Applied Numerical Mathematics 4 (1988) 97–139
North-Holland

NUMERICAL METHODS FOR INVERSE PROBLEMS IN THREE-DIMENSIONAL GEOPHYSICAL MODELING *

V. PEREYRA
Weidlinger Associates, 620 Hansen Way, Suite 100, Palo Alto, CA 94304, U.S.A.

We present an overview of work concerning various numerical aspects of some inverse problems in geophysics. Three-dimensional ray tracing in general piecewise smooth inhomogeneous media is used to forward and inverse model the elastic properties of the subsoil. The forward modeling provides an approximation to the propagation of elastic waves in the earth, while optimization methods are coupled with ray tracing in order to fit postulated models to reflection or refraction data.

Introduction

The purpose of this research has been to establish the feasibility of a number of techniques for modeling complex regions in three dimensions. To focus our attention we have concentrated on geological regions, but many of the techniques are applicable to a number of other classes of problems.

The main aspects we have considered are

(i) automatic three-dimensional two-point ray tracing in various types of media,

(ii) time-to-depth migration and material properties determination from travel time and peak amplitude data,

(iii) nonlinear least-squares inversion of travel time data.

For these problems we have discussed several algorithms and written working prototype implementations. These codes have been used to illustrate the techniques in some test examples.

The work done has shown the feasibility of the general approach, which emphasizes two-point ray-tracing and nonlinear optimization techniques. Details on the three main topics can be found in the corresponding sections. We summarize here some of the findings:

Three-dimensional ray tracing is an essential tool in this development. We feel that the techniques proposed in Section 1 and implemented in the program module AUTO3DRT (based on RAYT3D and PASVA4) provide a firm basis for further development. The combination of shooting as a search procedure and initiator, with the faster two-point algorithm, is novel and has proven most successful. The resulting code is flexible, reliable, accurate, robust and easy to use. It subsumes our accumulated experience in the area, and among other things solves nicely the problem of obtaining automatically all the arrivals of a given type that can be produced by irregularities in the velocity or in the interfaces. Such generality is quite essential if we want to concentrate our attention in the other aspects of the problem, without being distracted by idiosyncracies of the ray-tracing process.

* Research sponsored by the National Science Foundation under SBIR Grant ISI-8560154.

The subjacent two-point boundary value approach and supporting software have paid off by allowing to state and solve with relative ease and naturality a number of problems, namely

(a) calculation of geometrical spreading amplitudes,

(b) calculation of the sensitivity of travel time data, that can be used in an automatic 'receiver mesh placement' procedure in order to obtain a required resolution with minimum cost,

(c) time-to-depth migration.

For problems of moderate complexity in piecewise constant media with curved interfaces we show that a minicomputer is sufficient for performing most of these tasks. However, as the complexity increases, or if we consider variable velocities, the ray tracing becomes more expensive, and thus the iterative global inversion procedures of Section 3 require more powerful computing tools.

We have been lucky for the timely opening of the NSF supercomputer facilities, and in particular the one at the University of California, San Diego. All the modules developed in our office on a PRIME 2250 minicomputer have been ported successfully to a CRAY XMP/48, showing two orders of magnitude speedups on one processor, for code not optimized for vectorization. This has made our development possible within the given time span, and have brought the computer times to a level in which three-dimensional interactive modeling with these techniques seems feasible.

1. Automatic two-point ray tracing in three dimensions

1.1. Introduction

In [29] we have described an automatic two-point ray-tracing system for piecewise constant media with fairly general curved interfaces in two dimensions. In this section we extend those ideas to three dimensions.

We will first consider the piecewise constant media case, but without including pinch outs or other nonconformities. Within this framework we will discuss the control scheme necessary to start and restart automatically the two-point process, and to obtain multiple arrivals and ray signatures. This scheme will be complemented by an existing general code RAYT3D [28], which has options for shooting and two-pointing rays in three dimensions, even in completely inhomogenous media. We will show later on how to extend the piecewise constant case to this more general situation.

The basis of the control scheme will be a fast shooting algorithm for three-dimensional piecewise constant, curved layered media. The idea of a control screen with hierarchical refinements will be introduced. This will provide a mechanism for generating initial shooting directions, and for constructing an adequate control data structure which is intuitive and easy to manipulate.

Shot rays arriving sufficiently near a receiver will be used to start up a two-point solution for the ray joining the source and the receiver. Once initialized, the two-point solver takes over and continues from receiver to nearby receiver, until either the receivers are exhausted, or there is a failure in the algorithm (divergence, shadow zone, caustic,...). In either case, shooting is restarted until a new receiver is located, and so on. The control screen is used to keep track of directions already inspected, and also to avoid computing repeatedly the same source-receiver ray. When all directions in the (discrete) screen have been inspected, the algorithm terminates.

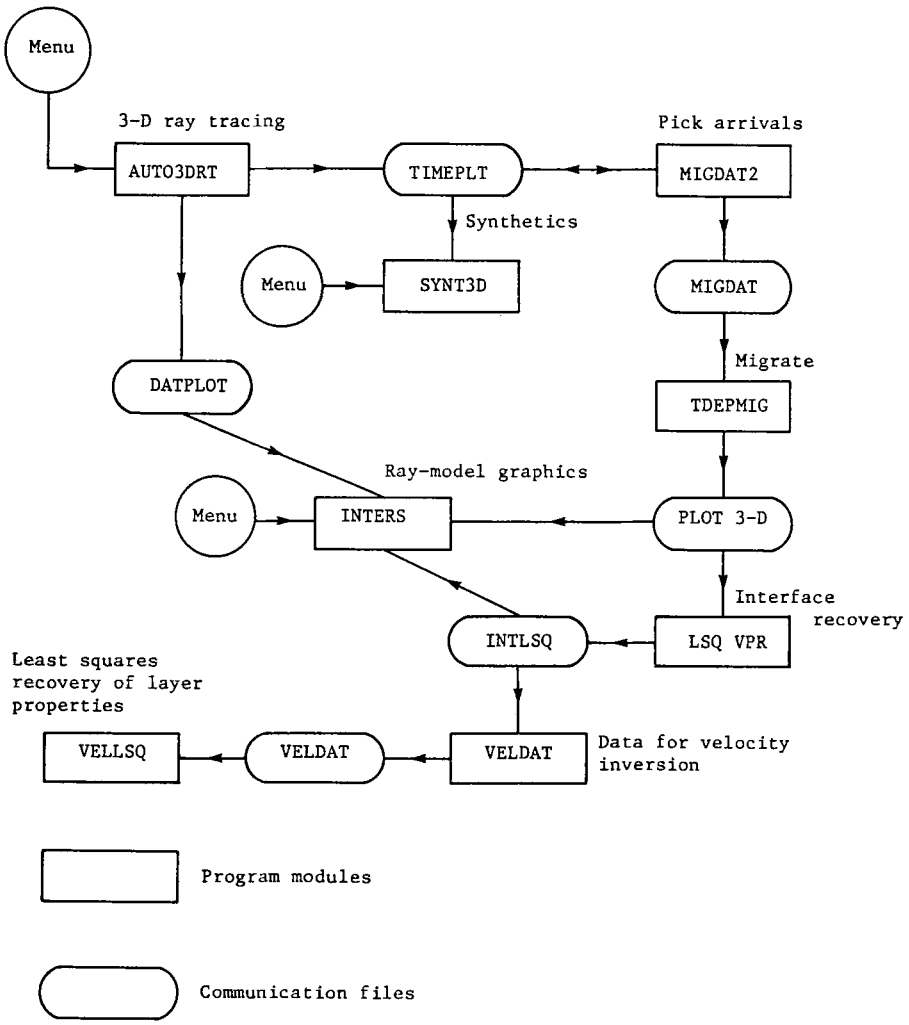

Fig. 1. INVPAK: 3-D inversion systems.

The code can be run purely in shooting mode in order to rapidly illuminate a complicated region, and also to obtain information on position and required resolution in subregions of interest. This is necessary since usually, for complicated regions, there will be several families of trajectories reaching the same receivers from a given source. Often, the initial directions for each individual family will be concentrated only on a thin beam emerging from the source. Thus, only a small part of the control screen may be needed in high resolution, and since the total work will be proportional to the number of pixels (directions) to be inspected, it is worthwhile to introduce a somewhat elaborated data structure at this point. The resulting automatic ray-tracing system will be called AUTO3DRT (see also Fig. 1).

1.2. Shooting rays in a three-dimensional piecewise constant media

We consider the problem of shooting rays from a given source $S_0 = (x_0, y_0, z_0)$ through a piecewise constant medium, with discontinuities occurring at smooth interfaces I_j, $j = 1, \ldots, \mathrm{NI}$.

These interfaces separate a given geological region B into connected components Re_j, $j = 1, \ldots, \mathrm{NR}$. In our current discussion B will be a parallelepiped.

Each region Re_j has associated a set of interfaces $[I_1^j, \ldots, I_{kj}^j]$, its faces, and we also assume that each interface I_i is the boundary for at most two regions $[\mathrm{Re}_a^i, \mathrm{Re}_b^i]$. The simplest case is that in which the regions are complete layers, and the interfaces are single-valued functions (say of (x, y)), and do not intersect over the whole window of interest. We refer to this as the 'layered model'.

Given a receiver $R = (x_r, y_r, z_r)$, there may be zero, one, or many different paths a ray can follows from the source to this receiver. Each one of these paths can be fully described by giving an ordered sequence of regions and interfaces traversed by the ray, or alternatively by a sequence of interfaces and of flags indicating transmission or reflection. We call either set of descriptors the 'ray signature'. The 'multiple arrivals' we referred to earlier, correspond to different paths joining source and receiver *with the same signature*. The two-point paths can also be uniquely determined by their initial direction and signature.

We will assume that the receivers lie on an interface I_{R} (real or artificial, i.e. without velocity contrast), and list I_{R} as the last interface in the signature of any ray. Because the regions have constant velocity, the ray paths within a region are straight lines. These segments (legs) break and change direction at interfaces according to Snell's law of geometrical optics. In a layered model, signatures can be abbreviated to contain only the list of interfaces at which the ray is reflected, since it is clear what are the interfaces between reflectors at which the ray will be transmitted. Again I_{R}, the receivers' interface, will have to be included as the last one in the list, in order to indicate when the ray terminates.

Another nice feature of layered models is that families of similar rays, say those arriving from a common source to an array of receivers, do not change signature. This is not true for more general models (even in two dimensions), and therefore when using a shooting generated signature in a two-point calculation, one must check that the signature is preserved.

Our purpose in using shooting is two-fold:

(a) rapid illumination of a new region, in order to understand its response to a signal, and also to determine subregions of interest,

(b) initialization for a two-point algorithm.

As we shall see later, our two-point algorithm is of the global (or bending) variety, in contrast with more common shooting and receiver-binning ones. We will show that it is worthwhile to have such a hybrid algorithm, as has been amply demonstrated in practical two-dimensional implementations.

