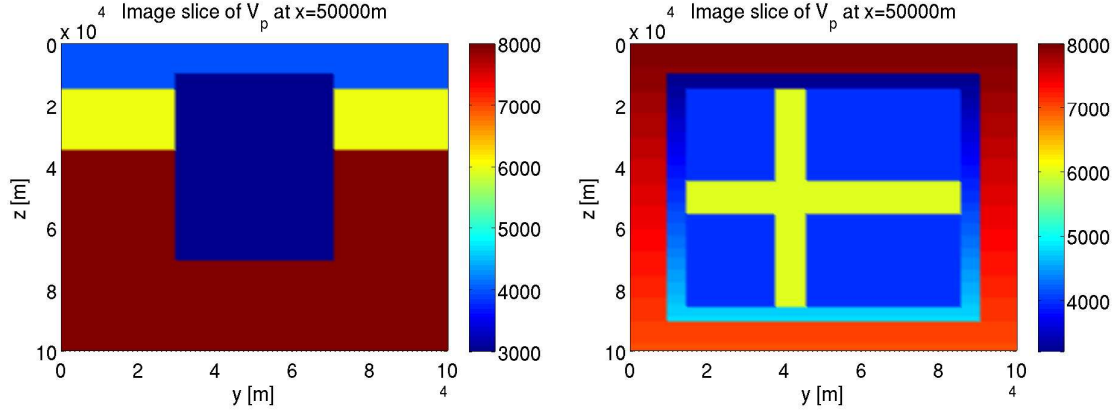


Figure 4.1: Examples of material models specified with the block command.

```
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
```

4.1.1 Modeling water with blocks

In sea water the density is 1000 kg/m^3 $V_p \approx 1500 \text{ m/s}$ and $V_s = 0$. With the block command it is easy to include a layer of water at the top of the surface, 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=1 curlCFL=0.1
```

The curl of the 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 air is present.

The curlcurl dissipation coefficient d_2 is not taken into account when the time step is computed. Therefore, a too large value can lead to instability. The input value is given as fractions of the maximum allowed value by the stability condition. Therefore the value 1 should give maximum damping, however in practice the maximum limit is closer to 0.1. This is because the maximum limit is computed for the dissipation term only. Effects from interaction with the terms of the basic scheme are not considered.

4.2 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 the grid points. These files are not self describing, and thus do not contain any header metadata information, only the raw binary floating point data for each field point. Assuming `vp.flt`, `vs.flt` and `r.flt` contains values for the P-velocity, the S-velocity and the density in 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 above lines will cause *WPP* to read the floating point numbers in the files in a binary raster format. For example, the P-wave velocity is read from the vfile 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 i (x) and k (z) directions but in decreasing order in the j (y) direction, this is because the vfiles are specified (for historical reasons) in a left handed coordinate system, where the y -direction points in the opposite direction compared to *WPP*. If $i1$, $j1$, $k1$, $i2$, $j2$ and $k2$ are not explicitly stated they are defaulted to $i1=j1=k1=1$, $i2=nx$, $j2=ny$, $k2=nz$ (the full computational domain). If they (as in the example below) are specified:

```
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
```

WPP will read the files to specify the material model in a box shaped sub-region of the computational grid. The above lines will specify the model according to the velocities and density in the binary files `vp.flt`, `vs.flt` and `r.flt` over the domain from grid point $i=51$ to $i=61$ in the x -direction, from $k=1$ to $k=11$ in the z -direction, and from $j1 = 1$ to $j2 = ny$ in the y -direction.

There is a one-to-one mapping between the points in the vfile and the computational grid points within the specified region. Thus, the number of velocity values in the vfile must equal $(i2-i1+1) \times (j2-j1+1) \times (k2-k1+1)$. Since a floating point number is 4 bytes long, the size of the vfile must contain exactly $4 \times (i2-i1+1) \times (j2-j1+1) \times (k2-k1+1)$ bytes. If the vfile is smaller than that, the *WPP* will exit with an error message.

It is however allowed for the vfile to contain more grid points than the computational grid. For example, consider the i index and suppose 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 $i2=100$. This will be the normal case. If the vfile represents a fraction of the computational grid, the values of i could be something like $i1=20$ and $i2=50$. If the vfile represents a region larger than the computational grid, then the values of i could be $i1=-50$ and $i2=150$. In that case, the data in the vfile corresponding to $-50 \leq i \leq 0$ and $101 \leq i \leq 150$ are ignored.

Note that it is possible to specify the sub-domain boundaries using the `x1`, `x2`, `y1`, `y2`, `z1`, `z2` options, however this is not advised because roundoff may cause problems when the physical coordinates are converted to grid point indices, causing the number of points to be "off by one". If physical sub-domain boundaries are used, we strongly recommend verifying the material model before starting a simulation, for example using the `image` command (see Section 5.3) to output a couple of cross sections through the computational domain to check that the P- and S-wave velocities as well as the density look reasonable.

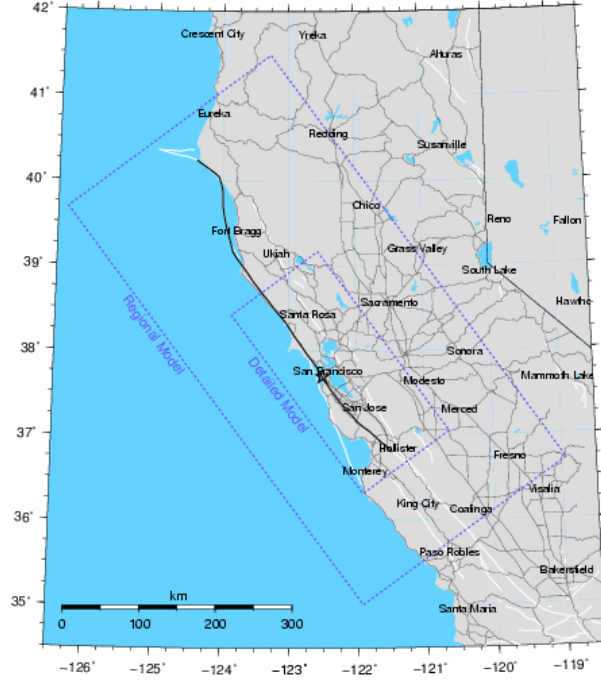


Figure 4.2: The geographical location where the efile model=SF is valid.

4.3 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 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.

Currently we only support reading efile data from the central California velocity model provided by the USGS (see Figure 4.2). Internally in the code we use the formula (2.4)-(2.5) 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 is used to query the etree database and then populate V_p , V_s and ρ with the values returned. It is important to note the bounds of the geographical region in the database, if queried outside the software returns -999 for the material property values.

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

```
grid x=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 where the domain is larger than what the efile covers, a block command can be used to initialize the material data which falls outside of the efile block:

```
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
```

The final case for mapping an efile to a grid, is if we want to have the efile map to a sub-region of the grid.

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 4.1: Data for the central California velocity model.

```
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 will only span the region between grid point 10 and 90 in the x direction, everywhere else the material properties will come from the block command.

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

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

4.3.1 Modeling water with Etree models

When water is present in an etree database, the efile command will default to the `water=noshear` option to assign a zero shear velocity to the grid points in the water. Other water options are available for historical reasons, see Section 6.3.4. 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=1 curlCFL=0.1
```

For numerical reasons it is required that the depth of the water be at least two grid points. If this is not the case, *WPP* will assign water properties to the grid point below the water surface.

Chapter 5

Output options

5.1 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.

5.2 Time-history at a point: the `sac` command

The Seismic Analysis Code (*SAC*[2]) 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`
- `s1.I.J.K.y`
- `s1.I.J.K.z` (note: z is positive down)

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.

5.3 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. 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 6.5.3. 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.

The images files are written in a binary format. WPP stores the grid size h (a single precision floating point number) and the dimensions of the image (two integers) first in the file. Then follows the image as single precision floating point numbers stored in column order. The Matlab statements

```
fd = fopen('picturefile.cycle=100.x=1000.div','r');  
h = fread(fd,1,'float');  
ni = fread(fd,1,'int');  
nj = fread(fd,1,'int');  
u = fread(fd,[ni nj],'float');  
fclose(fd);
```

read the image file in the example above into the Matlab matrix u of size $ni \times nj$. 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.

