AnisWave2D: USER'S GUIDE TO THE 2D ANISOTROPIC FINITE-DIFFERENCE CODE

Aoife Toomey

Earth Sciences Division, Lawrence Berkeley National Laboratory,

University of California, Berkeley, CA 94720

June 2004

This work was supported by the Director, Office of Science, Office of Basic Energy Sciences, Division of Chemical Sciences, Geosciences and Biosciences of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098

Abstract

This document describes a parallel finite-difference code for modeling wave propagation in 2D, fully anisotropic materials. The code utilizes a mesh refinement scheme to improve computational efficiency. Mesh refinement allows the grid spacing to be tailored to the velocity model, so that fine grid spacing can be used in low velocity zones where the seismic wavelength is short, and coarse grid spacing can be used in zones with higher material velocities. Over-sampling of the seismic wavefield in high velocity zones is therefore avoided. The code has been implemented to run in parallel over multiple processors and allows large-scale models and models with large velocity contrasts to be simulated with ease.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Finite-difference modeling is now routinely carried out to aid interpretation of seismic data acquired during hydrocarbon exploration. Generating accurate synthetic seismograms for wave propagation in materials with large velocity contrasts is difficult using conventional fixed-grid finite-difference schemes running on serial processors. Low velocity modes require fine sampling in order to avoid numerical dispersion. In fixed-grid finite-difference schemes this fine grid spacing must be used throughout the model, resulting in over-sampling of the seismic wavefield in materials with higher velocities.

Large velocity contrasts are common in many of the problems relevant to the hydrocarbon industry. Most hydrocarbon reservoirs are overlain by low velocity sediments, whose effects on the wavefield must be considered in order to successfully image deeper structures. Borehole seismic surveys involve large velocity contrasts between the fluid-filled borehole, the steel casing and the region of interest outside the borehole. The small dimension of the borehole, which requires fine grid spacing for accurate representation, in contrast with the large-scale region of interest outside the borehole, add another layer of complexity to borehole modeling. Including the effects of anisotropy on the seismic wavefield further increases the computational expense due to the larger number of material properties that need to be stored. Serial-processor, fixed-grid finite-difference modeling has thus been restricted to small-scale models, unrealistic velocity structures, and/or long wavelengths due to its demand on computer time and memory.

The AnisWave2D code overcomes these limitations by using parallel computation combined with a mesh refinement scheme, enabling larger and more realistic geological materials to be tackled.

We use the finite-difference time domain staggered grid scheme described by Levander (1988), with a mesh refinement algorithm based on that developed by Pitarka (1999). The code is written in the C programming language and has been implemented to run in parallel across multiple processors using MPI (message passing interface) subroutines to handle communication between processors.

2. THE FINITE-DIFFERENCE METHOD

Wave propagation in a 2D linear elastic medium is described by two sets of equations – the equations of motion and the stress-strain relations (Hooke's law).

$$\rho \frac{\partial^2 u_x}{\partial t^2} = \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xz}}{\partial z} + f_x \qquad \textbf{1}$$

$$\rho \frac{\partial^2 u_z}{\partial t^2} = \frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{zz}}{\partial z} + f_z$$

We use the stress-strain relations for a fully anisotropic material:

$$\tau_{xx} = c_{11} \frac{\partial u_x}{\partial x} + c_{13} \frac{\partial u_z}{\partial z} + c_{15} \left(\frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x} \right)$$

$$\tau_{zz} = c_{13} \frac{\partial u_x}{\partial x} + c_{33} \frac{\partial u_z}{\partial z} + c_{35} \left(\frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x} \right) \qquad \textbf{2}$$

$$\tau_{xz} = c_{15} \frac{\partial u_x}{\partial x} + c_{35} \frac{\partial u_z}{\partial z} + c_{55} \left(\frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x} \right)$$

The u's are the components of displacement, the τ's are the components of stress, ρ is the density and the c's are the six independent elastic constants of a 2D fully anisotropic medium. The f's are the body force components.

Rewriting 1:

$$\frac{\partial v_x}{\partial t} = \frac{1}{\rho} \left(\frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xz}}{\partial z} + f_x \right) \qquad \textbf{3}$$

$$\frac{\partial v_z}{\partial t} = \frac{1}{\rho} \left(\frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{zz}}{\partial z} + f_z \right)$$

where the v's are the velocity components. Differentiating the stress-strain relations with respect to time gives a set of first-order differential equations:

$$\frac{\partial \tau_{xx}}{\partial t} = c_{11} \frac{\partial v_x}{\partial x} + c_{13} \frac{\partial v_z}{\partial z} + c_{15} \left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x} \right)$$

$$\frac{\partial \tau_{zz}}{\partial t} = c_{13} \frac{\partial v_x}{\partial x} + c_{33} \frac{\partial v_z}{\partial z} + c_{35} \left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x} \right) \qquad \textbf{4}$$

$$\frac{\partial \tau_{xz}}{\partial t} = c_{15} \frac{\partial v_x}{\partial x} + c_{35} \frac{\partial v_z}{\partial z} + c_{55} \left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x} \right)$$

Equations 3 and 4 form a system of first-order differential equations that can be solved numerically using finite-differences to approximate the derivatives. We use the staggered grid formulation described in Levander (1988) and Graves (1996). This scheme is 4^{th}-order in space and 2^{nd}-order in time**.** The wavefield variables and material parameters are discretized on a staggered grid mesh as shown in figure 1. Variables located on the node are called node-centered variables, while those located between nodes are called face-centered. The advantage of the staggered grid approach is that in an isotropic material the spatial derivatives of velocity and stress are always centered in space around the variable being updated, which increases the stability of the finite-difference method. The leap frog scheme also avoids taking derivatives of the material properties. For anisotropic materials, some of the spatial derivatives of stress and velocity must be interpolated in order to center them about the variable being updated. Staggered grid schemes can be used to model any variation in material properties with minimal numerical dispersion and numerical anisotropy (Levander, 1988).

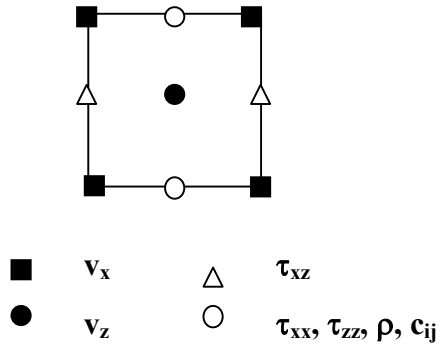


Figure 1. Schematic representation of the staggered grid. v_x is node-centered in both the x- and z-directions, v_z is face-centered in the x- and z- directions, t_{xz} is node-centered in the x-direction and face-centered in the z-direction, and so on.

The difference equations are given by:

$$D_t v_x(i,j) = \frac{1}{\rho} \left(D_x \tau_{xx}(i,j) + D_z \tau_{xz}(i,j) \right)$$

$$D_t v_z(i+1/2,j+1/2) = \frac{1}{\rho} (D_x \tau_{xz}(i+1/2,j+1/2) + D_z \tau_{zz}(i+1/2,j+1/2)) \quad \textbf{5}$$

for updating the velocities, and:

$$D_t \tau_{xx}(i+1/2,j) = c_{11} D_x v_x(i+1/2,j) + c_{13} D_z v_z(i+1/2,j) + c_{15} (D_z v_x(i+1/2,j) + D_x v_z(i+1/2,j))$$

$$D_t \tau_{zz}(i+1/2,j) = c_{13} D_x v_x(i+1/2,j) + c_{33} D_z v_z(i+1/2,j) + c_{35}(D_z v_x(i+1/2,j) + D_x v_z(i+1/2,j)) \quad \textbf{6}$$

$$D_t \tau_{xz}(i,j+1/2) = c_{15} D_x v_x(i,j+1/2) + c_{35} D_z v_z(i,j+1/2) + c_{55}(D_z v_x(i,j+1/2) + D_x v_z(i,j+1/2)$$

for updating the stresses.