We describe now the fast shooting algorithm. Given a starting point $\eta_0 = (x_0, y_0, z_0)$ and a starting direction μ_0, we need to find the intersection of the ray with the first interface. We assume in what follows that interface I_j is described by $z - \psi_j(x, y) = 0$. Finding the intersection of the ray

$$\eta(t) = \eta_0 + t\mu_0 \tag{1.1}$$

with I_j amounts to solving for t the scalar equation

$$f(t) = \eta_3(t) - \psi_j(\eta_1(t), \eta_2(t)) = 0. \tag{1.2}$$

We use for this purpose a standard, high-quality code, ZEROIN [11], that combines bisection with a safeguarded cubically convergent iteration. Given an interval in t, $[t_l, t_r]$, such that sign

$f(t_l) \neq \text{sign } f(t_r)$, ZEROIN will return a zero $\bar{T} \in [t_l, t_r]$ to any (reasonable) desired precision. In order to obtain such an interval, we set $t_l = 0$ and provided that $\mu_0 \not\perp \nabla f(\eta_0)$

$$\tilde{t}_r = -f(\eta_0)/\langle \mu_0, \nabla f(\eta_0)\rangle, \tag{1.3}$$

where $\langle , \rangle$ denote the vector innerproduct.

The rationale for this choice is given by the Taylor expansion

$$0 = f(\bar{\eta}) = f(\eta_0) + \langle \nabla f(\eta_0), \bar{\eta} - \eta_0\rangle + \cdots,$$

where $\bar{\eta} = \eta_0 + \bar{t}\mu_0$ is the desired intersection. If we drop the high-order terms, it follows that $\tilde{t}_r$ may be a sensible first approximation to $\bar{t}$. If sign $f(\eta_0) \neq$ sign $f(\eta(\tilde{t}_r))$, then $t_r \leftarrow \tilde{t}_r$, and we are ready to enter ZEROIN, otherwise a search is activated. Observe that this procedure is exact for plane interfaces (i.e. $\bar{t}_r = \tilde{t}_r$), and quite accurate if the curvature of I_j is small in the region of interest.

Once the intersection is calculated, Snell's law is applied to obtain the direction of the next leg of the ray. If v_0, v_1 are the velocities before and after the contact with I_j, the new direction μ_1 is given by [28]

$$\mu_1 = \mu_0 + \left[\pm\sqrt{\left(u_1^2 - u_0^2 + \langle \mu_0, \nu\rangle^2\right)} - \langle \mu_0, \nu\rangle\right]\nu, \tag{1.4}$$

where $u_i = 1/v_i$, $\| \mu_i \|_2 = u_i$, and ν is the unit normal to the interface at $\bar{\eta}$, i.e.

$$\nu = \nabla f(\bar{t})/\| \nabla f(\bar{t}) \|_2. \tag{1.5}$$

The $+$-sign in front of the square root holds for transmission, while the $-$-sign holds for reflection and $\| \cdot \|_2$ stands for the Euclidean vector norm. Observe that for reflected rays $u_0 = u_1$, and therefore (1.4) simplifies to

$$\mu_1 = \mu_0 - 2\langle \mu_0, \nu\rangle\nu. \tag{1.6}$$

If $u_0 \neq u_1$, and the quantity below the square root is negative, then we have a supercritical angle, and no transmission is possible.

Once μ_1 has been computed, the process continues in the same fashion with μ_1 taking the place of μ_0 and so on, untile I_R is reached. If at any stage the ray leaves the geologic window B, the process is interrupted (we say that the ray is clipped).

The cost of this algorithm is divided between the evaluation of the interfaces required by the sign change search, and by those used by ZEROIN and Snell's law, plus some additional arithmetic operations. So if a ray touches k interfaces and it uses s search evaluations and zr ZEROIN evaluations per interface, the cost will be approximately

$$k * [(s + \text{zr}) \text{ evals.} + \text{sqrt} + 12\text{P} + 8\text{A} + 22 * \text{zr} * (\text{P} + \text{A})], \tag{1.7}$$

where we have assumed that all the interfaces have the same evaluation cost, and P stands for product or division, while A stands for addition or subtraction.

1.3. Two-point initialization with shooting

Most of the problems we will be concerned with require source-to-receiver ray tracing. We will assume, for simplicity in the description, that a rectangular grid of equally spaced receivers R_{ij} is given on the free surface. We introduce now a more precise description of the model geometry in

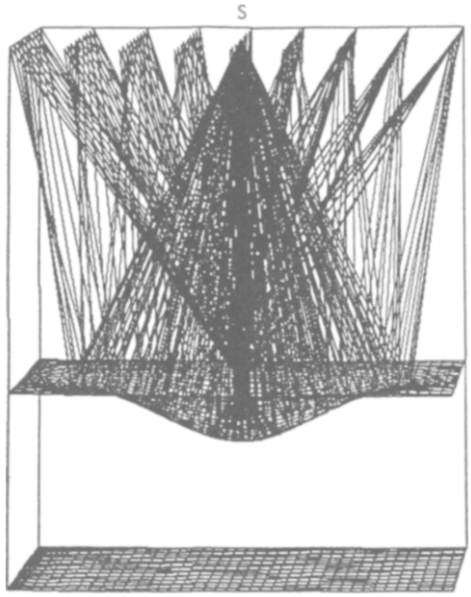

Fig. 2. Rays starting at source S, bouncing on a curved interface and producing a triple coverage of a 9×9 equally spaced net of receivers in the free surface.

Cartesian coordinates (z positive downwards). The geological region B (three-dimensional window) is described by the extreme values of the coordinates XMIN, YMIN, ZMIN, XMAX, YMAX, ZMAX. The free surface corresponds to the plane $z = \text{ZMIN}$, and therefore the receiver positions must be contained in the rectangle XMIN, YMIN, XMAX, YMAX. Let XSP, YSP be the uniform spacings between receivers in the x- and y-directions, respectively.

We also introduce an auxiliary plane $z = \text{ZSCR}$, parallel to the (x, y)-plane, which will usually be located near the source $S = (x_0, y_0, z_0)$. This is the control screen (CS) plane. On it we will place a number of points, in a way to be described below. Joining each one of these points P with the source S will provide a set of initial directions $\mu_0 = P - S$ to shoot rays through the structure. In analogy with computer graphic applications, we will sometimes refer to these points as 'pixels'.

Given a complicated geological region, due to velocity variations (in the general case), or to interface curvature, we may have different families of rays illuminating repeatedly the same receiver region (Fig. 2). Very often, these families start as very thin beams, that subsequently scatter due to region complexities. Thus, the directions of interest can be expected to be clumped on isolated spots, both in direction space and in their corresponding imprints in the control screen CS.

Most algorithms based exclusively on shooting rays saturate a given cone of directions with a uniformly distributed family of rays, which we can mimic by giving a uniform distribution of points P in our control screen CS. In order to ensure trapping all receivers on the surface, the resolution on the CS will have to be that corresponding to the minimum spacing necessary to have at least one shot ray landing near each receiver.

From our discussion above, finding an adequate distribution of pixels on the CS is a problem akin to adaptive mesh distribution (in two dimensions). Since generating and controlling an

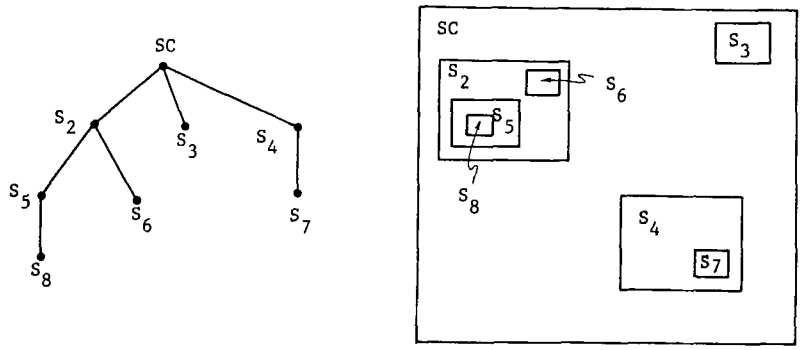

Fig. 3. Example of tree structure and screens geometry.

arbitrary distribution of pixels is far too complicated, we have settled on a compromise that has been useful in other applications. We will consider a hierarchical system of rectangular sub-screens, each with its own resolution. This system will be organized as a tree.

In this way we will be able to provide enhanced resolution locally where needed, and use coarser meshes otherwise, at a considerable savings in the number of searched directions. By choosing an organization based essentially on uniformly distributed sub-nets, we can keep the overhead low in the control part of the algorithm, and also leave the door open to the use of parallel architectures.

We introduce now the tree of two-dimensional screens that will form our variable resolution control screen. Each screen is described by the six numbers: SXMIN, SYMIN, SXMAX, SYMAX, SELX, SELY. The first four indicate the extent of the rectangular screen, while the last two indicate the spacing of the pixels in the coordinate directions, and therefore are responsible for the resolution of that particular screen. From these numbers we can easily derive:

$$\text{NXSC} = (\text{SXMAX} - \text{SXMIN})/\text{SELX},$$

$$\text{NYSC} = (\text{SYMAX} - \text{SYMIN})/\text{SELY},$$

the number of pixels in each direction.

The definition of the tree structure is based on set inclusion. We assume that if two screens S_i, S_j overlap, then either $S_i \subset S_j$ or $S_j \subset S_i$. With this assumption, we will say that a screen S_i is a child of screen S_j if $S_i \subset S_j$. Figure 3 gives an example of the tree structure and corresponding geometry. Given a control screen of the form just described, we observe that there may be regions of CS multiply covered at various resolutions. A fundamental convention is that the valid resolution for a search in the neighborhood of a point P, is that corresponding to the smallest screen to which P belongs.

We use several modes of search with different degrees of intelligence. The dumbest one starts on the main level screen CS, and proceeds through the pixels row-wise. As soon as a point penetrates a sub-screen, the search is continued in the sub-screen at the new resolution. Physical points P are 'rounded' to the nearest pixel in the current screen, if necessary. When all the pixels of a screen have been inspected this fact is recorded. Whenever a ray is successfully continued and reaches the free surface within 'rounding' distance of a receiver, the search is interrupted and control is passed to the two-point algorithm.

An array CSINC(I, J, K) is used to record the pixels already inspected. K refers to the screen ID, while I, J indicate the pixel in that screen. This is not a very economical use of storage, but it is the simplest to manipulate. If storage becomes an issue, then this and similar arrays can be highly compressed by using other type of data structures, such as linked lists. In any case, we initialize CSINC to 0, and record any pixel that has been inspected by a 1 in the appropriate spot.

Observe that successful two-point rays will also have an initial direction that may be different from any previously shot direction, and therefore this direction will need also to be marked as inspected. In order to avoid repeated calculation of the same two-point rays (an expensive affair), we keep a record for each receiver of the initial directions (as physical coordinates on CS) of successful arrivals. Thus, if an initial direction for a two-point calculation is 'near' to that corresponding to an earlier arrival to the same station, then that calculation is skipped.

This information will also be useful for correlating arrivals into families of coherent signals corresponding to smooth branches of the arrival time surface. This is an important step in the reconstruction of interfaces from arrival time data (time-to-depth migration). Also, in the least-squares inversion of model parameters, where the whole ray-tracing process may have to be repeated several times with small changes in the model parameters, the information on initial rays for two-pointing can be profitably used in order to diminish the overall cost. That is to say, the complicated search procedure needs to be activated only in the first iteration, while later on the saved converged rays can be used to initiate two-pointing in a slightly changed model (continuation on model parameters).