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 maxima are performed after each time step.

5.4 Volumetric data: the image3d command

Warning: Saving volumetric data results in huge files which require specialized post processing software to read.

The `image3d` command may be used to output 3-D data for different solution variables at a given time (step) intervals. Data is saved in the compressed “brick-of-wavelet” format using Mark Duchaineau’s Bow library, see Section 7.7. The image plane can be specified. These files can be manipulated using the `bof/bow/bob` tools, from the LibGen library.

Here is an example using the `image3d` command line:

```
grid x=500e3 y=200e3 z=40e3 lat=41.10 lon=-123.60 az=144.0 nx=500
fileio path=mydir/
image3d cycle=50 mode=veldiv
```

The `image3d` command generates one bow file per processor. If this simulation ran on four processors, the following files would be generated after time step 50:

```
mydir/image3d00000/proc00000/step000050_p00000.bow
mydir/image3d00001/proc00001/step000050_p00001.bow
mydir/image3d00002/proc00002/step000050_p00002.bow
mydir/image3d00003/proc00003/step000050_p00003.bow
mydir/image3d-bowdump.txt
```

The last file, `image3d-bowdump.txt`, is the header file for the meta data in the run.

The options `logifymin` and `logifymax` can be used to change the logarithmic range to which data is converted before being compressed and written to disk. The resulting file size (and the quality of the saved data) is also controlled by the `compressionFactor` option. For example,

```
image3d cycleInterval=50 mode=velcurl file=more compressionFactor=0.0001
image3d cycleInterval=50 mode=velcurl file=less compressionFactor=0.01
```

The first line provides the default accuracy, which is the maximum available accuracy in the compression library. The second line uses a compression factor of 0.01, which provides less accuracy and smaller files. The `compressionFactor` is relative to the minima and maxima of the range of the compressed variable. Using the default settings of the `image3d` command, we have seen a file size reduction for the `velcurl/veldiv` image modes of 12-22 times.

Note: No compression takes place if `compressionFactor=1.0`.

5.5 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 6

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, 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.

6.1 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.

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

6.2 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.

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

6.3 Specifying the model [required]

6.3.1 specifying a box shaped sub-region

In general, a box shaped sub-region is specified using either i,j,k grid points or x,y,z domains. These options are used in conjunction with the block, vfile and efile commands.

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

6.3.2 basic 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.

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

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.

6.3.3 velocity file (vfile)

Required parameters:

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

These command options may be used in conjunction with the box shaped sub-region options. (Table 6.3.1)

Option	Description	Type	Units
type	input file type (vp, vs, or r)	string	none
file	name of input vfile	string	none
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
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`.

6.3.4 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 6.3.1).

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	double	m/s	none
vpmin	minimum threshold for the p velocity in solids	double	m/s	none
water	set to noshear, poisson, or seafloor	string	none	noshear
query	type of query to perform	string	none	MAXRES
resolution	for FIXEDRES, the resolution to sample the data	double	m	h
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).

6.4 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]).

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	the xx moment tensor component	float	none	0.0
Myy	the yy moment tensor component	float	none	0.0
Mzz	the zz moment tensor component	float	none	0.0
Mxy	the xy moment tensor component	float	none	0.0
Mxz	the xz moment tensor component	float	none	0.0
Myz	the yz moment tensor component	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, and BruneSmoothed.

6.5 Specifying output

6.5.1 commands to control stdout (fileio)

Required parameters: none

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

6.5.2 SAC files (sac)

Required parameters:

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

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 occurred: YYYY/MM/DD	int/int/int	none	date of run
eventTime	time the event occurred: hours:minutes:seconds	int:int:int	none	time of run

6.5.3 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)

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

Options for mode include:

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
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 [<i>only available with efile input</i>]

6.5.4 3D volumetric data (image3d)

Required parameters:

- Image mode (see image mode command options)
- Timing interval (time, timeInterval, cycle or cycleInterval)