D_t is the second-order forward-difference approximation for the time derivatives, i.e.,

$$D_t v_z = \frac{\partial v_z}{\partial t}, \text{ which is approximated by } \frac{v^{(t+1/2\Delta t)} - v^{(t-1/2\Delta t)}}{\Delta t} \quad \textbf{7}$$

where Δt is the timestep. D_x and D_z are the fourth-order forward- or reverse-difference operators used for the space derivatives. Either the forward- or the reverse- operator is chosen so that the difference operator is centered about the quantity being updated. For instance, to update τ_{xz} we use the reverse difference operator for $\frac{\partial v_z}{\partial x}$:

$$D_x v_z(i,j+1/2) = \frac{\partial v_z}{\partial x}, \text{ which is approximated by}$$

$$\frac{1}{\Delta x} \{ c_0 [v_z(i+1/2,j+1/2) - v_z(i-1/2,j+1/2)] - c_1 [v_z(i+3/2,j+1/2) - v_z(i-3/2,j+1/2)] \} \quad \textbf{8}$$

where $c_0 = 9/8$ and $c_1 = 1/24$, and Δx is the grid spacing in the x-direction.

3. MESH REFINEMENT

In uniform-grid finite-difference schemes the grid must be sampled finely enough to avoid dispersion at all points in the model. Since the seismic wavelength is shortest in low velocity zones, this means that the grid

spacing everywhere in the model must be fine enough to avoid dispersion in the low velocity areas. As a result, significant over-sampling of the wave field can occur in zones with higher velocities. The use of variable grid spacing allows us to partially avoid this over-sampling by assembling grids with different spacing to cover different regions of the model. This can result in significant reductions in computational memory requirements. We have implemented a mesh refinement scheme in the parallel code using the approach developed by Pitarka (1999). This approach avoids the need to interpolate the wavefield between regions with different spacing. Instead, the fourth-order finite-difference operators are designed for the nonuniform grid spacing. A brief description of the method is given here; the reader is referred to Pitarka (1999) for further detail.

In Pitarka's scheme, each dimension of the grid spacing can vary in one direction (i.e. x-spacing can vary in x-direction, z-spacing can vary in z-direction), as shown in Figure 2.

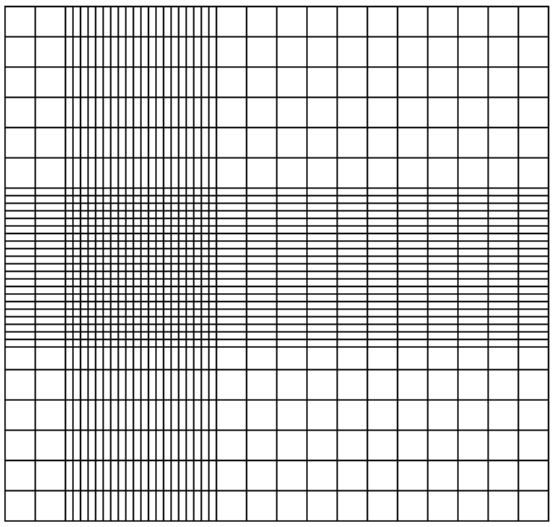


Figure 2. Example grid layout with spatially varying grid spacing.

Referring back to equation 8 above, in a uniform grid, the fourth-order finite-difference operator with respect to, for example, x, acting on a field variable g, is given by:

$$\frac{\partial g}{\partial x} = \frac{1}{\Delta x} \{ c_0 [g_{(i+1/2,j,k)} - g_{(i-1/2,j,k)}] - c_1 [g_{(i+3/2,j,k)} - g_{(i-3/2,j,k)}] \} \qquad \mathbf{9}$$

where $c_0 = 9/8$ and $c_1 = 1/24$. On a variable grid, Δx is no longer constant, but can vary in the x-direction. The finite-difference operator now depends on the spatial increments, Δ_i, which can be expressed in terms of the variable grid spacing, as shown schematically in figures 3 and 4. Because we are using a staggered grid, the finite-difference operators are face-centered or node-centered, depending on the location of the field variable being differentiated with respect to the field variable being updated. For example, to update τ_{xx} we need to calculate $\partial v_x/\partial x$ and $\partial v_z/\partial z$ at the location of τ_{xx}. τ_{xx} is face-centered in the x-direction, and node-centered in the z-direction, as shown in Figure 1. Hence, to calculate $\partial v_x/\partial x$ we use the face-centered spatial increments shown in Figure 3, and to calculate $\partial v_z/\partial z$ we use the node-centered spatial increments shown in Figure 4

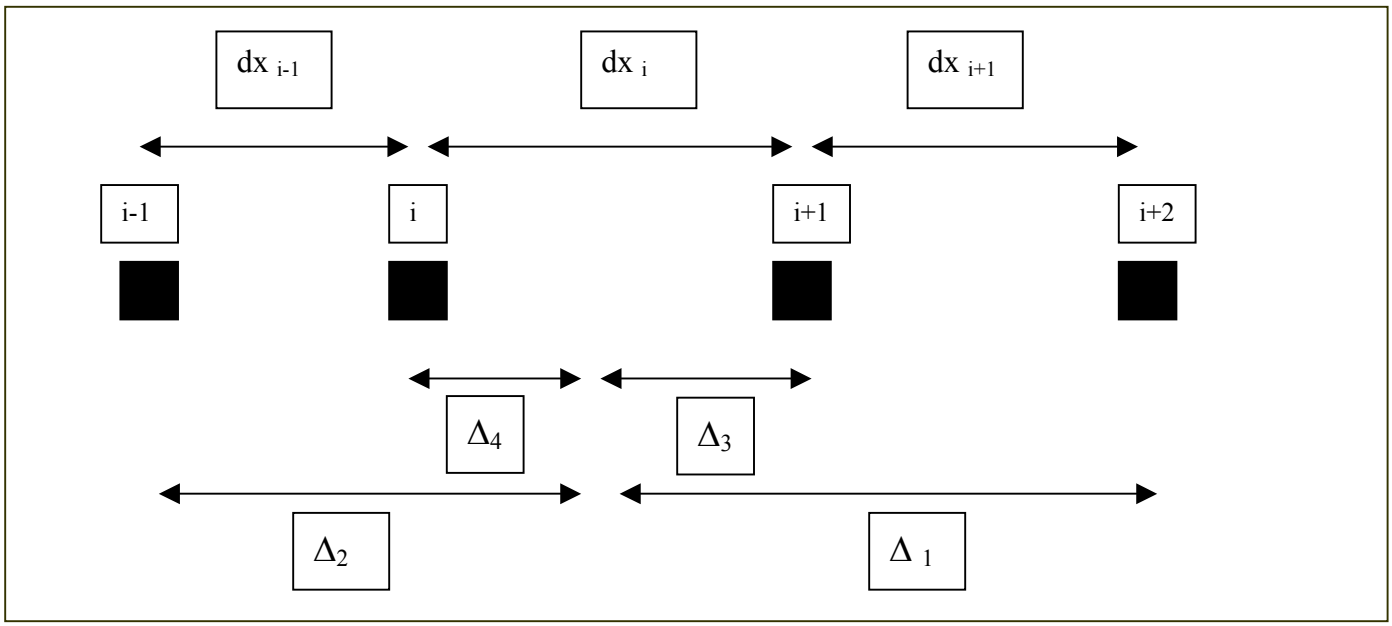


Figure 3. Schematic drawing showing how the Δ_i's are calculated from the grid spacing, for face-centered finite-difference operators. These are used to calculate the coefficients for the finite-difference operator.