1.4. Adaptive control screen generation

Currently we have implemented a fairly simple way of reading interactively the information corresponding to a tree of screens. Ideally, we would like to generate this information automatically. In adaptive mesh refinement for multivariate representation of surfaces, or for solving partial differential equations, a guiding principle is that of choosing a nonuniform mesh that equidistributes an appropriate monitor function, which usually depends upon the solution to the problem (that is where the adaptive aspect comes in) [32,34]. Monitor functions are such that they induce local refinements of the net where they are large, while allowing local coarse meshes where they are small. What is an appropriate, or even natural, monitor function for our current problem?

If we consider the mapping between the control screen and the receiver plane induced by the rays, we see that the relevant quantity is the ratio $\sigma(P) = A_s/A_r$, where, for a point $P \in \mathrm{CS}$, A_s, A_r are respectively the areas spanned at So and R by an infinitesimal tube of rays, centered at the ray defined by the direction $\mu_0 = P - \mathrm{So}$. The function $\sigma(P)$, is nothing but the Jacobian of the ray mapping, while $\sigma^{-1}(P)$ is known as the geometrical spreading of the ray, which measures (through an energy conservation argument) the variation in amplitude of a unit signal due to geometric effects. This is a quantity that we already obtain for source-receiver rays, as part of amplitude calculations [28].

Thus, given $\sigma(P)$ on CS, and assuming that we want at least one ray in each rectangle (of area $A_r = \mathrm{XSP} * \mathrm{YSP}$) of the receiver net, then the local mesh size on CS would have to be such that the area of the resulting sub-rectangles is

$$A_s(P) \leqslant A_r \sigma(P). \tag{1.8}$$

If for a direction $(P - \mathrm{So})$, the corresponding ray fails to arrive at the receiver rectangle (for any reason), then we set $\sigma(P) = \infty$, thus indicating an uninteresting zone. It is enough to pick for ∞ any number larger than Total Screen Area$/A_r$. The $\sigma(P)$ so defined is our monitor function (actually σ^{-1} would be more like a traditional monitor function).

Obviously we do not known $\sigma(P)$ a priori, so an adaptive procedure will be required. Starting with a coarse mesh we can build up information about $\sigma(P)$, which in turn will indicate the areas of interest, and also those in which a mesh refinement (or mesh coarsening) is desirable, and thus the screen tree can be created and modified adaptively. Similar ideas are exploited in Section 2.4 for optimizing a time-to-depth migration scheme.

1.5. Two-point ray tracing

The algorithm and computer code RAYT3D, for solving two-point ray-tracing problems in three-dimensional general piecewise smooth inhomogeneous media have been presented in detail elsewhere [28]. We will recall here only the minimum material necessary for our current discussion.

RAYT3D solves the two-point boundary value problem for the ray equations, a coupled system of six ordinary differential equations of the first order for each of a family of source-receiver pairs and a given ray signature. RAYT3D uses the general purpose boundary value problem solver PASVA4 [24], which has adaptive mesh, variable order, and global error estimation capabilities.

PASVA4 solves a finite difference approximation to the ray equations in global form, i.e. as a coupled system of algebraic equations. As soon as the velocity in one region traversed by the ray is nonconstant, these equations are nonlinear, and the corresponding ray leg is curved. In regions of constant velocity, the code does not introduce any mesh points, but of course, for general curved interfaces, the ray-interface intersection equation and Snell's law are themselves nonlinear.

The resulting system of equations is essentially block tridiagonal, with some further sparseness in the off-diagonal blocks. This system is solved by a safeguarded Newton like iteration. The systems of linear equations that arise at each Newton step are carefully handled by a special Gaussian elimination algorithm that produces no fill-in in the LU decomposition. The block size is 6×6, the same as the size of the differential system, or 7×7 when we include travel time computation.

For a ray with k legs, the operation count for the linear equation solver is approximately

$$\text{Op. LES} = 216 * l * k, \tag{1.9}$$

where $l - 1$ is the number of interior mesh points in each leg. l is not necessarily the same for each leg, so in general we would have Op. LES $= 216 * n$, where n is the total number of mesh points.

To compare with shooting in the piecewise constant velocity case, we set $l = 1$ in (1.9), and let N be the number of Newton iterations performed. The operation count for computing one ray is then approximately

$$N * k * [\text{Interface evals.} + 240(\mathrm{P} + \mathrm{A})], \tag{1.10}$$

where we have included the work for the computation of Snell's law.

We are not including here Jacobian evaluations, since Jacobians are not updated all the time in our algorithm, but roughly speaking a term like $90 * k * (P + A)$ should be added per Jacobian evaluation. In the case we are considering, no mesh refinement, deferred correction or error estimation is performed, so this is the total operation count. Typically, a value $N = 3$ is ample for the accuracy being sought.

If we assume the number of interface evaluations $zr = 5$ in (1.7), then two-pointing with RAYT3D will take about ten times more operations than the fast shooting algorithm. Of course, shooting does not solve the two-point problem, and therefore the comparison has to be made between pure shooting with uniform resolution and our hybrid system.

A simple but realistic example shows the relative merits of the two approaches more clearly. Assume that we have a 10×10 regular array of receivers with a source-region system that produces a two-fold coverage. Assume that one of the ray families requires a regular 10×10 spacing in a screen of area $A = 1 \times 1$, while the other family has directions concentrated on a square of area $A = 0.01 \times 0.01$, also with uniform 10×10 coverage. Thus, the finest resolution required is 0.01×0.01, which transported uniformly to the whole screen would require shooting 10,000 rays. Using the variable resolution cuts down the maximum number of rays to about 200. Since some, if not most of the control pixels will be filled in by the two-point algorithm, which should compute 200 rays, we see that there will be a factor of 50 more shot rays in the fully uniform, saturation approach, as compared with the selective shot/two-point combination.

One of the problems of any method for solving discrete nonlinear boundary value problems, is the need for starting values. In our case, we need a guess for the entire discrete ray. The closer this guess is to a solution, the easier and less expensive it will be to solve the nonlinear equations. RAYT3D, in its current automated version, uses a starting trajectory provided by the fast shooting routine. Since we have no real control on where a given shot will land in the receiver window, RAYT3D will usually start at an arbitrary receiver R_{ij}. Once a two-point ray has been successfully calculated between So and R_{ij}, we move to a neighboring receiver using the previously computed trajectory as a starting ray. This is known as receiver continuation, and the guess can be improved sometimes by using an Euler correction [23]. Even more simply, after two rays have been computed we can use their difference to predict the new trajectory [29]. Neither procedure will produce a good approximation in the neighborhood of caustics or wave front folds.

In any case, RAYT3D will continue to move regularly through the receiver net until either it is exhausted or a failure occurs. A failure may be due to numerical causes, but most often it arises because of physical reasons; as pointed out above, near caustic surfaces, shadow zones, or any irregularity in the wave front we can expect difficulties, like lack of existence of solutions, multiple solutions which are close together and thus, produce ill-conditioned Jacobians, etc. At such a point we switch back to the shooting search, until we lock into a new receiver, and so on until all the control pixels have been inspected. In this way we have been able to automate the initialization process for RAYT3D, thus increasing considerably its value for practical applications, and making it possible to capture multiple solution branches with no user intervention.

Observe that by construction, the shooter will also provide the complete signature of the ray needed by RAYT3D to set up the ray equations correctly. This will be true even in the general non-layered case. However, in that case, as the ray converges from the initial guess to the final solution, it may happen that its signature changes, and should therefore be checked for validity.

We show in Fig. 4 a two-dimensional example of a complicated, non-layered geologic region

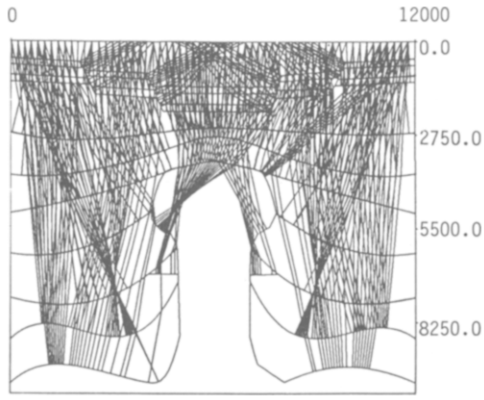

Fig. 4. An example of changes of signature for two-point normal incidence ray tracing in two dimensions.

with a large number of normal incidence rays (i.e. rays with coincident source-receiver pairs) traced, where the phenomenon of signature change within coherent ray families is apparent. This phenomenon will be clearly exacerbated in three dimensions for regions of comparable complexity. In the two-dimensional case [29] we have chosen to interrupt two-pointing if a change of signature occurs, since it is quite difficult to continue an iteration in which we may need to insert or delete ray legs. This will be most likely our choice also in three dimensions.

In the inhomogeneous case we supplement the constant velocity shooter with the shooting option on general media available in RAYT3D. That is, we use an approximate constant velocity model to get a starting trajectory with the fast shooter and the search algorithm described above, and then use this ray to start the RAYT3D shooter, in order to obtain a ray in the desired inhomogeneous model. If successful, this is in turn used to start the two-point procedure. Since the shooting option in RAYT3D is as expensive as the two-point one, now the start and restart procedure can be quite costly. In more recent work [38], we have replaced this shooter by a Runge–Kutta–Fehlberg initial value problem solver coupled with a zero finder, which has provided improvements of up to 50% over those reported later on in this paper.

We feel that the technique we have described, of coupling shooting with two-pointing, with overall control maintained through a screen with variable resolution, may be valuable in other calculations involving families of solutions of nonlinear boundary value problems where bifurcation phenomena are present.

2. Progressive time-to-depth migration and material properties determination from nonzero offset data

2.1. Introduction

We consider in this section the problem of determining the geological structure of a three-dimensional region of the earth from prestack seismic reflection data collected on the surface. The methods we will describe combine two-point ray tracing with least-squares tech-

niques to determine an earth model that is consistent with the data. The type of data used will be travel times and peak amplitudes of the various waves reflected by material discontinuities in the structure. The source of the waves can be man-made (explosions, vibrators) or natural (earthquakes), with its backscatter collected at seismic stations or geophone layouts.

We will assume that the medium is composed of curved layers whose boundaries mark material discontinuities, and in the most general case we will allow the interlayer velocities (of P or S waves) to be smoothly varying functions of position. We will assume also that the data has been interpreted, i.e. that coherent signals have been identified and picked, and that what is desired is to migrate these time surfaces to their correct spatial position, determining in the process the corresponding layer velocities. Adequate parameterization of interfaces and velocity models is an important issue that will not be explored in depth in this paper, since we want to establish first the main algorithms and procedures.

The process starts from the free surface and goes down layer by layer. Given a source that generates an elastic wave, we assume that the velocities in the first layer are known, and that a coherent signal reflecting from the first interface has been collected on a number of receivers on the free surface. Then we show how to use ray tracing to recover (approximately) the points of intersection of the seismic rays between the source and the receivers with the unknown reflecting interface; we also determine in this process the normal vector to the interface at the intersection point. We then use that information to fit a parameterized model of the interface.

This may require linear or nonlinear least-squares techniques, depending on the complexity of the model, but we observe that the ray-tracing and model-fitting stages are consecutive not concurrent. Thus, this is a reasonably inexpensive procedure, as we will see in more detail later. We thus refer to this process as 'static inversion'.