Option	Description	Type	Units	Default
time	simulation time to output volumetric image, will be closest depending on dt taken	float	s	none
timeInterval	simulation time interval to output series of volumetric images	float	s	none
compressionFactor	option to vary how much accuracy will be preserved during compression	float	none	0.0001
cycle	time-step cycle to output volumetric image	int	none	none
cycleInterval	time-step cycle interval to output a series of volumetric images	int	none	none
file	file name header of volumetric image	string	none	image3d
mode	specifies which field	string	none	none
topdirmod	option to change how many top level directories are written	int	none	100
logifymin	option to set min value to convert field to before compression	float	none	1e-8
logifymax	option to set max value to convert field to before compression	float	none	1e+8
sample	interval on which to sample the data in the x,y and z direction	int	none	1
verbose	flag to output more info on write	int	none	0

Note: Options for mode include all image modes except topo - it is only for making slices in the xy plane.

6.5.5 saving and restoring the simulation (restart)

Required parameters: none

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

6.6 Specifying numerical simulation controls

6.6.1 curl-curl damping command options (in water only)

Required parameters: none

Option	Description	Type	Default
curlcurl	Turn on the curlcurl-damping	int	0
curlCFL	A normalized version of coefficient d_2 above	float	0

6.6.2 Controlling solid body motion (projection)

Required parameters: none

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

Chapter 7

Building WPP

WPP source can be downloaded from:

<http://www.llnl.gov/CASC/serpentine/software.html>

Note: A really detailed example for Mac OS X build and install can be found in section 7.8.

7.1 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

7.2 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 powerful scripting language written in C
- 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.

7.3 Directory structure

For more detailed information about the files and directories in the WPP source, please read `wpp-v1.0/README.txt` after you've untared your tarball.

```
shell> gunzip wpp-v1.0.tar.gz; tar xfv wpp-v1.0.tar
```

This will create a directory named `wpp-v1.0` which will contain several files and subdirectories:

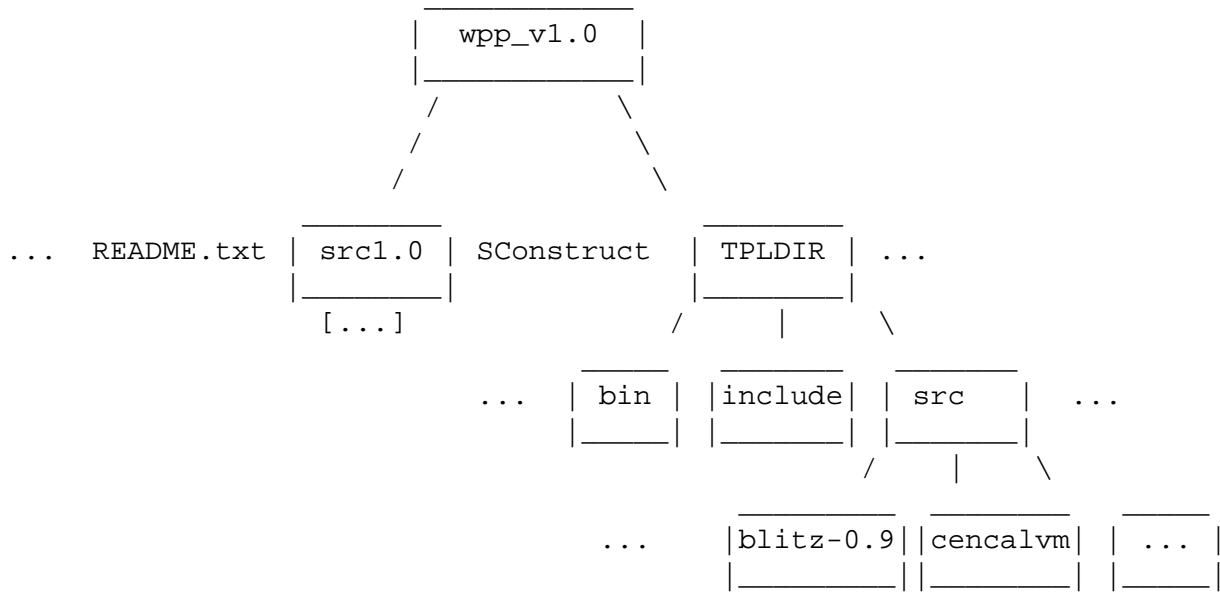
- `INSTALL.txt` Information about how to build WPP
- `KNOWN-BUGS.txt` List of known problems or issues with porting or other bugs
- `README.txt`
- `TPL.txt` Details about building the third party libraries WPP requires
- `configs` Directory which contains build configuration files
- `src1.0` C++ and 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

7.4 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` A powerful library which contains the array class WPP uses
- `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 any case, we will refer to this common directory as `TPLDIR`.