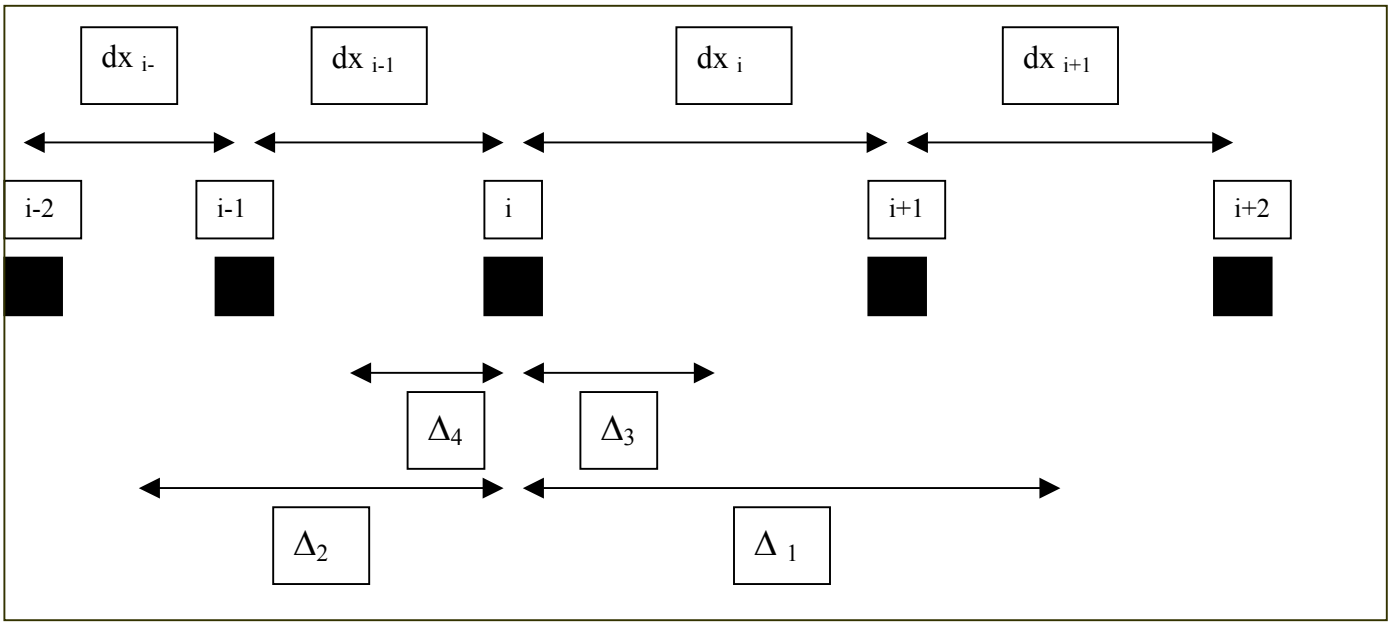


Figure 4. Schematic drawing showing how the Δ_i's are calculated for node-centered finite-difference operators.

The variable-grid finite-difference operator is given by:

$$\frac{\partial g}{\partial x} = c_1\, g(x+\Delta_1, y, z) + c_2\, g(x-\Delta_2, y, z) + c_3\, g(x+\Delta_3, y, z) + c_4\, g(x-\Delta_4, y, z) \qquad \mathbf{10}$$

where c_i are four coefficients to be determined. Because we are considering the spatial derivative of g with respect to x, we can assume $g(x,z)=g_z \exp(ikx)$ with g_z representing its dependence only on z. Substituting this in to equation 10, we get

$$ik = c_1 \exp(ik\Delta_1) + c_2 \exp(-ik\Delta_2) + c_3 \exp(ik\Delta_3) + c_4 \exp(-ik\Delta_4) \qquad \mathbf{11}$$

Taylor's expansion up to order $O(\Delta_i^4)$ is used to approximate the exponentials in equation 11:

$$\exp(ik\Delta_i) \approx (1 + ik\Delta_i - \tfrac{1}{2} k^2\Delta_i^2 - 1/6\, ik^3\Delta_i^3) \qquad \mathbf{12}$$

Substituting this in to equation 11:

$$
\begin{aligned}
ik &= (c_1+c_2+c_3+c_4) \\
&+ ik(c_1\Delta_1 - c_2\Delta_2 + c_3\Delta_3 - c_4\Delta_4) \\
&+ k^2/2(-c_1\Delta_1^2-c_2\Delta_2^2-c_3\Delta_3^2-c_4\Delta_4^2) \qquad \textbf{13} \\
&+ ik^3/6(-c_1\Delta_1^3+c_2\Delta_2^3-c_3\Delta_3^3+c_4\Delta_4^3)
\end{aligned}
$$

which can be replaced by a system of linear equations:

$$
\begin{pmatrix} 1 & 1 & 1 & 1 \\ \Delta_1 & -\Delta_2 & \Delta_3 & -\Delta_4 \\ -\Delta_1^2 & -\Delta_2^2 & -\Delta_3^2 & -\Delta_4^2 \\ -\Delta_1^3 & \Delta_2^3 & -\Delta_3^3 & \Delta_4^3 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \qquad \textbf{14}
$$

This system can be solved to find the coefficients of the finite-difference operator $\partial/\partial x$. Coefficients of the $\partial/\partial z$ operator can be similarly determined. Each finite-difference operator has two sets of four coefficients c_i that depend on the position of the field variables in the staggered grid. Since the coefficients of the finite-difference operator $\partial/\partial x$ vary only in the x direction, we need to store $2*4*N_x$ coefficients for $\partial/\partial x$, where N_x is the number of grid cells in the x-direction. The number of coefficients required for $\partial/\partial z$ is $2*4*N_z$. The total number of coefficients required is $8*(N_x+N_z)$. The amount of memory required to store these is much smaller than the memory saved by application of this scheme. These coefficients are determined in the program prior to the finite-difference calculation, after the user has chosen the grid spacing.

4. ANISOTROPY

The full tensor 6 independent elastic constants (21 for a 3D model) is used in this code, allowing fully general anisotropy to be simulated:

$$
Cij = \begin{pmatrix} c_{11} & c_{13} & c_{15} \\ c_{13} & c_{33} & c_{35} \\ c_{15} & c_{35} & c_{55} \end{pmatrix} \qquad \textbf{15}
$$

Isotropic, VTI and TTI materials can be modeled.

5. STABILITY AND NUMERICAL DISPERSION

The timestep and grid spacing must be chosen to avoid numerical dispersion and aliasing of the wave field. Levander (1988) carried out an analysis of the stability and dispersion properties of the second-order in time, fourth-order in space staggered grid finite-difference scheme and showed that 5 grid points per minimum seismic wavelength is sufficient to avoid numerical dispersion.

The stability condition for a 4^{th}-order in space finite-difference scheme is:

$$\Delta t \leq .606\ \Delta l / V_{max} \qquad \textbf{16}$$

(Graves, 1996) where V_{max} is the maximum phase velocity in the model and Δl is the grid spacing. Pitarka (1999) confirmed through numerical tests that these criteria hold for models with variable grid spacing, in which case the value of Δt is calculated for each grid point, using the minimum of Δx and Δz at that point. The minimum of all the Δt's is then chosen.

6. ABSORBING BOUNDARIES

Absorbing boundaries are implemented by ramping down the particle velocities and stresses in a strip surrounding the model (Cerjan et al, 1985):

$$v_x = v_x \exp(-(\gamma\ dx)^2 - (\gamma\ dz)^2) \qquad \textbf{17}$$

γ controls the strength of the absorbing boundaries. dx and dz are the horizontal and vertical distances, in number of grid points, from the inner edge of the absorbing boundary. The decay function thus increases with distance in to the absorbing boundary. The user chooses the strength and width of the absorbing boundaries. Typical values are γ=.017 and a width equivalent to three seismic wavelengths.

7. EXPANDING BOX METHOD

An expanding box algorithm is implemented in the code, significantly reducing computation times. The maximum velocity in the model is calculated and the wavefield is only updated within those grid points that have been reached by the fastest phase. This avoids the unnecessary update of the wavefield in grid points where the velocity and stress are unchanging. The user may notice that the time taken to complete one timestep increases as the simulation progresses, due to expansion of the domain within which the wavefield is updated.