Once the interface is determined, the peak amplitudes are used to determine the material properties in the subsequent layer. For this purpose, the effect of geometrical spreading and reflection-transmission in previous known layers is factored off the amplitude data, which will then correspond only to the elastic impedance contrast at the reflecting layer. With this data we can then do a least-squares fit to determine unknown parameters in a parameterized material model. This least-squares process can again be linear or nonlinear, depending upon the class of model parameterization chosen.

Having performed those two steps, we can proceed to determine the next interface, assuming again that a coherent set of arrivals is available, and so on. This process will, naturally, become more inaccurate as we get deeper into the earth, but its main purpose is to help us generate a preliminary model in an inexpensive fashion. This model will be refined later by more sophisticated means (see Section 3). Some of these ideas have been discussed earlier (see for instance [16,17,35]), but we believe that our implementation is novel and more efficient.

2.2. Recovering the first interface in an homogeneous medium

The principal elements in our procedures will be the source of elastic waves, So, and a rectangular array of receivers R_{ij} $(i = 1,\ldots,\mathrm{NR}_y;\ j = 1,\ldots,\mathrm{NR}_x)$ located at $(x_{Rj},\ y_{Ri})$, which we assume to lie on a flat free surface for simplicity in the discussion. Topography, or other kinds of receiver layouts, like those used in vertical seismic profiling or in offshore surveys, can be accomodated by the same methodology with minor changes.

Let us call $T(x, y) \equiv T(x, y, 0)$ the time taken by an elastic wave (P or S) to go from the source So to the point $(x, y, 0)$ on the free surface, after reflecting from a curved interface I_2 (we call the free surface I_1). Identification of this particular signal in the seismograms collected at the receivers R_{ij} will give corresponding observed values T_{ij} of the travel time function $T(x, y)$.

An important fact used in all the algorithms below, and which is valid for smooth travel time surfaces, is that for $\eta = (x, y, z)$, $[\nabla_\eta T(x, y)]$ is the direction of arrival of the seismic wave to the point $(x, y, 0)$. This follows since wave fronts are described by the level surfaces of the travel time function $T(x, y, z) = \tau_0$, and the seismic rays (in an isotropic medium) are normal to the wave fronts, thus coinciding in direction with ∇T.

By taking differences of the discrete data T_{ij}, we can generate approximations to T_x, T_y, and therefore to the direction μ_0 of the seismic ray arriving at (x, y). In fact, from the eikonal equation (with $u(\eta) = 1/v(\eta)$, $v(\eta)$ the velocity of wave propagation)

$$\| \nabla_\eta T(\eta) \|_2 = u(\eta), \tag{2.1}$$

we obtain $T_z(\eta) = \sqrt{u^2 - T_x^2 - T_y^2}$, which gives the third component of ∇T.

Alternatively, we can use the values $[(x_j, y_i), T_{ij}]$ first with fixed i, to generate cubic splines (say) $\mathrm{Sp}_i^{\mathrm{r}}(x)$, $i = 1, \ldots, \mathrm{NR}_y$, and then do the same in the normal direction to obtain $\mathrm{Sp}_j^{\mathrm{c}}$, $j = 1, \ldots, \mathrm{NR}_x$. These two sets of one-dimensional splines can then be used to obtain the approximate partial derivatives

$$T_x(x, y_i) \approx \frac{\mathrm{d}}{\mathrm{d}x}\left[\mathrm{Sp}_i^{\mathrm{r}}(x)\right], \quad T_y(x_j, y) \approx \frac{\mathrm{d}}{\mathrm{d}y}\left[\mathrm{Sp}_j^{\mathrm{c}}(y)\right],$$

and they can also be used to fill smoothly any gaps in the data, or even to densify the original data set, for instance in regions of high variation in the time surface (for other alternatives and more details see for instance [18]).

For small NR_x, NR_y, interpolatory splines will be adequate, although care has to be taken in regions where there are abrupt variations in the data, since they may result in unwanted oscillations. In such cases it may be wiser to use shape preserving interpolants [12], which although generally less smooth globally, represent more faithfully the data. For large NR_x, NR_y, we may want to replace interpolatory splines by their least-squares counterparts, with some reasonable automatic knot placement strategy.

In all cases it is wise to check a posteriori that the spline representation is adequate. We show in Section 2.7 the results of our implementation of an automatic procedure to classify multiple arrivals into separated coherent signals, its representation by one-dimensional cubic splines, and an a posteriori visual check of the quality of the overall procedure, showing how easy it is to introduce gross errors, particularly in regions of high curvature of the time surfaces. This is a crucial part of the migration algorithm, since if the data is not organized into coherent time surfaces, the procedures will not migrate it correctly.

As is becoming more common to collect data with three component instruments [36], all these complexity may be alleviated in the future.

Taking the negative of the direction μ_0 generated by differentiating the time surface, we can shoot a ray backwards, and if we knew the interface I_2, we could reconstruct the entire ray back to the source So. However, this is not the case: interface I_2 is our objective.

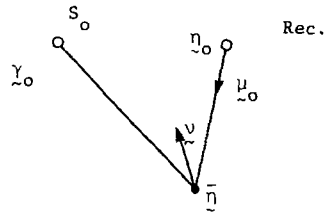

Fig. 5. Geometry for reconstruction of ray path from travel time data, for a one layer homogeneous medium.

For a constant velocity layer we can establish a simple linear equation for the appropriate point of reflection $\bar{\eta}$. Let η_0 be the receiver position, μ_0 the negative of the arriving direction, γ_0 the source position (see Fig. 5).

The straight line segment that starts at η_0 and is defined by the direction μ_0 can be represented as

$$\eta = \eta_0 + s\mu_0,$$

for a scalar parameter s. We want to determine s, so that traversing the ray path $[\gamma_0, \bar{\eta}, \eta_0]$ with speed v_1 takes time $T(\eta_0)$. In other words,

$$\| \bar{\eta} - \eta_0 \|_2 + \| \bar{\eta} - \gamma_0 \|_2 = v_1 T(\eta_0),$$

or

$$\| \eta_0 - \gamma_0 + \bar{s}\mu_0 \|_2 = v_1 T - \bar{s},$$

which after a few manipulations leads to

$$\bar{s} = 0.5\big[v_1^2 T^2 - \| \eta_0 - \eta_0 \|_2^2\big] / \big[T + \langle \eta_0 - \gamma_0, \mu_0 \rangle\big].$$

Since $v_1 T$ is the total distance traveled by the ray, $\bar{s}$ is clearly positive and $\bar{\eta}$ must lie on the unknown interface I_2. Furthermore, the bisector ν is the normal to I_2 at $\bar{\eta}$,

$$\nu = a + b,$$

where $a = -\mu_0$, $b = (\gamma_0 - \bar{\eta}) / \| \gamma_0 - \bar{\eta} \|_2$.

Thus, doing this computation for each piece of data $[T_{ij}, \mu_{0ij}]$, we generate a corresponding set of points on, and normals to, the unknown reflecting surface I: $[\bar{\eta}_{ij}, \nu_{ij}]$. This data is used to fit an analytical surface in the least-squares sense, which will be the migrated surface. That is, from a time surface picked from recorded seismograms, we have reconstructed the correct spatial position of the reflector producing such signals. Further details of the least-squares fit are discussed in Section 2.5.

2.3. Recovery of deeper interfaces

We assume now that the structure (material properties and interfaces) is known down to, but excluding interface I_D. The objective is to reconstruct I_D from a set of arrivals that have reflected from it. So, in principle, the initial data is, as before, a set of arrivals and directions constructed from them by numerical differentiation: $[T_{ij}, \mu_{0ij}]$, plus an assumed signature for the rays that generated them. For simplicity, we consider rays that reflect only once on I_D; outside of that the path can be completely general. There is no restriction either with respect to

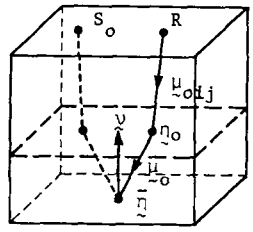

Fig. 6. Schematic representation of deep layer migration for three-dimensional nonzero offset data.

the velocities: the procedure we will describe is valid in a completely general medium. However, in the constant velocity case there can be considerable savings by using the fast shooting algorithm described in Section 1.2, so we consider that first.

2.3.1. Constant velocity case

For each piece of data, we shoot from the receiver with initial direction $-\mu_{0ij}$, up to and past the last known interface (with the given signature); this generates an intersection η_0 and a direction μ_0, such that $\eta = \eta_0 + s\mu_0$, ($\| \mu_0 \|_2 = 1$) is as in the one-layer case (see Fig. 6).

After this we solve the two-point ray-tracing problem between the source So and the point $\eta(s)$, where s is also to be determined. To account for the unknown parameter s we use the additional constraint

$$\text{Total Travel Time} = T_R.$$

If τ_R is the travel time between R and η_0, and τ_{So} is the travel time from the source So to η, then we can write this constraint as

$$\tau_R + \tau_{\text{So}} + s/v - T_R = 0 \tag{2.2}$$

where v is the velocity in the layer above I_D.

This shows one of the strengths of the two-point ray procedure and the underlying general purpose two-point boundary value problem solver. The two-point ray tracer includes as one of the dependent variables the travel time along the trajectory, so (2.2) is just an end condition involving also an unknown parameter which can be easily accomodated by the general solver PASVA4 [28] with minor changes.

We now show that it is worth using the hybrid procedure; in Section 1 we showed that two-pointing required ten times more work than the fast shooting algorithm. Thus, the work for the hybrid algorithm (STP) is

$$\text{STP} = w + 0.1lw = 1.1lw,$$

where $2l$ is the number of legs between source and receiver, and w is the work per leg required by two-pointing. This compares very favorably with the work required by the full two-pointing formulation (GTP): GTP = $2lw$. Observe that in this case we cannot argue as in Section 1.5 that, globally, full two-pointing will fare better than indiscriminated shooting. This is an altogether different problem than the one discussed there.

2.3.2. Inhomogeneous media

In the variable velocity case we cannot use the fast shooter, and integration of the initial value problem for the ray equations becomes necessary. Of course, root finding has to be coupled with

the integration to take into account the direction discontinuities at the interfaces. An alternative to all this additional complication is to dispense completely with the combined procedure, and formulate the problem as just one global boundary value problem. Since we have not implemented these ideas we will not say anything further about the general inhomogeneous case until we discuss dynamical inversion procedures in Section 3.

2.4. Error estimation and mesh equidistribution

There are three sources of error in the procedure described in Section 2.3, namely
(1) errors in the travel time data,
(2) errors in the numerical differentiation of the time data, and
(3) errors in the ray-tracing algorithms.

We have essentially no control over errors of type (1). Of course the situation can be aggravated by erroneous picking of the data into coherent time surfaces. If the cubic spline procedure is used, in principle, the error in the partial derivatives for a sufficiently smooth time surface would be $O(\Delta^3)$, where Δ is the receiver net spacing. Of course, this error will be larger in regions where there is more rapid variation in the time surface and vice versa. An error equidistribution procedure [19,32] could be used here to select a variable resolution in the receiver net in order to increase the efficiency of the procedure without sacrificing accuracy, as discussed later on. In all cases, the point $\bar{\eta}$ on the unknown interface is calculated, while the bisector of the two ray legs defining the reflection point is the normal ν to the reflecting interface.