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.

7.4.1 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
```

7.4.2 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
```

7.5 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 7.6 and 7.7 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
```

7.5.1 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> scon
```

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

7.6 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`).

7.6.1 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}
```

7.6.2 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

7.6.3 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
```

7.7 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.

7.7.1 Building BOW

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

```
set ZINCLUDE = .

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

7.8 Detailed build example for Mac OS X

This describes how to install WPP and its required libraries in a directory `/Users/daniel/wpp` using C shell (`csh`). It may require you to download and install `g77`, (see <http://hpc.sourceforge.net/>). Also please note this example differs slightly from the directory structure described previously.

```
[~/wpp] % pwd
/Users/daniel/wpp
[~/wpp] % csh
[~/wpp] % ls
blitz-0.9.tar.gz  g77-bin.tar.gz  mpich2-1.0.5p4.tar.gz
scons-0.97.tar.gz  wpp-v1.0.tar.gz
```

Start by installing `g77` (you must be able to `sudo` as root to do this)

```
[~/wpp] % gunzip g77-bin.tar.gz
[~/wpp] % sudo tar -xvf g77-bin.tar -C /.
Password:
usr/local/
usr/local/bin/
usr/local/bin/cpp
usr/local/bin/g77
...
...
```

```
[~/wpp] % setenv PATH /usr/local/bin:$PATH
[~/wpp] % g77
g77: no input files
[~/wpp] % which g77
/usr/local/bin/g77
```

Now install scon

```
[~/wpp] % gunzip scon-0.97.tar.gz
[~/wpp] % tar -xf scon-0.97.tar
[~/wpp] % cd scon-0.97
[~/wpp/scon-0.97] % python setup.py install --prefix=/Users/daniel/wpp/scon
...
...

[~/wpp/scon-0.97] % setenv PATH /Users/daniel/wpp/scon/bin:$PATH
[~/wpp/scon-0.97] % cd ..
[~/wpp] % which scon
/Users/daniel/wpp/scon/bin/scon
```

Install WPP's libraries in /Users/daniel/wpp/wpplibs, start with mpich2

```
[~/wpp] % mkdir wpplibs
[~/wpp] % mv mpich2-1.0.5p4.tar.gz wpplibs/
[~/wpp] % cd wpplibs/
[~/wpp/wpplibs] % gunzip mpich2-1.0.5p4.tar.gz
[~/wpp/wpplibs] % tar -xf mpich2-1.0.5p4.tar
[~/wpp/wpplibs] % cd mpich2-1.0.5p4
[~/wpp/wpplibs/mpich2-1.0.5p4] % setenv WPP_TPL /Users/daniel/wpp/wpplibs
[~/wpp/wpplibs/mpich2-1.0.5p4] % setenv CC gcc
[~/wpp/wpplibs/mpich2-1.0.5p4] % setenv CXX g++
[~/wpp/wpplibs/mpich2-1.0.5p4] % setenv F77 g77
[~/wpp/wpplibs/mpich2-1.0.5p4] % ./configure --prefix=${WPP_TPL} \
--disable-f77 --disable-f90 | & tee configure.log
[~/wpp/wpplibs/mpich2-1.0.5p4] % make |& tee make.log
[~/wpp/wpplibs/mpich2-1.0.5p4] % make install | & tee install.log
[~/wpp/wpplibs/mpich2-1.0.5p4] % setenv PATH ${WPP_TPL}/bin:$PATH
[~/wpp/wpplibs/mpich2-1.0.5p4] % which mpd; which mpirun;
/Users/daniel/wpp/wpplibs/bin/mpd
/Users/daniel/wpp/wpplibs/bin/mpirun
[~/wpp/wpplibs/mpich2-1.0.5p4] % mpd &
[1] 8732
[~/wpp/wpplibs/mpich2-1.0.5p4] % mpdtrace
datan
[~/wpp/wpplibs/mpich2-1.0.5p4] % make testing
```

Here some tests failed (16 to be precise) but it did not seem to matter for the WPP installation. Now let's install blitz++:

```
[~/wpp/wpplibs/mpich2-1.0.5p4] % cd ..
[~/wpp/wpplibs] % cd ..
[~/wpp] % mv blitz-0.9.tar.gz wpplibs/
[~/wpp] % cd wpplibs/
[~/wpp/wpplibs] % gunzip blitz-0.9.tar.gz
[~/wpp/wpplibs] % tar -xf blitz-0.9.tar
[~/wpp/wpplibs] % cd blitz-0.9
[~/wpp/wpplibs/blitz-0.9] %
[~/wpp/wpplibs/blitz-0.9] % ./configure --prefix=${WPP_TPL}
[~/wpp/wpplibs/blitz-0.9] % make lib
[~/wpp/wpplibs/blitz-0.9] % make install
[~/wpp/wpplibs/blitz-0.9] % make check-testsuite
```

All tests passed, woo hoo! **Finally we install WPP:** copy the file `macconfig.py` (see below) into `/Users/daniel/wpp/wpp-v1.0/configs/macconfig.py`

```
[~/wpp/wpplibs/blitz-0.9] % cd ..
[~/wpp/wpplibs] % cd ..
[~/wpp] % gunzip wpp-v1.0.tar.gz
[~/wpp] % tar -xf wpp-v1.0.tar
[~/wpp] % setenv WPPCONFIG /Users/daniel/wpp/wpp-v1.0/configs/macconfig.py
[~/wpp] % cd wpp-v1.0
[~/wpp/wpp-v1.0] % scons
```

If all works you should now see:

```
'''-.,_.,-'''-.,_.,=''''-.,_.,-'''-.,_.,='''''-.,_.,-'''-.,_.,='''
      WPP
      LIS
      in /Users/daniel/wpp/wpp-v1.0/optimize
'''-.,_.,-'''-.,_.,=''''-.,_.,-'''-.,_.,='''''-.,_.,-'''-.,_.,='''
```

Now we shall invoke the example from Section 1.2:

```
[~/wpp/wpp-v1.0] % cd tests/examples/
```

```
[wpp-v1.0/tests/examples] % cp ../../optimize/wpp .
[wpp-v1.0/tests/examples] % mpirun -np 4 ./wpp lamb.in
```

It might be a good idea to save the following lines into a file called `wppsetup.csh` then you can type `source wppsetup.csh` in `csh` when you want to run `wpp` using `mpirun` or if you want to recompile.

```
setenv PATH /Users/daniel/wpp/scons/bin:$PATH
setenv WPP_TPL /Users/daniel/wpp/wpplibs
setenv PATH ${WPP_TPL}/bin:$PATH
setenv CC gcc
setenv CXX g++
setenv F77 g77
setenv WPPCONFIG /Users/daniel/wpp/wpp-v1.0/configs/macconfig.py
```

Alternatively you could copy the following lines into your `~/.profile` file:

```
export PATH=/Users/daniel/wpp/scons/bin:$PATH
export WPP_TPL=/Users/daniel/wpp/wpplibs
export PATH=${WPP_TPL}/bin:$PATH
export CC=gcc
export CXX=g++
export F77=g77
export WPPCONFIG=/Users/daniel/wpp/wpp-v1.0/configs/macconfig.py
```

Save the text below as `macconfig.py` and change the line:

```
env.Replace(WPPLIBDIR = os.path.join(os.sep,
                                     'Users', 'daniel', 'wpp', 'wpplibs'))
```

to point to your library directory.