8. ACCURACY OF THE CODE

To confirm the accuracy of our code, we compared numerical results for wave propagation in a homogeneous, isotropic 2D model with an analytical solution to the wave equation. We used a model with 460x460 grid points, a grid spacing of 2.75m and a timestep of 4 ms. The P wave velocity in the model was 2300 m/s, the S wave velocity was 1100 m/s and the density was 2100 kg/m3. Figure 5 shows the vertical velocity field sampled at 390m from the source (a 20Hz vertical body force). Excellent agreement is seen between the numerical and analytical results.

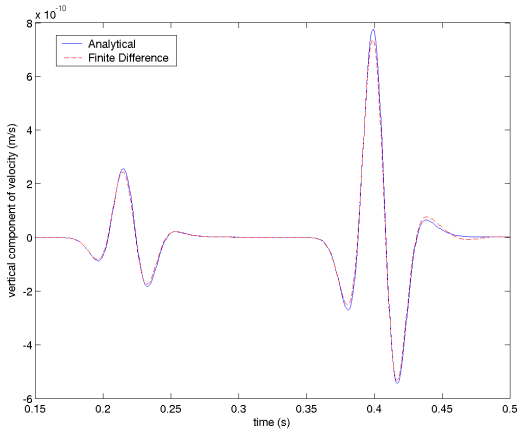


Figure 5. Comparison between numerical results and analytical solution for wave propagation in a homogeneous, isotropic grid.

9. PARALLELIZATION

The finite-difference method is ideally suited to parallelization as the problem domain can easily be decomposed among multiple processors. The AnisWave2D code is written in C/MPI and has been tested on

a Linux cluster and the National Energy Research Scientific Computing (NERSC) Center's IBM SP machine. The code is designed to be fully portable.

Decomposition of the model is handled within the code. The model is split into horizontal layers, each of which is sent to a different processor. The user should ensure that the model size and number of processors used are such that each processor receives at least 4 model rows.

In order to update the stress and velocity fields at a grid point, the stresses and velocities at 8 surrounding grid points must be known. If any of these grid points are located in another processor's subdomain, the values of stress and velocities at those points must be communicated between processors. The amount of communication carried out by each processor thus depends on the surface area of the subdomain to which it has been assigned. Communication is slow and is the reason why linear speed-up is not always achieved in parallel computing. The current implementation of the code employs one-dimensional decomposition of the grid – i.e. the model is divided into rows instead of squares. This type of decomposition requires more inter-processor communication than 2D decomposition, but offers ease of implementation. Future versions of the code may use 2D decomposition if increased communication time is found to be a significant hindrance.

Speedup for an example parallel finite-difference simulation is plotted in Figure 6 as a function of the number of processors used. The model used in this simulation had 5000x5000 grid points and was run for 100 timesteps, with no data output. The simulation was carried out on a Linux cluster consisting of 15 Linux PC's, each with two Pentium III processors, connected by Ethernet. Processor speeds ranged between 1GHz and 1.4 GHz. Each processor had 1 GB of RAM. Linear speedup is also plotted in Figure 6.

Speedup for the finite-difference code is seen to be superlinear when less than 15 processors are used. For more than 15 processors a slight degradation in performance is observed – this is due to the heterogeneous nature of the cluster used. For small numbers of processors, the faster processors are used in preference to slower processors. As the number of processors is increased, the slow processors are being used also. Because data needs to be transferred between processors at each timestep, the simulation can only proceed at the pace of the slow processors.

Number of processors	execution time (seconds)	% memory use
1	4971.468	66
2	1975.679	34
4	1076.906	32
8	574.887	16
10	469.541	13
15	436.746	9
20	301.685	7
25	252.105	6
29	219.61	6

Table 1. Execution times for a parallel finite-difference simulation using a 5000x5000 model, for 100 timesteps with no data output.

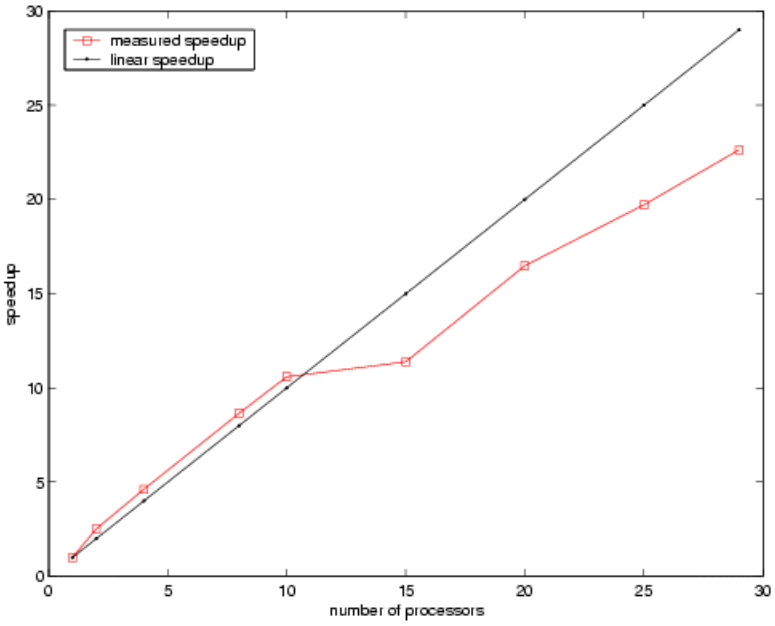


Figure 6. Observed speedup (serial execution time/parallel execution time) on the LBNL Center for Computational Seismology PC cluster, compared with linear speedup.

If a model is decomposed over too many nodes, each processor will receive only a small number of grid points and will spend most of its time communicating values at the subdomain boundaries rather than

updating the wave field within the subdomain, resulting in a serious degradation in performance. It is thus important to remember that using more processors does not necessarily increase speedup and can actually result in longer wall-clock times. This effect is not observed for our current simulations on only 29 processors.

Tests indicate that a 400 million grid point (approximately 20,000x20,000 grid points) model can be run on the LBNL cluster (29 processors, 1 GB RAM each). Since memory use scales linearly, these figures can be used to estimate the maximum possible model size on any machine.

10. LOAD BALANCING

Load-balancing allows faster processors to request extra data from the master during execution if they find themselves idle, and may improve performance of the code. A simple load-balancing algorithm has been implemented and tested in the AnisWave2D code. This algorithm allows a heterogeneous number of rows to be sent to each processor at the beginning of execution but does not allow the processor subdomains to vary dynamically during execution. Each process probes for its CPU speed at the beginning of the simulation and is then assigned a number of rows, weighted according to its CPU speed. A significant improvement in performance is seen when load balancing is used. However, using a load balancing algorithm decreases the maximum model size that can be run on the cluster since the additional data sent to the faster nodes increases their memory usage. The user can choose whether to use load balancing using an option in the input file. The portability of the load balancing algorithm has not been tested – the load balancing finds the CPU speed of each processor by checking the /proc/cpuinfo file, which may not be available on other systems.

11. SOURCES

The source, f(t), can be applied as a stress or a body force:

Body force source: $$\frac{\partial v_i}{\partial t} = \frac{1}{\rho}\left[\frac{\partial \sigma_{ij}}{\partial x_j} + f_i\right] \quad \textbf{18}$$

Stress source: $$\sigma_{ij} = f_i \quad \textbf{19}$$

This method of applying a stress source results in a source that is not transparent, hence we use

$$\text{Stress source:} \qquad \frac{\partial \sigma_{ij}}{\partial t} = \underline{c_{ijkl}} \left(\frac{\partial \underline{u_k}}{\partial x_l} + \frac{\partial \underline{u_l}}{\partial x_k} \right) + f_i \qquad \textbf{20}$$

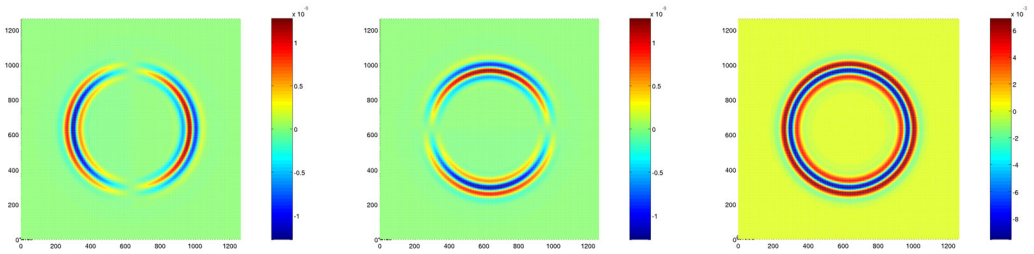
which results in a transparent source.