Errors in the partial derivatives of the arrival time surface affect the initial direction of the backward shooting, and these errors in turn will affect the whole trajectory, including what we are attempting to calculate: $[\bar{\eta}, \nu]$. Of course, this output data will also be affected by errors in the numerical integration of the ray equations, but this third source of errors is under good control in our procedures [26,28]. From the discrete time data and the chosen interpolation procedure, we can estimate the error in the gradient calculation at a point (x, y). For cubic splines, we have

$$\mathrm{Ie}(x, y) = \| \nabla T(x, y) - \nabla \mathrm{Sp}(x, y) \|_2 \equiv g(x, y)\Delta^3. \tag{2.3}$$

In the ray-tracing process we can evaluate the Backward Geometric Spreading $\mathrm{BGS}(x, y)$, which measures the variations in the ray variables produced by small variations in the initial direction $-\mu_0(x, y) \equiv -\nabla \mathrm{Sp}(x, y)$. This calculation amounts to solving the ray equations, linearized around the computed ray, with appropriate conditions at the receiver end (x, y). This can be easily and economically done when a full boundary formulation and the PASVA4 code is used, since upon convergence we have available the discrete Jacobian decomposed in LU form, and a discrete approximation to the backward spreading can be obtained with just a back solve of a sparse linear system [28]. The larger the spreading, the more sensitive the ray trajectory will be to errors in the direction at the receiver (x, y).

In what follows we use $\mathrm{BGS}(x, y)$, to represent the value at $\bar{\eta}$ of the backward geometrical spreading corresponding to a ray received at (x, y). Thus, we can define a sensitivity function by combining (2.3) with the BGS:

$$\mathrm{Se}(x, y) = \mathrm{Ie}(x, y) * \mathrm{BGS}(x, y) = g(x, y) * \mathrm{BGS}(x, y) * \Delta^3. \tag{2.4}$$

Essentially, $\mathrm{Se}(x, y)$ is a measure of the errors at $\bar{\eta}$ produced by errors in the numerical differentiation of the data, taking into account also the effect of the ray equations. In symbolic form, if $Y(\mu_0)$ represents the values of the dependent variables of the ray problem (ray position, ray direction and partial travel time) at the reflection point $\bar{\mu}$, then $\mathrm{Se}(x, y) = \| Y(\mu_0 + \delta\mu_0) - Y(\mu_0) \|$, where $\delta\mu_0 = g(x, y)\Delta^3$, and $\mathrm{BGS}(x, y)$ is the norm of the (Fréchet) derivative of $Y(\mu_0)$ with respect to μ_0.

The principle of mesh equidistribution [32] establishes that if a variable mesh spacing $\Delta(x, y)$ can be chosen, so that $\mathrm{Se}(x, y) = \mathrm{dl}$, for a given constant dl, then the number of mesh points will be the minimum required to achieve such a level. We call dl, the desired level of equidistribution, and since $\mathrm{Se}(x, y)$ is a measure of the error in the quantities of interest, we see that for an equidistributed mesh the error is constant and of size dl. From (2.4) we obtain, for the optimal mesh spacing:

$$\Delta(x, y) = (\mathrm{dl}/[g(x, y) * \mathrm{BGS}(x, y)])^{1/3}. \tag{2.5}$$

It is well known that exact equidistribution, as indicated in (2.5), is very hard to achieve and usually too costly. Approximate (or asymptotic) procedures are most cost efficient and usually do a commendable job [19,26,34]. In particular, one such procedure is used in the solvers PASVA for efficient automatic mesh placement in the solution of one-dimensional boundary value problems.

We should not forget the nature of the current problem, and the limitations imposed by data collection, which in most cases would have been made a priori, independently of this analysis, leaving no room to refine the mesh beyond a certain level. At this point, a practical digression on the amount of data produced by a typical three-dimensional reflection survey will show why we may want to be conservative in its use, and that what we are really suggesting is the possibility of using less than the available volume of data.

Let us assume that the region of interest is 1×1 km square, on which a square grid of 100 geophones has been laid, spaced by 100 m in both directions. Assume that only ten shots have been set, and that primary reflections from an underground reflector have been received at all geophones. This makes for a minimum of 1000 arrivals. In fact, for curved interfaces there is a high likelihood of receiving multiple (primary) arrivals at some or all of the receivers, coming from different parts of the reflector. This is really a small example of what is actually done in practice, and of course, at this point we are not discussing issues of noise in the data. One way to improve the signal to noise ratio is by stacking the data and producing what amounts to a normal incidence section. This effectively reduces the volume of data considerably, and when the geology allows it, is a very useful procedure. Unfortunately, for complex regions this processing can produce more harm than good, and what we are interesting in developing here are methods for pre-stack migration.

An interactive self-adaptive procedure for choosing as small an amount of data as required, would start by taking a subset of receivers and performing the error estimation indicated above. If the error level is satisfactory, then we are done, otherwise we can use the sensitivity function $\mathrm{Se}(x, y)$ to introduce new receivers in those areas where $\mathrm{Se}(x, y) \gg \mathrm{dl}$, within the limits of what is available, and then repeat the process. We can also simultaneously remove receivers where $\mathrm{Se}(x, y) \ll \mathrm{dl}$ (i.e. where there is too much accuracy), although this should be done with caution, since the spline interpolation and gradient computation can be affected globally by local changes, thus introducing unwanted instabilities.

Combining (2.4) and (2.5), the practical refinement criterium would be:

If $\mathrm{Se}(x, y) = k * \mathrm{dl}$, $k \gg 1$, then refine the mesh around (x, y) so that $\Delta_{\text{new}} = \sqrt[3]{k}\,\Delta_{\text{old}}$.

For the example discussed above, if $\Delta_{\text{old}} = \Delta/2$ (i.e. we use an initial 5×5 mesh of receivers, requiring only $1/4$ of the available data and rays to be traced), then our only possibility is to make $k^{1/3} = 2$, and therefore this discrete refinement will be triggered when $k \approx 8$. We can refine at smaller levels of k, but that in general will be unnecessary.

2.5. *Least-squares representation of surfaces*

Once the set $[\bar{\eta}_{ij}, \nu_{ij}]$ of interface data has been created, following the procedures of Sections 2.2–2.4, we want to obtain an analytical representation of the interface that is compatible with the ray-tracing modeling system, i.e. at least twice differentiable, and in the case we are considering, single valued (as a function of x, y) over the whole window of interest.

We will consider interface models of the form

$$z = I(x, y; \boldsymbol{a}, \boldsymbol{\alpha}) \equiv \sum_{k=1}^{K} a_k \sigma_k(x, y; \boldsymbol{\alpha}), \tag{2.6}$$

where some or all of the parameters $\boldsymbol{a} = [a_k]$, $\boldsymbol{\alpha} \in \mathbb{R}^P$ are to be determined so that $z = I(x, y; \boldsymbol{a}, \boldsymbol{\alpha})^k$ fits the given data in the least-squares sense.

Let $\bar{\eta}_{ij} = (x^{\circ}_{ij}, y^{\circ}_{ij}, z^{\circ}_{ij})$ and $\nu_{ij} = (\nu^{\circ}_{ij1}, \nu^{\circ}_{ij2}, 1)$, where we have chosen a convenient normalization for the normal vectors ν_{ij}, and the superscript stands for observed. $\bar{\eta}_{ij}$ is to match $(x_{ij}, y_{ij}, z_{ij} = I(x_{ij}, y_{ij}; \boldsymbol{a}, \boldsymbol{\alpha}))$, while ν_{ij} should be matched with the gradient

$$\nabla\big(z - I(x, y; \boldsymbol{a}, \boldsymbol{\alpha})\big) = \begin{bmatrix} -I_x \\ -I_y \\ 1 \end{bmatrix}.$$

In other words, the least-squares problem to be solved is

$$\min_{\boldsymbol{a}, \boldsymbol{\alpha}} \sum_{i,j} \Big(\big[z^{\circ}_{ij} - I\big(x^{\circ}_{ij}, y^{\circ}_{ij}; \boldsymbol{a}, \boldsymbol{\alpha}\big) \big]^2 + \big[\nu^{\circ}_{ij1} + I_x\big(x^{\circ}_{ij}, y^{\circ}_{ij}; \boldsymbol{a}, \boldsymbol{\alpha}\big) \big]^2 + \big[\nu^{\circ}_{ij2} + I_y\big(x^{\circ}_{ij}, y^{\circ}_{ij}; \boldsymbol{a}, \boldsymbol{\nu}\big) \big]^2 \Big). \tag{2.7}$$

Thus, if we had N arrivals originally, we will have $3N$ equations for the least-squares process. Models of the form (2.6) are known as 'separable' [15], and very effective methods and software are available to solve the resulting nonlinear least-squares problem (2.7). Observe that (2.6) contains as special cases both the linear (no $\boldsymbol{\alpha}$) and the 'general nonlinear least-squares' (no $\boldsymbol{a}$) cases. Observe also that since

$$I_x(x, y; \boldsymbol{a}, \boldsymbol{\alpha}) = \sum_{k=1}^{K} a_k \sigma_{kx}(x, y; \boldsymbol{\alpha})$$

and similarly for I_y, we do not loose the separability by introduction of the gradients. Also, since the linear parameters are the same throughout, the complete separable problem (2.7) still has

only K functions, with the unusual feature that a different functional form appears after the first N and $2N$ 'observations'.

Following the notation of [15], we can rewrite (2.7) in vector-matrix form as

$$\min_{a,\alpha} \| \boldsymbol{b} - \Phi(\alpha)\boldsymbol{a} \|_2^2, \tag{2.8}$$

where

$$\boldsymbol{b} = \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \boldsymbol{b}_3 \end{bmatrix} \begin{matrix} N \\ N \\ N \end{matrix},$$

and $\Phi(\alpha)$ is a $3N \times K$ matrix partitioned in a similar fashion: $[\Phi_1, \Phi_2, \Phi_3]^T$. Here $\boldsymbol{b}_1 = [z_{ij}^\circ]$, $\boldsymbol{b}_2 = [v_{ij1}^\circ]$, $\boldsymbol{b}_3 = [v_{ij2}^\circ]$, and the columns Φ_{rk}, $k = 1, \ldots, K$ of Φ_r, $r = 1, 2, 3$ are

$$\Phi_{1k}(\alpha) = \left[\sigma_k\left(x_{ij}^\circ, y_{ij}^\circ; \alpha\right)\right], \qquad \Phi_{2k}(\alpha) = \left[\sigma_{kx}\left(x_{ij}^\circ, x_{ij}^\circ; \alpha\right)\right],$$

$$\Phi_{3k}(\alpha) = \left[\sigma_{ky}\left(x_{ij}^\circ, y_{ij}^\circ; \alpha\right)\right].$$

The variable projection method of [15] seeks a solution of (2.8) by first eliminating the linear parameters from the problem. It can be shown that the minimization in (2.8) is equivalent to

$$\min_{\alpha} \| (I - \Phi(\alpha)\Phi^+(\alpha))\boldsymbol{b} \|_2^2 = \min_{\alpha} \| P_{\Phi(\alpha)}^{\perp} \boldsymbol{b} \|_2^2,$$

where $\Phi^+(\alpha)$ is the generalized inverse of the matrix $\Phi(\alpha)$ and $P_{\Phi(\alpha)}^{\perp}$ is the projector over the orthogonal complement of the column space of $\Phi(\alpha)$, which motivated the name 'variable projections'. An immediate consequence of this step is the reduction in dimensionality of the parameter space, which in turn eliminates the need for initial values $\boldsymbol{a}^{(0)}$. After a solution α^* has been obtained, one computes the optimal a^* by a final linear least-squares step: $a^* = \Phi^+(\alpha^*)\boldsymbol{b}$.