```
[wpp-v1.0/configs/] % cat macconfig.py
# -*- python -*-
import os
```

```
# Bring in information from the SConstruct file
Import('env')
```

```
# To locate your third party libraries built for wpp, set the "tool" directory
env.Replace(WPPLIBDIR = os.path.join(os.sep,
                                     'Users', 'daniel', 'wpp', 'wpplibs'))
```

```
# Set a different C++, C and Fortran compiler.
env.Replace(CXX      = os.path.join(env['WPPLIBDIR'], 'bin', 'mpicxx'))
env.Replace(CC       = os.path.join(env['WPPLIBDIR'], 'bin', 'mpicc'))
env.Append(FORTRAN   = os.path.join(os.sep, 'usr', 'local', 'bin', 'g77'))
```

```
# Set C and C++ compiler flags
env.Append(CCFLAGS = ' -DHAVE_MPI_CPP -Wno-long-double')
env.Append(CXXFLAGS = ' -DHAVE_MPI_CPP -Wno-long-double')
```

```
# Depending on which Fortran compiler you use, you may need this option
env.Append(F77FLAGS = " -fno-underscoring -fcase-upper")
env.Append(FORTRANFLAGS = env['F77FLAGS'])

# Any other system libraries that may be required
env.Append(EXTLIBS = 'g2c')

# MPI Library, includes for C++ bindings
mpi2cxx = os.path.join(env['WPPLIBDIR'], 'include', 'mpi2c++')

env.Append(WPPINCDIRS = os.path.join(os.sep, 'usr', 'include' , 'malloc'))
```

Index

- block options
 - see sub-region options, 35
 - vp, vs, r, ps, vpggrad, vsgrad, rhograd, 36
- command
 - block, 35
 - damping, 41
 - efile, 36
 - fileio, 38
 - grid, 34
 - image, 39, 40
 - projection, 41
 - restart, 41
 - sac, 39
 - source, 37
 - time, 35
 - vfile, 36
- command line options
 - v version info, 4
- coordinate system, 3
- efile options
 - model, logfile, vsmin, vpmmin, water, query, resolution, etree, xetree, 37
 - see sub-region options, 35
- fileio options
 - name, path, verbose, printcycle, 38
- geographic coordinates, 10
- grid options
 - location - az, lat, lon, 34
 - size - x, y, z, h, nx, ny, nz, 34
- grid size, 18
- image options
 - file, mode, 39
 - location - x, y, z, i, j, k, 39
 - timing - time, timeInterval, cycle, cycleInterval, 39
- image3d options
 - file, mode, topdirmod, logifymin, logifymax, compressionFactor, verbose, sample, 41
 - timing - time, timeInterval, cycle, cycleInterval, 41
- parallel execution, 3
- restart options
 - file, dumpInterval, fromCycle, 41
- sac options
 - location - x, y, z, i, j, k, lat, lon, depth, 39
 - sta, sample, file, type, writeEvery, eventDate, eventTime, 39
- source options
 - Aki and Richards - strike, dip, rake, 38
 - location - x, y, z, i, j, k, depth, lat, lon, 38
 - moment - m0, Mxx, Myy, Mzz, Mxy, Mxz, Myz, 38
 - point force - f0, Fx, Fy, Fz, 38
 - t0, freq, type, 38
- source time dependence
 - GaussianInt, Gaussian, RickerInt, Ricker, Ramp, Triangle, Sawtooth, Smoothwave, Brune, BruneSmoothed, 38
- srun, 3
- sub-region options
 - x1, x2, x, y1, y2, y, z1, z2, z, i1, i2, i, j1, j2, j, k1, k2, k, 35
- time options
 - t, steps, 35
- units, 3
- vfile options
 - see sub-region options, 35
 - type, file, vp, vs, r, ps, scaling, float, 36

Bibliography

- [1] K. Aki and P.G. Richards. *Quantitative Seismology*. University Science Books, second edition, 2002.
- [2] 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.
- [3] B. Gustafsson, H.-O. Kreiss, and J. Oliger. *Time dependent problems and difference methods*. Wiley–Interscience, 1995.
- [4] H. Lamb. On the propagation of tremors over the surface of an elastic solid. *Phil. Trans. Roy. Soc. London, Ser. A*, 1904.
- [5] Harold M. Mooney. Some numerical solutions for Lamb’s problem. *Bulletin of the Seismological Society of America*, 64, 1974.
- [6] S. Nilsson, N.A. Petersson, B. Sjögreen, and H.-O. Kreiss. Stable difference approximations for the elastic wave equation in second order formulation. Technical Report UCRL-JRNL-222276, Lawrence Livermore National Laboratory, 2006. Accepted for publication in SIAM J. Numer. Anal.