The source should be generated and appended to the input parameters file. Monopole, dipole and quadrupole sources can be specified, as well as plane wave sources. The user specifies the number of source points to be used, and the number of different source functions. Combining different source functions at different locations in the model can generate any type of source.

For a monopole source, the source lines in the input file has the following type of format:

```
#No. sources; no. source wavelets; source type: v=velocity, t=traction; format %d,%d,%c\n
1,1,t
#Source x,z coordinates; x,z amplitudes: 1. or 0.; wavelet number; format (%e,%e,%e,%e,%d\n [1 to No.
sources])
632.5,632.5,1.0,1.0,1
```

i.e. a single source applied as a stress at the location (632.5,632.5), with equal amplitudes in the x and z directions. Figure 7 shows the wavefield produced by a monopole source in a homogeneous model.



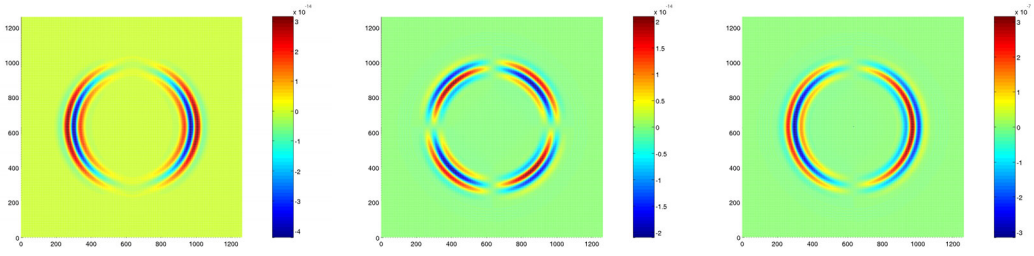
(a) (b) (c)

Figure 7. Velocity and pressure fields generated by a monopole source in a homogeneous model. (a) Horizontal velocity, (b) vertical velocity, (c) pressure.

Two different types of dipole source are commonly used. For a source in water, two monopole sources side by side, with opposite polarity, act as a dipole:

```
#No. sources; no. source wavelets; source type: v=velocity, t=traction; format %d,%d,%c\n
2,1,t
#Source x,z coordinates; x,z amplitudes: 1. or 0.; wavelet number; format (%e,%e,%e,%e,%d\n [1 to No.
sources])
632.5,632.5,1.0,1.0,1
635.25,632.5,-1.0,-1.0,1
```

This is for a model with a grid spacing of 2.75m. This source produces P wave radiation only, whether used in a solid or liquid medium (Figure 8).



(a) (b) (c)

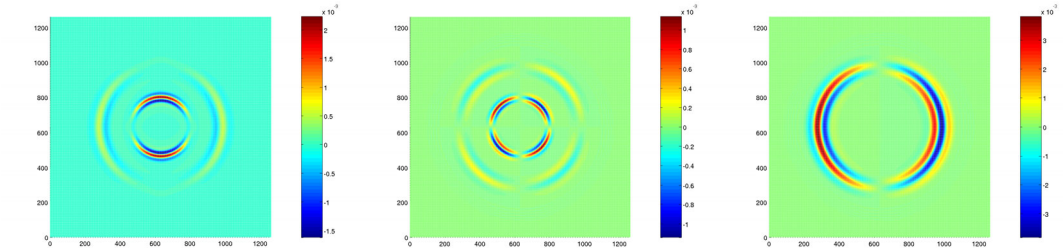
Figure 8. Velocity and pressure fields generated by a dipole source in a homogeneous model. (a) Horizontal velocity, (b) vertical velocity, (c) pressure.

If S wave radiation is desired, the following source format can be used (a directional point body force):

```
#No. sources; no. source wavelets; source type: v=velocity, t=traction; format %d,%d,%c\n
```

```
1,1,v
#Source x,z coordinates; x,z amplitudes: 1. or 0.; wavelet number; format (%e,%e,%e,%e,%d\n [1 to No.
sources])
632.5,632.5,1.0,1.0,1
```

The radiation pattern from this source is shown in Figure 9.



(a) (b) (c)

Figure 9. Velocity and pressure fields generated by a dipole source in a homogeneous model. (a) Horizontal velocity, (b) vertical velocity, (c) pressure.

To generate a quadrupole source, the source lines could have the following format:

```
#No. sources; no. source wavelets; source type: v=velocity, t=traction; format %d,%d,%c\n
4,1,t
#Source x,z coordinates; x,z amplitudes: 1. or 0.; wavelet number; format (%e,%e,%e,%e,%d\n [1 to No.
sources])
632.50,635.25,1.0,1.0,1
638.00,635.25,1.0,1.0,1
635.25,632.50,-1.0,-1.0,1
635.25,638.00,-1.0,-1.0,1
```

This specifies four source points arranged in a diamond and one source function. A monopole source is used at each point, with opposite polarities at adjacent points on the diamond. Figure 10 shows the arrangement of the source points and the polarity of the source at each point.

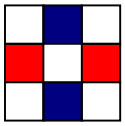
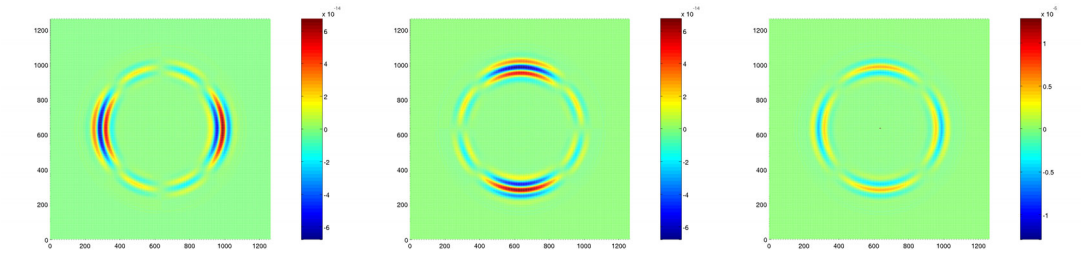


Figure 10. Arrangement of the source points for a quadrupole source. Red indicates positive polarity of the pressure source, blue indicates negative polarity.

Figure 11 shows the wavefield generated by this source.



(a) (b) (c)

Figure 11. Velocity and pressure fields generated by a quadrupole source in a homogeneous model. (a) Horizontal velocity, (b) vertical velocity, (c) pressure.

12. FILE ARCHIVE

The file archive AnisWave2D_beta5.0.tar contains program files and example input files for a finite-difference wave simulation. To extract files from the archives type 'tar –xvf AnisWave2D_beta5.0.tar' (unix/linux systems). The files have been compressed using gzip.