The special algorithm for solving separable problems proposed in [15] and improved in [21] has been thoroughly analyzed in [33], and found to have faster convergence than the same minimization procedure applied in nonseparable form, for a comparable amount of work per iteration. An implementation of this algorithm was written along with the paper [15] by Pereyra. This implementation has been successively revised by J. Bolstad (1977) and R. Leveque (1978), who have introduced a number of improvements. This program, named VARPRO, has been widely distributed, and it has confirmed experimentally the theoretical results of [33]. The nonlinear least-squares techniques used optionally in VARPRO are either Gauss–Newton with step control or a Marquardt type algorithm. More recently, L. Kaufman and D. Gay have spliced the variable projection technique into a more modern unconstrained nonlinear least-squares solver: NL2SOL [9], and they have also versions that allow bound constraints on the nonlinear variables. In [22], Kaufman and Pereyra have introduced separable nonlinear least-squares problems with separable nonlinear equality constraints, describing also an efficient solution algorithm.

The conclusion is that there exists a considerable basis of available good quality software, at least to start our development. We will see later that in practice and even at a simple level, care will have to be taken with outliers, i.e. with observations containing large errors, and with the ill-conditioning inherent in many of these problems. Outliers can be introduced, for instance, by erroneous picking of the travel time surfaces, or by some of the error amplification mechanisms

discussed in Section 2.4. It is likely that procedures for robust nonlinear least-squares estimation, as those considered in [6] for the linear case, would be useful in this respect.

Later on, as we increase the complexity of the models, we may start to see the need for solvers that take sparseness and more general types of constraints into account. Presently, the representation (2.6) allows for considerable generality, from plane interfaces to a variety of curved shapes given in analytical form. One can envision a catalogue of pre-chosen functional forms (two-dimensional polynomials, trigonometric polynomials, exponentials, Gaussians, rational functions, etc.) from which it would be easy to select functions and their partial derivatives to suit many interface representations.

If no preconceived global analytic form is available (or desirable), (2.6) can accommodate also multivariate spline approximations with preassigned knots (only the a's to be determined), or with free knots. Splines under tension, or other types of shape preserving splines and tensor products [7,19,31] can be used to avoid introducing unwanted oscillations in regions of high gradients. Eventually, when more ellaborated models are considered, including surfaces that fold or terminate, we will need to rely on parametric splines, and the formulation of [2] seems to be a good starting point.

2.6. Recovery of homogeneous material properties

Once an interface has been reconstructed and we have found a satisfactory representation through the procedures of the former sections, we proceed to approximate the material constants of the next layer. We would like to do that without further ray tracing.

In the course of the construction of I_D we calculate, for each ray, the geometrical spreading GE_{ij} and the product of all the elastic reflection and transmission coefficients at the shallower interfaces Tc_{ij}. If we knew the material properties (v_p, v_s and ρ) for the layer below I_D, we could have also calculated the reflection coefficient corresponding to I_D, RID_{ij}. The peak amplitude A_{ij}, of the arrival under consideration, is given as

$$A_{ij} = GE_{ij} * Tc_{ij} * RID_{ij}(v_p, v_s, \rho; \sigma_{ij}),$$

where σ_{ij} is the angle of incidence of the ray (in the plane formed by the incident ray, the reflected ray and the normal to the interface). Thus, we can consider

$$RID^o_{ij} = A_{ij}/(GE_{ij} * Tc_{ij}) \tag{2.9}$$

as the observed reflection coefficient.

This is a fairly simple minded approach to an extremely complex phenomenon. These coefficients correspond to a plane wave approximation, and in general one may need to consider higher-order approximations. Attenuation and anisotropy are not considered, and it is not clear if this method will extend to inhomogeneous materials. However, recall that all we are trying to construct here is an approximate model that should be refined later by more comprehensive techniques if so required.

$RID^c_{ij}(v_p, v_s, \rho; \sigma)$, for given v_p, v_s, ρ, can be calculated by using Zoeppritz equations, and it will be in general a nontrivial function of its arguments for nonnormal angles of incidence ($\sigma \neq 0$). Thus, we can formulate another nonlinear least-squares problem to determine the best fit of the unknown parameters v_p, v_s, ρ:

$$\min_{v_p, v_s, \rho} \sum_{ij} \left[RID^o_{ij} - RID^c_{ij}(v_p, v_s, \rho; \sigma_{ij})\right]^2.$$

Observe that the only variables here are the material properties of the layer below I_D, which do not affect the ray trajectory at all; therefore σ_{ij} does not change, and no further ray tracing is necessary.

In order to test a variety of software, and since we do not want to compute analytical derivatives with respect to the parameters, we have chosen a different code for this task: LMDIF from MINPACK [13], an implementation of a Marquardt type algorithm which does not require derivatives of the objective functional. Once the material properties of the next layer have been identified, we can proceed to reconstruct the next interface, provided that arrivals reflected by it are available.

2.7. Implementation and test results

We have implemented some of the algorithms described in Sections 2.2–2.6, creating a number of new modules to perform the various tasks. These modules are complete programs that communicate through files, thus eliminating variable conflicts and side effects, simplifying maintenance and providing a limited, controlled environment for development of this complex prototype system. This design will facilitate running different parts of the calculation in different machines or processors. For production runs, some of these modules may be fused together when there is no need for human intervention. Also an interactive controller can be overlaid with ease.

We start by generating synthetic data using program AUTO3DRT from Section 1 on a known structure. This data consists of arrival times and peak amplitudes for rays that emanate from a given source, travel through the structure, reflect once on a deep interface and return to the free surface, to be recorded on a rectangular array of receivers. AUTO3DRT will perform this task automatically after some preliminary tuning of the control screen (see Section 1 for more details).

The time-offset output can be displayed in the form of synthetic seismograms (by SYNT3D), along either x- or y-lines. This is how real, nonzero offset, or pre-stack data will usually be available (in digital form), except for the absence of noise (a very important factor).

Because of the way in which AUTO3DRT searches and finds source-receiver arrivals, its output will have these arrivals in the order in which they were obtained, which is essentially arbitrary. It is easy to sort them by receiver, and when there is more than one arrival to the same receiver we can also sort them by arrival time. However, there is no guaranty that grouping all the nth arrivals will form a coherent time surface, since there may be gaps, either real (shadows, caustics), or artificial (failure of the solver).

Although this is only an artifact created by our desire to produce controlled data and the way AUTO3DRT operates, it is a real problem that must be solved correctly, and preferably automatically. We think we have succeeded in implementing a good selection algorithm to pick up coherent signals. This program, MIGDAT2, produces also the spline interpolants along x- and y-lines of the time data, as indicated in Section 2.2. MIGDAT2 will also be useful later on when we consider dynamic inversion with the ray tracing inside a nonlinear least-squares loop.

After checking the coherence and editing the data if necessary, we feed the output of MIGDAT2 into the next module TDEPMIG that effects the time-to-depth migration of the interface by the methods of Sections 2.2 and 2.3. Currently we have implemented only the special algorithm for a first homogeneous layer and the hybrid fast shooting-two-pointing for multi-layered piecewise homogeneous media. For the general, inhomogeneous layer case we favor at

this time, a full boundary value formulation as mentioned in Section 2.3. TDEPMIG produces for each ray successfully computed the interface contact point and normal vector. This data is then passed along to LSQVPR, which performs the least-squares fit of the interface for a pre-chosen parameterization.

Because of the shooting component, TDEPMIG does not produce the necessary amplitude information. Thus, we save the rays computed by TDEPMIG, and perform a full two-point ray tracing with amplitudes included in VELDAT, using the TDEPMIG rays as initial trajectories. Observe that, contrary to what one may expect, these initial rays are not exact, since the least-squares calculation made by LSQVPR would have modified the interface I_D. Of course, if everything is reasonably consistent, they will usually be fairly good first approximations, and the two-point iteration will converge very fast. In any case, this calculation is somewhat redundant, and it will be eliminated once TDEPMIG is replaced by a full two-point formulation.

VELDAT produces the information described in Section 2.6, and finally VELLSQ performs the least-squares inversion to obtain the next layer material properties. In what follows we show commented numerical and graphical output of all these stages for the model problem:

Model: Interfaces $z = I_k(x, y; \boldsymbol{a}, \boldsymbol{\alpha})$ where

$$I_k \equiv a_1^k + a_2^k * x + a_3^k * y + a_4^k * \exp\left[- \frac{\left(x - \alpha_1^k\right)^2 + \left(y - \alpha_2^k\right)^2}{\left(\alpha_3^k\right)^2} \right], \tag{2.10}$$

i.e. two-dimensional Gaussian surfaces with a planar background. The parameters of the target model are given in Table 1 (I_1 is the free surface).

In Fig. 7 we show a typical menu for starting AUTO3DRT. Besides the reflector and velocity parameters (not in the same order as in Table 1), we can modify interactively the number of receivers in each direction (NRAYX, NRAYY), the spacing (XSTEP, YSTEP), the position of the first receiver (XSTA, YSTA, ZSTA) in the receiver net, the position of the source, and we can also input the characteristics of the control screens (see Section 1.3).

For this example we have a small array of 8×8 receivers, spaced 0.5 units in each direction, with first receiver at (1, 1, 0). The source is located on the free surface at (4, 4, 0). Item '(5)' in the menu refers to the number of interfaces (plus one) at which the ray reflects, and '(10)' to which are these interfaces; the last number in '(10)' is the region in which the source is located. Finally, in '(7)' we can choose between a pure shooting or a two-point option. In this case we are making an exploratory shooting in order to locate the target and select an appropriate control screen for two-pointing. Only one control screen is used. Its characteristics are shown in Fig. 9, where the numbers displayed correspond to subregions of the receiver plane, with a code shown in Fig. 8.

Table 1
Exact model

k	a_1	a_2	a_3	a_4	α_1	α_2	α_3	v_p	v_s	ρ
2	5.0	0.0	0.0	0.4	3.0	3.0	1.0	4.0	3.0	2.0
3	10.0	0.1	0.1	0.3	3.5	3.7	2.5	6.5	2.89	2.3
4	–	–	–	–	–	–	–	8.0	6.0	1.0

```
1 1 0
 CURRENT VALUES OF PARAMETERS: :
1
 (1) REFLECTOR PARAMETERS
  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
  5.0000   0.4000   3.0000   3.0000   1.0000   0.0000   0.0000   0.0000   0.0000   0.0000
 10.0000  -0.2000   4.0000   5.0000   2.0000   0.0000   0.0000   0.0000   0.2000   0.2000

 (2) VELOCITY PARAMETERS
  4.0000   0.0000   0.0000   0.0000   0.0000   3.0000   2.0000   0.0000   0.0000   0.0000
  6.5000   0.0000   0.0000   0.0000   0.0000   2.8900   2.3000   0.0000   0.0000   0.0000


... MORE? (0/1)
1
 (3)NRAYX,NRAYY  8  8    (4)LAYERS  2
 (5)REFLECTIONS  2
 (6) X,Y,Z SOURCE      0.40E 01     0.40E 01     0.00E 00
 (7) XSTEP     0.50E 00 YSTEP     0.50E 00     SHOOT  1
 (8) SCREENS       1  Z-SCREEN         1.0000
 (9) DETAILED PRINTING    -0.10E 01
 (10) JCRO    2    1    1
 (11) XSTA,YSTA,ZSTA        0.10E 01     0.10E 01     0.00E 00
 (12) WINDOW  -0.1000E 02 -0.1000E 02  0.0000E 00
               0.1000E 02  0.1000E 02  0.2000E 02
 (13) MAXIMUM SCREEN WINDOW
    3.2000     3.2000     4.2000     4.2000
**** ENTER HOW MANY ITEMS YOU WANT TO CHANGE ****
0
```

Fig. 7. Menu for parameter choice in AUTO3DRT.

We recall that in shooting mode, rays will be generated starting at the source, they will go through the pixels of the control screen, reflecting from the desired interface, and finally returning to the free surface. The output in Fig. 9 represents the mapping just described, between the controll screen (in this case contained on the plane $z = 1$) and the free surface. For a ray landing outside of the receiver rectangle, we assign to the pixel that defines it the region identifier, according to the diagram in Fig. 8. For rays landing within $\frac{1}{2}$ spacing distance of a receiver we assign the number of the receiver. Receivers are numbered consecutively by rows (y constant), starting from the top left corner of the array. Finally, we use -9 to mark the boundary of the screen.

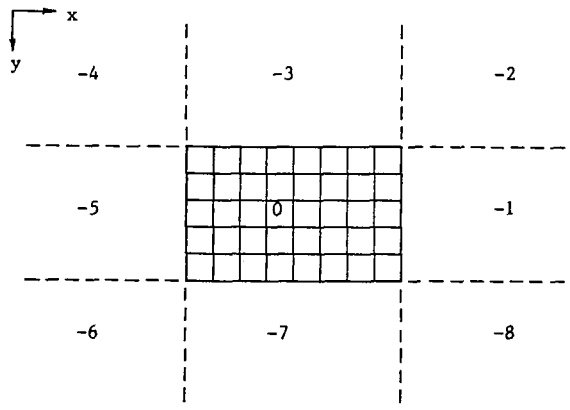

Fig. 8. Code for regions in receiver plane.

```
SOURCE        4.0000        4.0000        0.0000
  400 PIXELS SHOT
RECEIVER NET     1.0000     1.0000     4.5000     4.5000     8   8

SCREEN CORNERS ---  XMIN=    3.2000  YMIN=    3.2000  XMAX=    4.2000  YMAX=    4.20
INCREMENTS --- DELX=    0.0400  DELY=    0.0400
NX,NY  26  26 COARSE FACTOR   0
  0-----------------
 -9 -9 -9 -9 -9 -9 -9 -9 -9 -9  -9 -9 -9 -9 -9 -9 -9 -9 -9 -9  -9 -9 -9 -9 -9 -9
 -9 -4 -4 -4 -4 -4 -4 -4 -4 -4  -4 -4 -3 -3 -3 -3 -3 -3 -3 -3  -3 -3 -2 -2 -2 -9
 -9 -4 -4 -4 -4 -4 -4 -4 -4 -4  -4 -4 -3 -3 -3 -3 -3 -3 -3 -3  -3 -3 -2 -2 -2 -9
 -9 -4 -4 -4 -4 -4 -4 -4 -4 -4  -4 -4 -3 -3 -3 -3 -3 -3 -3 -3  -3 -3 -2 -2 -2 -9
 -9 -4 -4 -4 -4 -4 -4 -4 -4 -4  -4 -4 -3 -3 -3 -3 -3 -3 -3 -3  -3 -3 -2 -2 -2 -9
 -9 -4 -4 -4 -4 -4 -4 -4 -4 -4  -4 -4 -3 -3 -3 -3 -3 -3 -3 -3  -3 -3 -2 -2 -2 -9
 -9 -4 -4 -4 -4 -4 -4 -4 -4 -4  -4 -4 -3 -3 -3 -3 -3 -3 -3 -3  -3 -3 -3 -2 -2 -9
 -9 -4 -4 -4 -4 -4 -4 -4 -4 -4  -4 -4 -3 -3 -3 -3 -3 -3 -3 -3  -3 -3 -3 -2 -2 -9
 -9 -4 -4 -4 -4 -4 -4 -4 -4 -4  -4 -3 -3 -3  3  3 -3 -3 -3 -3  -3 -3 -3 -2 -2 -9
 -9 -4 -4 -4 -4 -4 -4 -4 -4 -4  -3  2  2 11 11 19 11 11  4  5  -3 -3 -3 -2 -2 -9
 11-----------------
 -9 -4 -4 -4 -4 -4 -4 -4 -4 -4   1 10 19 27 27 35 27 27 19 12   5 -3 -3 -2 -2 -9
 -9 -4 -4 -4 -4 -4 -4 -4 -4  9  10 19 28 36 44 43 42 34 26 19  12  5  7 -3 -2 -9
 -9 -4 -4 -4 -4 -4 -4 -5 -5  9  19 28 37 45 53 51 50 49 33 25  19 13 14  8 -2 -9
 -9 -5 -5 -5 -5 -5 -5 -5 -5 18  20 29 38 46 53 60 49 -5 -5 -5  18 20 14 16 -1 -9
 -9 -5 -5 -5 -5 -5 -5 -5 17 18  20 30 39 39 46 44 41 -5 -5 -5  17 19 22 16 -1 -9
 -9 -5 -5 -5 -5 -5 -5 -5 17 19  21 22 23 32 30 28 25 -5 -5 -5  -5 19 21 23 -1 -9
 -9 -5 -5 -5 -5 -5 -5 -5 -5 18  20 14 15  7  6  4  1 -4 -4 -5   9 19 22 32 -1 -9
 -9 -5 -5 -5 -5 -5 -5 -5 -5 18  20 13  7 -3 -3 -3 -3 -4 -4 -4   9 20 30 32 -1 -9
 -9 -5 -5 -5 -5 -5 -5 -5 -5 25  19 12  5 -3 -3 -3 -3 -4 -4 -5  18 29 30 40 -1 -9
 -9 -5 -5 -5 -5 -5 -5 -5 -5 33  26 19  4 -3 -3 -3 -3 -3 -3 10  28 37 39 48 -1 -9
 -9 -5 -5 -5 -5 -5 -5 -5 -5 -5  33 26 19 11  3 -3  2  2 11 28  37 46 47 -1 -1 -9
 22-----------------
 -9 -5 -5 -5 -5 -5 -5 -5 -5 -5  -5 33 34 27 19 19 19 27 36 37  46 55 56 -1 -1 -9
 -9 -6 -6 -6 -6 -6 -6 -6 -6 -5  -5 49 42 42 43 35 43 44 44 53  54 63 64 -8 -8 -9
 -9 -6 -6 -6 -6 -6 -6 -6 -6 -6  -6 -6 57 58 58 51 60 60 61 62  -7 -7 -7 -8 -8 -9
 -9 -6 -6 -6 -6 -6 -6 -6 -6 -6  -6 -6 -7 -7 -7 -7 -7 -7 -7 -7  -7 -7 -7 -8 -8 -9
 -9 -9 -9 -9 -9 -9 -9 -9 -9 -9  -9 -9 -9 -9 -9 -9 -9 -9 -9 -9  -9 -9 -9 -9 -9 -9
 ENTER 1 TO CONTINUE
```

Fig. 9. Control screen-to-receiver mapping. Shooting.

Thus, the presence of positive numbers indicate rays that have landed near a receiver, while the lack of regularity indicates wave front foldings. Clusters of repeated receivers indicate too much resolution in the screen for the given receiver spacing. In the case of multiple screens (see Section 1.3) we will have one of these diagrams for each sub-screen, which provides a zooming capability for regions of high complexity. Repeated receiver numbers in disconnected components indicate multiple arrivals.

All this information can then be used to tune in the control screen position and resolution. For instance, from Fig. 9 we infer that we can trim the current screen top and left side, and we can also increase its resolution, since there is no substantial clustering of repeated receivers. Thus for subsequent two-pointing, we use a 33×33 pixels screen with top left corner at (3.5, 3.5, 1), bottom right-hand corner at (4.14, 4.14, 1), and with spacings equal to 0.02 in both directions. We also see clear indication of at least three distinct families of arrivals.

On output we get in Fig. 10 a diagram similar to Fig. 9, where '0' indicates a hit in the receiver region that did not get converted into a convergent two-point ray. Two-point rays need not pass through our preset control screen pixels, so we assign the receiver ID to the pixel nearest to the actual intersection of the ray with the control screen. Since more than one ray may pass close to a given pixel, this information may get overwritten, and only the last arrival will be displayed. This gives then fast visual information on where the different arrivals came from. A similar

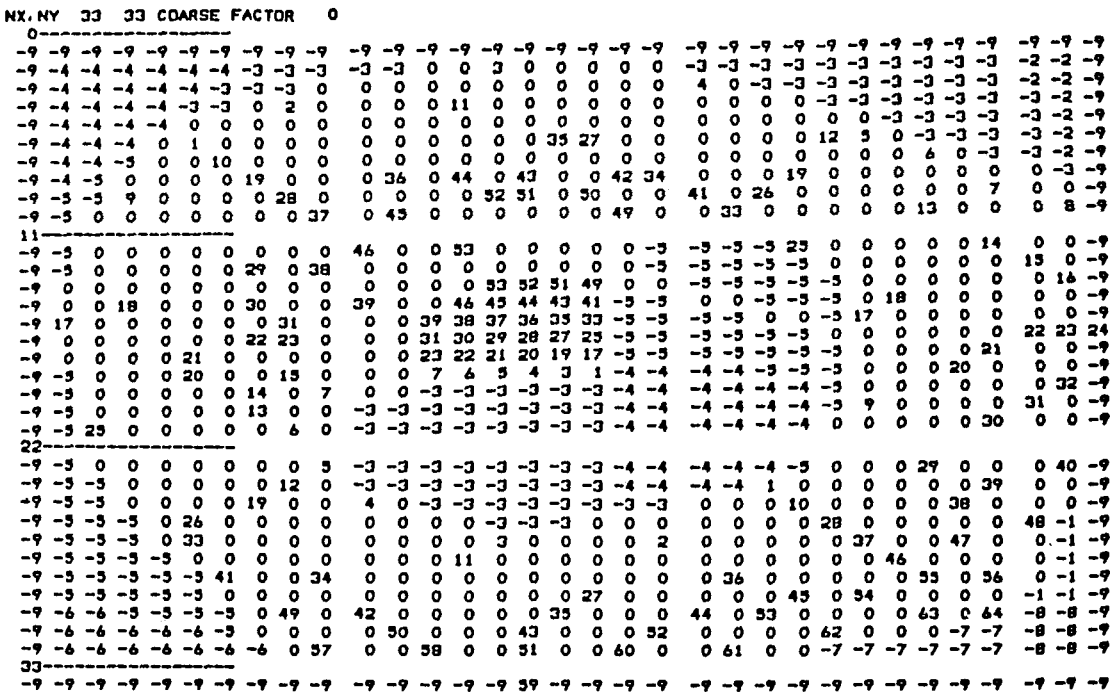

Fig. 10. Control screen-to-receiver mapping. Two-pointing.

diagram could be used to show the contacts of the different arrivals with the reflecting interface. Also one could refine the information about the shot arrivals outside the receiver rectangle RR, by including an indication of the distance d to RR. Thus, instead of displaying only the region number, we could add to it $[-10 * k]$, where k is the integer part of d/lev., lev. > 0. This would then show 'level lines' or contours of the distance function. We have found these diagrams extremely useful to help us in visualizing these complicated three-dimensional mappings.