The following files are enclosed:

Input files for the Example Simulation:

uni.in	input file containing the problem parameters for a uniform grid simulation
var.in	input file containing the problem parameters for a variable grid simulation
uni3.mod	type 3 model file, uniform grid
var3.mod	type 3 model file, variable grid
uni.grid	grid spacing file, uniform grid
var.grid	grid spacing file, variable grid

Other Input Files:

test2.mod	example of a type 2 model file

Program files:

AnisWave2D_beta5.0.c	...	main program
AnisWave2D_asnaps.c	...	outputs ASCII snapshots
AnisWave2D_bsnaps.c	...	outputs binary snapshots
AnisWave2D_decompose.c	…	decomposes the model into separate domains for each processor
AnisWave2D_exchange1DD.c	...	exchanges material properties in the ghost rows between different processors
AnisWave2D_exchange1DI.c...		exchanges material properties in the ghost rows between different processors
AnisWave2D_exchange2DD2.c	...	exchanges velocities in the ghost rows between different processors
AnisWave2D_exchange2DD3.c	...	exchanges stresses in the ghost rows between different processors
AnisWave2D_f_abs.c	...	calculates the decay function in the absorbing boundaries
AnisWave2D_fd_forward.c		forward calculations
AnisWave2D_find_gridpoints.c		Finds the grid point locations of the sources and receivers
AnisWave2D_read_modeltype2.c		Reads in type 2 models
AnisWave2D_read_modeltype3.c		Reads in type 3 models
AnisWave2D_restart_in.c		Reads in a restart file
AnisWave2D_rotate.c		Rotates the TI stiffness matrix to find its components in the g rid coordinate system
AnisWave2D_segywrite.c		outputs SEG-Y traces
AnisWave2D_tupdate.c		updates the stress field
AnisWave2D_variable_gridding.c		reads in the grid spacing and solves for the finite-difference coefficients of the variable grid
AnisWave2D_vupdate.c		updates the velocity field
AnisWave2D_waveforms.c		outputs ASCII traces
AnisWave2D_header.h	...	contains header information for the main program
AnisWave2D_f2c.h	…	header file
makefile	…	contains compilation script for the code

Additional files:

cijfrac2D.f	…	computes elastic moduli for a fracture, for a given fracture orientation and stiffness
cijfrac2D_example.inp	…	example input file for cijfrac2D.f
cijfrac2D_example.out	…	example output file from cijfrac2D.f, used to generate beta.mod

Output files generated by the example simulation are also included in the file archive. The example simulation is described in section 20.

13. INPUT PARAMETER FILE

The input file contains parameters for the simulation. Comment lines in the input file (lines beginning with #) explain each input parameter and give the format for that parameter. All model coordinates given in the input file must include the absorbing boundaries (for example, the number of grid points is the *total* number of grid points, not the number inside the absorbing boundaries). Note also that coordinates are always given in meters.

Since the input parameter file is self-explanatory, only a brief explanation is given here. Model types, source formats, etc. are explained in greater detail in the sections 'Model Building' and 'Sources' respectively. The format description in every comment line describes the C language format of the variable, i.e. %d refers to a single integer, %e refers to a double. Note that every comment line must always be included in the input file. However, if an option has not been chosen, for example, if no snapshot output has been chosen, the lines containing parameters for snapshot output are omitted. Optional lines are printed in blue below. The format of parameters in the input file should be adhered to.

The order of anisotropy in the model should be specified in the input file. For example, if an isotropic or VTI model is specified, only the elastic constants needed for a VTI material (4 elastic constants) will be stored during the simulation. If a fully anisotropic material is specified, six elastic constants will be stored in memory during the simulation – increasing the computational expense. The input file also gives the option

of choosing an elastic or acoustic run – the acoustic option is not implemented yet and will be available in future versions of the code.

```
#Model: beta test
#Input Model Type: 2=Thomsen cell-based model, 3=stiffness domain-based model; format %d\n
3
#Order of anisotropy: 0=Isotropic or VTI,1=Fully anisotropic; format %d\n
0
#Elastic or Acoustic run: 0=Elastic, 1=Acoustic; format %d\n
0
#Number of Shots; format %d\n
5
#Shot and receiver increments, if more than 1 shot; format %e %e %e %e\n
50.,50.,50.,50.
#Mesh upper-left corner coordinates; format %e %e\n
100.0000 -100.000000
#Free surface: 1=free surface boundary condition used, 0 otherwise; format %d\n
0
#No. x, z  grid points; x, z bottom and top abs. boundaries; first timestep; no. timesteps; timestep; format
%d,%d,%d,%d,%d,%d,%d,%e\n
700,450,30,30,30,1,2100,.0006
#Strength of the absorbing boundary decay function; format %e\n
.015
#Load balancing: 1=use load balancing, 0=otherwise; format %d\n
0
#Restart file: 1=write restart file, 0=otherwise; format %d\n
1
#Timestep interval for writing restart file; format %d\n
500
#Data output: 1=snapshots, 2=seismic traces, 3=both; format %d\n
3
#Length of output file suffix, suffix string; format %d,%s\n
7,example
#No. snapshots, timesteps for snapshot output; format %d,(%d [1 to No. snapshots])\n
1,900
#No. receivers and receiver list for trace output; format %d\n (%e,%e\n [1 to No. receivers])
10,
400.,448.
480.,448.
560.,448.
640.,448.
720.,448.
800.,448.
880.,448.
960.,448.
1040.,448.
1120.,448.
#Sampling interval (in seconds) for trace output; format %e\n
.0006
#Data format for trace output: 0=ascii, 1=SEG-Y, 2=both; format %d\n
0
#No. sources; no. source wavelets; source type: v=velocity, t=traction; format %d,%d,%c\n
1,1,t
#Source x,z coordinates; x,z amplitudes: 1. or 0.; wavelet number; format (%e,%e,%e,%e,%d\n [1 to No.
sources])
1400.,448.,1.0,1.0,1
#Frequency range of source (Hz); format %e,%e\n
0.,60.
#No. data points in source wavelet; format %e\n
25
#Time,source function(s); format (%e,(%e [1 to No. source wavelets])\n [1 to No. data points in source])
        0.0000000000       -0.0000239679
        0.0006000000       -0.0000310509
        0.0012000001       -0.0000401036
        0.0018000001       -0.0000516361
        0.0024000001       -0.0000662802
        0.0030000001       -0.0000848149
        0.0036000002       -0.0001081968
```

```
0.0042000002       -0.0001375969
0.0048000002       -0.0001744423
0.0054000003       -0.0002204662
0.0060000003       -0.0002777634
0.0066000003       -0.0003488574
0.0072000003       -0.0004367756
0.0078000004       -0.0005451319
0.0084000004       -0.0006782265
0.0090000004       -0.0008411514
0.0096000005       -0.0010399119
0.0102000005       -0.0012815533
0.0108000005       -0.0015743126
0.0114000005       -0.0019277717
0.0120000006       -0.0023530161
0.0126000006       -0.0028628238
0.0132000006       -0.0034718403
0.0138000007       -0.0041967732
0.0144000007       -0.0050565623
```

14. GRID SPACING FILE

The grid spacing to be used in the simulation must be specified in a grid spacing file. The grid spacing file has the following format:

```
Mesh delta X nodes:
5.0 2.5 2.5 2.5 2.5 2.5 2.5 2.5
2.5 2.5 5.0 5.0 5.0 5.0 5.0 5.0
5.0
Mesh delta Z nodes:
5.0 5.0 5.0 2.5 2.5 2.5 2.5 2.5
5.0 5.0
```

Dimensions for up to eight grid points are listed on each line. The above is an example for a model with 17 grid points in the x-direction and 10 grid points in the z-direction. This example model is illustrated in Figure 12, along with the type of velocity model that this grid spacing might be used for. The yellow area has lower velocity than the red area and hence needs to be sampled more finely.

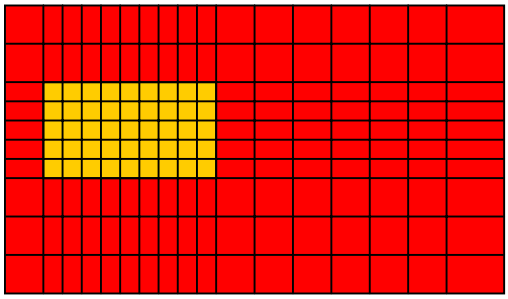


Figure 12. Example velocity model and grid spacing.

This figure illustrates how the variable grid spacing should be designed.

15. MODEL FILE

Two approaches are used for representing the properties of the model in this code.

Thomsen cell-based model (model type 2):

The first approach uses the Thomsen parameters of the material, which can be used to describe TI materials. The Thomsen parameters are:

$$\alpha = \sqrt{(c_{33}/\rho)} \qquad \textbf{21}$$

This is the P wave velocity along the TI symmetry axis.

$$\beta = \sqrt{(c_{44}/\rho)} \qquad \textbf{22}$$

This is the S wave velocity along the symmetry axis.

$$\varepsilon = \frac{(c_{11} - c_{33})}{2c_{33}} \qquad \textbf{23}$$

This is the difference between P wave velocity perpendicular to the symmetry axis and parallel to the symmetry axis, normalized by the velocity parallel to the symmetry axis.

$$\gamma = \frac{(c_{66} - c_{44})}{2c_{44}} \qquad \textbf{24}$$

This is the difference between the S wave velocity perpendicular to the symmetry axis, and velocity of S waves travelling parallel to the symmetry axis, with out-of-plane polarization. This parameter is not relevant in 2D modeling.

$$\delta = \{(c_{13} + c_{44})^2 - (c_{33} - c_{44})^2\} / 2c_{33}\ (c_{33} - c_{44}) \qquad \textbf{25}$$

If the TI axis is not vertical, the angle of dip, θ, relative to the grid coordinates must also be specified. θ is positive for rotation in the anti-clockwise direction.

In the AnisWave2D code, the parameters α, β, ρ, ε, γ, δ and θ are read in for every grid point and are used to calculate $c_{11,}$ c_{13}, c_{33}, c_{44}, and c_{66}. The C_{ij} matrix is then rotated through the angle θ to find the stiffness matrix in the grid coordinates.

The following is an example for a homogeneous, isotropic 5x5 grid-point type 2 model.

```
%Mesh Vp
2300.000000 2300.000000 2300.000000 2300.000000 2300.000000 2300.000000
2300.000000 2300.000000 2300.000000 2300.000000 2300.000000 2300.000000
2300.000000 2300.000000 2300.000000 2300.000000 2300.000000 2300.000000
2300.000000 2300.000000 2300.000000 2300.000000 2300.000000 2300.000000
2300.000000
%Mesh Vs
1100.000000 1100.000000 1100.000000 1100.000000 1100.000000 1100.000000
1100.000000 1100.000000 1100.000000 1100.000000 1100.000000 1100.000000
1100.000000 1100.000000 1100.000000 1100.000000 1100.000000 1100.000000
1100.000000 1100.000000 1100.000000 1100.000000 1100.000000 1100.000000
1100.000000
%Mesh Density
2100.000000 2100.000000 2100.000000 2100.000000 2100.000000 2100.000000
2100.000000 2100.000000 2100.000000 2100.000000 2100.000000 2100.000000
2100.000000 2100.000000 2100.000000 2100.000000 2100.000000 2100.000000
2100.000000 2100.000000 2100.000000 2100.000000 2100.000000 2100.000000
2100.000000
%Mesh Epsilon
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000
%Mesh Gamma
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000
%Mesh Delta
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000
%Mesh Rotation Angle
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000
```

Stiffness domain-based model (model type 3):

This model type is ideal for models with reasonably few different material types. The number of material types is given at the beginning of the model file, followed by the six elastic stiffnesses (c_{11}, c_{13}, c_{15}, c_{33}, c_{35}, c_{55},) and the density for each material type. A pointer to the material type is then read in for each grid-point. Consequently, only six stiffnesses are stored for each material type, in addition to an integer pointer for each grid-point. This is substantially more efficient than saving all six elastic stiffnesses for each grid-point.

Any order of anisotropy can be represented using this model type.

The following is an example for a homogeneous, isotropic 5x5 grid-point type 3 model.

```
test.mod: 1 Materials, 5x5 Model
1
material 1
1.110900e+10 6.027000e+09  0. 1.110900e+10 0. 2.541000e+09
2.100000e+03
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

The model file is read in by the master processor during parallel execution. The master reads in part of the model file, sends the data to the relevant slave processor, and then reads in the next segment of the model file. This ensures that the entire model is not stored in memory by any one processor. The maximum possible model size is thus governed by the total memory available on the parallel cluster.

Model types 1 and 4 are not yet implemented.

16. FRACTURES

Discrete fractures can be modelled in the anisotropic finite-difference code using the thin anisotropic grid cell approach of Coates & Schoenberg (1995), which uses equivalent medium theory to model material behaviour in the cells of the finite-difference grid intersected by the fracture. The compliance matrix of the equivalent medium is equal to the sum of the background compliance and the excess compliance due to the fracture. Inverting the compliance matrix gives the stiffness matrix.

If the background medium is isotropic, the stiffness matrix of a rotationally symmetric fault is transversely isotropic with its symmetry axis perpendicular to the fracture. The fracture can thus be represented either by calculating the Thomsen parameters of the TI medium, or by rotating the TI stiffness matrix to find its components in the grid coordinate system.

The code 'cijfrac2D.f', written by Kurt Nihei, can be used to calculate the stiffness matrix of the fracture in the grid coordinate system. This program requires an input file 'cijfrac2D*.inp' specifying the number of fracture grid cells, and the properties of the fracture in each of these grid cells, which include the orientation of the fracture in the grid cell (angle from horizontal, i.e. 90 for a vertical fracture), the grid size, the fracture stiffnesses (K_N and K_T), the shear stiffness of the background medium, the background Poisson's ratio, and the background density. An example input file is included in the AnisWave2D_beta5.0.tar file archive. The program outputs a file 'cijfrac2D*.out' containing the stiffness matrix for any grid cells containing this fracture. These values can then be included in a type 3 model file as the material parameters at the grid points containing the fracture.

Alternatively, the user may use the method of Coates and Schoenberg (1995) to calculate the stiffness matrix or Thomsen parameters for a fracture in an arbitrarily anisotropic background material.

17. COMPILATION AND EXECUTION OF THE CODE

Typing 'make' carries out compilation. This produces the executable 'AnisWave2D_beta5.0.exe'.

On a PC-cluster using LAM/MPI, a LAM session can be started by typing 'lamboot'. Begin execution by typing, for example, 'mpirun –np 15 ./AnisWave_beta5.0.exe parameter_file grid_file model_file', where

parameter_file, grid_file and model_file are the names of your input files. This begins execution on 15 processors.

18. OUTPUT FILES

The following output files are produced by the AnisWave2D code:

Snapshot files:

snp_timestep_shotnumber_suffix.dat

ASCII file with three components of velocity, and pressure, the grid point positions are also written.

File format is:

column 1:	column 2:	column 3:	column 4:	column 5:
x position	z position	vx	vz	pressure

Data is written with the x position varying most rapidly, followed by the z position.

Binary snapshot output can also be chosen. In this case, the entire pressure field is written out to a file in binary format.

Seismograms:

ASCII data is written in the following format to the file trace_shotnumber_suffix.out:

column 1:	column 2:	column 3:	column 4:	
receiver number (1)	vx	vz	pressure	(timestep 1)
.	.	.	.	(timestep 1)
.	.	.	.	(timestep 1)
receiver number (n)	vx	vz	pressure	(timestep 1)
receiver number (1)	vx	vz	pressure	(timestep 2)
.	.	.	.	…..

Each shot is written to a separate ASCII file.

SEG-Y data is outputted to separate files ('trace_p.sgy','trace_vx.sgy' etc.) in standard SEG-Y format. All shots are written to the same file. There is an option in the subroutine AnisWave2D_segywrite.c to choose between IBM or IEEE binary data format.

Output files for the example simulations described in section 19 are included in the file archive.

19. EXAMPLE SIMULATION

Included in the AnisWave2D_beta5.0.tar file archive are input files, grid spacing files, model files and output files for an example simulation. The velocity model used in this simulation (uni.mod and var.mod) consists of a high velocity layer overlying a lower velocity layer, with a vertical fracture in the bottom layer, as shown in Figure 13. The simulation was carried out twice - using uniform grid spacing of 4.m, and using spatially varying grid spacing (z grid spacing of 4.m in the top layer, 8.m in the bottom layer; uniform x grid spacing of 4.m over the entire model). The model had dimensions of 2.8x1.8 km. This corresponds to 315 thousand grid points for the uniform grid model, 237 thousand grid points for the variable grid model. The timestep was 6.ms and we ran the simulation for 2100 timesteps. A 20 Hz monopole source was used, located in the middle of the top layer.

These simulations were executed over 31 processors (Pentium III's, 14 with speeds of 1GHz, 1 with a speed of 1.26 GHz and 16 with speeds of 1.4 GHz, 1 GByte of RAM each). Execution took about 4 minutes. Both the uniform and variable grid models used at most 4% of RAM. Snapshot and trace files (ASCII) are included in the file archive. Data files from the uniform-grid simulation have the suffix 'uni' while those from the variable-grid simulation have the suffix 'var'.

Figure 14 shows a snapshot of the vertical velocity field after 900 timesteps. Figure 15 compares the vertical velocity field sampled at a point in the lower layer, for the uniform-grid and variable-grid simulations. There is excellent agreement between the results of the uniform- and variable-grid simulations, illustrating that a two-fold change in grid spacing can be handled by the variable-grid code without any reduction in accuracy.

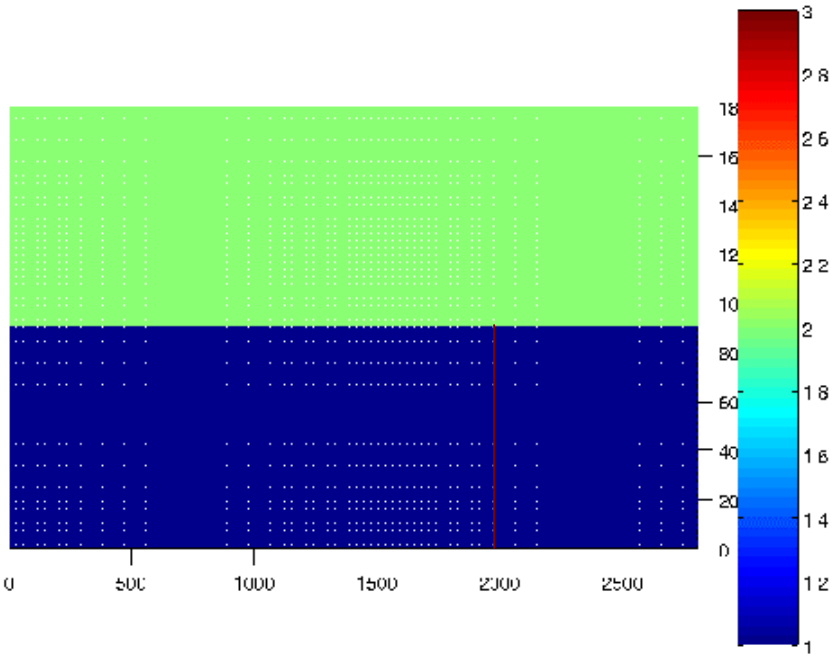


Figure 13. Velocity model used for the example simulation. Material type 1 has a P wave velocity of 2300 m/s, material type 2 has a P wave velocity of 3000 m/s, and material type 3 is a fracture.

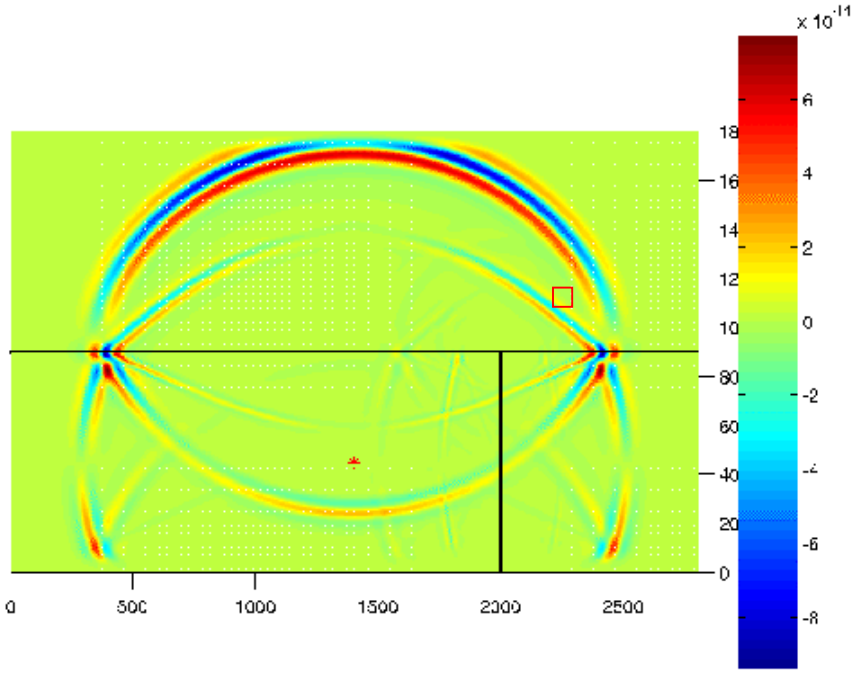


Figure 14. Snapshot of the vertical velocity field after 900 timesteps. Black lines show the location of the fracture and interface. The red star marks the source location. The red square marks the location where the trace shown in Figure 15 was sampled.

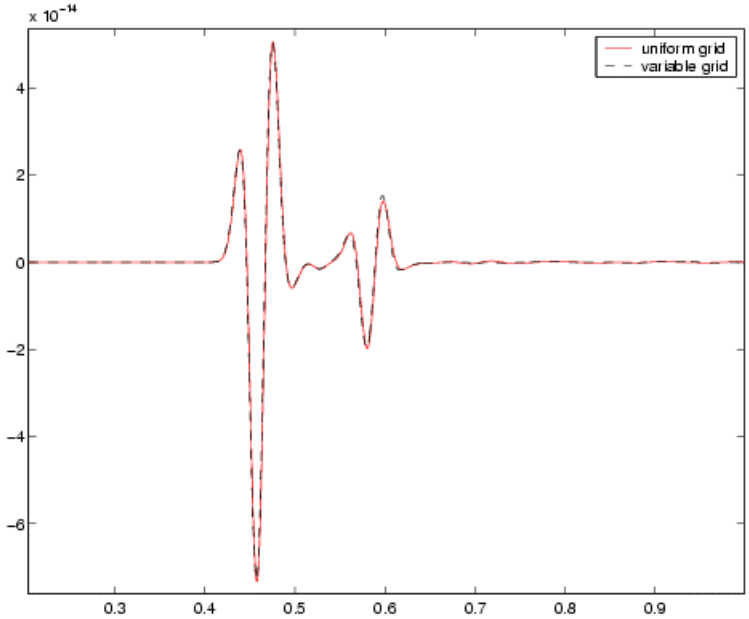


Figure 15. Vertical velocity field sampled at the location marked by a red square in Figure 13. Results for a uniform-grid simulation are compared to those for a variable-grid simulation.

20. CONCLUSIONS

The AnisWave2D code is a 4th-order in space, 2nd-order in time staggered-grid finite-difference code that includes variable grid capability for a parallel processor computer architecture. The code can be used to simulate wave propagation in 2D models with any order of anisotropy. Fractures can be simulated using the method of Coates and Schoenberg (1995). Tests have shown that results obtained using this code match those of an analytical solution to the wave equation. A mesh refinement algorithm implemented in the code allows the grid spacing to vary according to the velocity model, resulting in significant reductions in computational memory use and execution time.

21. REFERENCES

Cerjan et al. (1995). (Geophy., 50, 705-708, 1985)

Coates, R. T. and Schoenberg, M., Finite-difference modeling of faults and fractures, Geophysics, 60, 5, 1514-1526, 1995.

Graves, R.W. (1996). Simulating seismic wave propagation in 2D elastic media using staggered grid finite-differences, Bulletin of the Seismological Society of America, 86 (4), 1091-1106.

Levander, A. R., Fourth-order finite-difference P-SV seismograms. Geophysics, 53, 1425-1436, 1988.

Pitarka, A., 2D Elastic Finite-Difference Modeling of Seismic Motion Using Staggered Grids with Nonuniform Spacing. Bull. Seism. Soc. Am. 89, 1, 54-68, 1999.