Once we are satisfied that all possible shot-receiver arrivals have been obtained, we use MIGDAT2 to classify them in coherent time surfaces. The input to MIGDAT2 is shown in Fig. 11, where the two first columns indicate the receiver row (y constant) and column (x constant), while the following groups of two columns each describe the different rays in terms of their arrival times and peak amplitudes.

We see that in this case we have triple arrivals at most of the stations, with the exception of a fringe around the right and bottom edges, where only one arrival is present. Also, we observe five different arrivals for receiver (3, 3), which is located at $x = 2.0$, $y = 2.0$. This point happens to be symmetrically opposite to the source with respect to the center of the Gaussian depression in the interface.

The output of MIGDAT2 after the classification is shown in the form of synthetic seismogram sections in Fig. 12 (calculated with SYNT2D), on which we have overlayed the one-dimensional spline interpolants corresponding to the different time picks, as explained in Section 2.2. This gives us again a fast check of the correctness of the selection procedure, since the splines should interpolate the different arrival times.

1	1	2. 7595 0. 0759	2. 8946 0. 0794	2. 9243 0. 0269		
1	2	2. 7385 0. 0817	2. 8489 0. 0858	2. 8864 0. 0265		
1	3	2. 7184 0. 0859	2. 8214 0. 0783	2. 8595 0. 0267		
1	4	2. 6952 0. 0850	2. 8136 0. 0704	2. 8445 0. 0277		
1	5	2. 6707 0. 0811	2. 8220 0. 0643	2. 8418 0. 0297		
1	6	2. 6499 0. 0774	2. 8427 0. 0612	2. 8517 0. 0341		
1	7	2. 6357 0. 0753	2. 8732 0. 0698	2. 8744 0. 0489		
1	8	2. 6295 0. 0750				
2	1	2. 7385 0. 0817	2. 8489 0. 0839	2. 8864 0. 0265		
2	2	2. 7232 0. 0948	2. 7943 0. 1257	2. 8473 0. 0260		
2	3	2. 7083 0. 1067	2. 7619 0. 0960	2. 8195 0. 0262		
2	4	2. 6814 0. 0973	2. 7583 0. 0811	2. 8039 0. 0270		
2	5	2 6498 0. 0880	2. 7719 0. 0707	2. 8009 0. 0287		
2	6	2. 6244 0. 0810	2. 7969 0. 0638	2. 8110 0. 0322		
2	7	2. 6073 0. 0771	2. 8309 0. 0649	2. 8341 0. 0417		
2	8	2. 5992 0. 0759				
3	1	2. 7184 0. 0859	2. 8214 0. 0783	2. 8595 0. 0267		
3	2	2. 7083 0. 1067	2. 7619 0. 0960	2. 8195 0. 0262		
3	3	2. 7064 0. 2113	2. 7064 0. 2115	2. 7109 0. 1072	2. 7109 0. 1072	2. 7911 0. 0262
3	4	2. 6637 0. 1119	2. 7185 0. 0992	2. 7750 0. 0270		
3	5	2. 6275 0. 0929	2. 7375 0. 0780	2. 7719 0. 0286		
3	6	2. 6001 0. 0832	2. 7654 0. 0667	2. 7820 0. 0319		
3	7	2. 5819 0. 0784	2. 8015 0. 0652	2. 8057 0. 0406		
3	8	2. 5732 0. 0767				
4	1	2. 6952 0. 0850	2. 8136 0. 0704	2. 8445 0. 0277		
4	2	2. 6814 0. 0973	2. 7583 0. 0811	2. 8039 0. 0270		
4	3	2. 6637 0. 1119	2. 7185 0. 0992	2. 7750 0. 0270		
4	4	2. 6328 0. 1053	2. 7094 0. 1082	2. 7586 0. 0277		
4	5	2. 6014 0. 0912	2. 7243 0. 0800	2. 7555 0. 0294		
4	6	2. 5761 0. 0829	2. 7510 0. 0677	2. 7658 0. 0330		
4	7	2. 5590 0. 0788	2. 7866 0. 0675	2. 7898 0. 0432		
4	8	2. 5509 0. 0774				
5	1	2. 6707 0. 0811	2. 8220 0. 0643	2. 8418 0. 0297		
5	2	2. 6498 0. 0880	2. 7719 0. 0707	2. 8009 0. 0287		
5	3	2. 6275 0. 0929	2. 7375 0. 0780	2. 7719 0. 0286		
5	4	2. 6014 0. 0912	2. 7243 0. 0800	2. 7555 0. 0294		
5	5	2. 5755 0. 0857	2. 7317 0. 0725	2. 7523 0. 0315		
5	6	2. 5539 0. 0811	2. 7539 0. 0666	2. 7629 0. 0363		
5	7	2. 5393 0. 0785	2. 7866 0. 0787	2. 7874 0. 0558		
5	8	2. 5328 0. 0780				
6	1	2. 6499 0. 0774	2. 8427 0. 0612	2. 8517 0. 0341		
6	2	2. 6244 0. 0810	2. 7969 0. 0638	2. 8110 0. 0322		
6	3	2. 6001 0. 0832	2. 7654 0. 0667	2. 7821 0. 0319		
6	4	2. 5761 0. 0829	2. 7510 0. 0677	2. 7658 0. 0330		
6	5	2. 5539 0. 0811	2. 7539 0. 0666	2. 7629 0. 0363		
6	6	2. 5359 0. 0792	2. 7713 0. 0703	2. 7737 0. 0462		
6	7	2. 5241 0. 0783				
6	8	2. 5197 0. 0789				
7	1	2. 6357 0. 0753	2. 8732 0. 0698	2. 8744 0. 0489		
7	2	2. 6074 0. 0771	2. 8309 0. 0649	2. 8341 0. 0417		
7	3	2. 5819 0. 0784	2. 8015 0. 0652	2. 8057 0. 0406		
7	4	2. 5590 0. 0788	2. 7866 0. 0675	2. 7898 0. 0432		
7	5	2. 5393 0. 0785	2. 7866 0. 0787	2. 7874 0. 0558		
7	6	2. 5241 0. 0783				
7	7	2. 5150 0. 0788				
7	8	2. 5127 0. 0803				
8	1	2. 6295 0. 0750				
8	2	2. 5993 0. 0759				
8	3	2. 5732 0. 0767				
8	4	2. 5508 0. 0774				
8	5	2. 5328 0. 0780				
8	6	2. 5198 0. 0789				
8	7	2. 5127 0. 0803				
8	8	2. 5123 0. 0826				

Fig. 11. Time and amplitude data.

The synthetic seismogram $S_R(t)$ for each receiver R is calculated by superposing wavelets [4, Section 2.9] centered at each of the NARR arrival times T_i, scaled by the observed peak amplitudes A_i $(i = 1, \ldots, \text{NARR})$, and phase shifted by θ_i (this angle is obtained in the reflection-transmission coefficient calculation, and it may also contain phase shifts due to

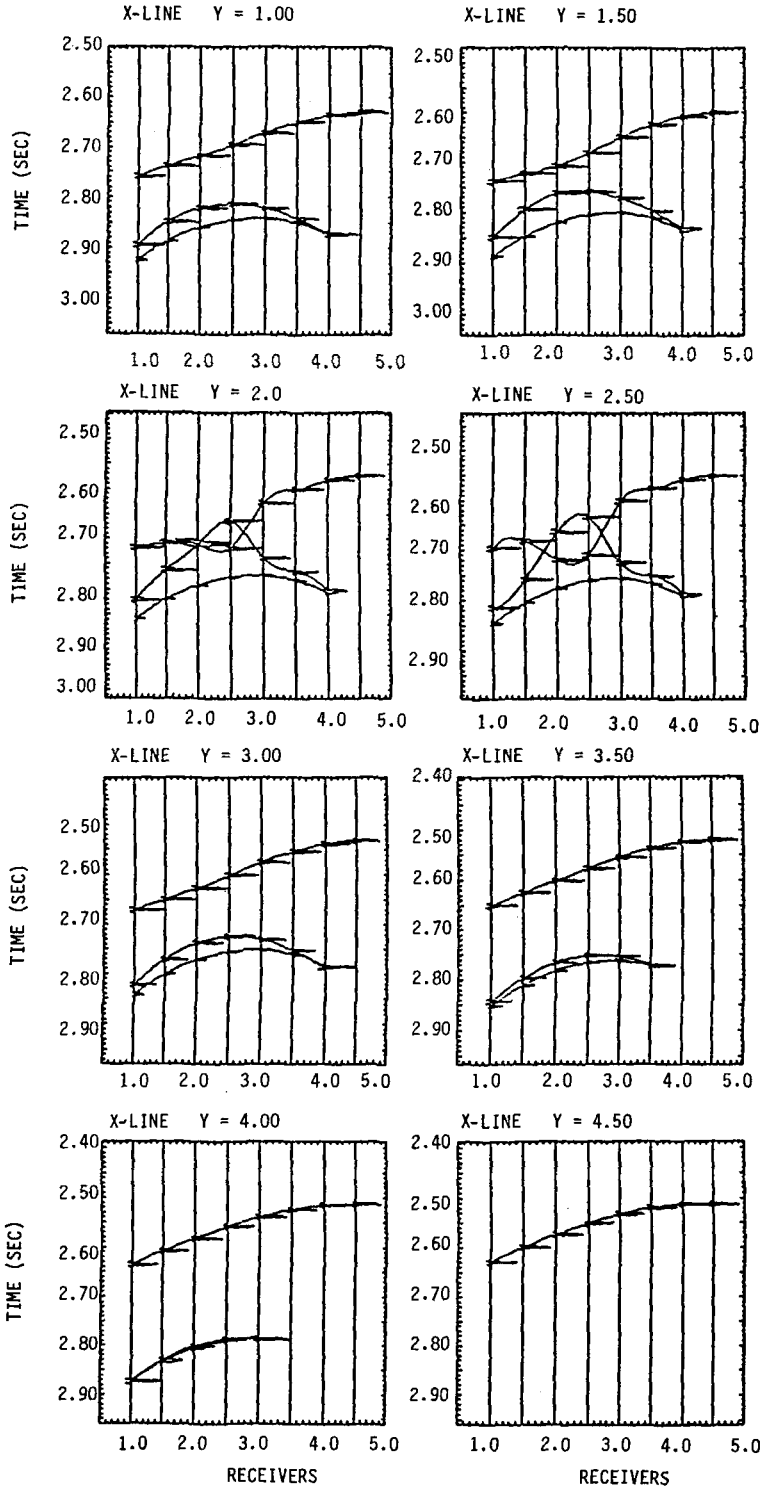

Fig. 12. Two-dimensional slices (x-direction) from three-dimensional time-amplitude data, with splines superposed.

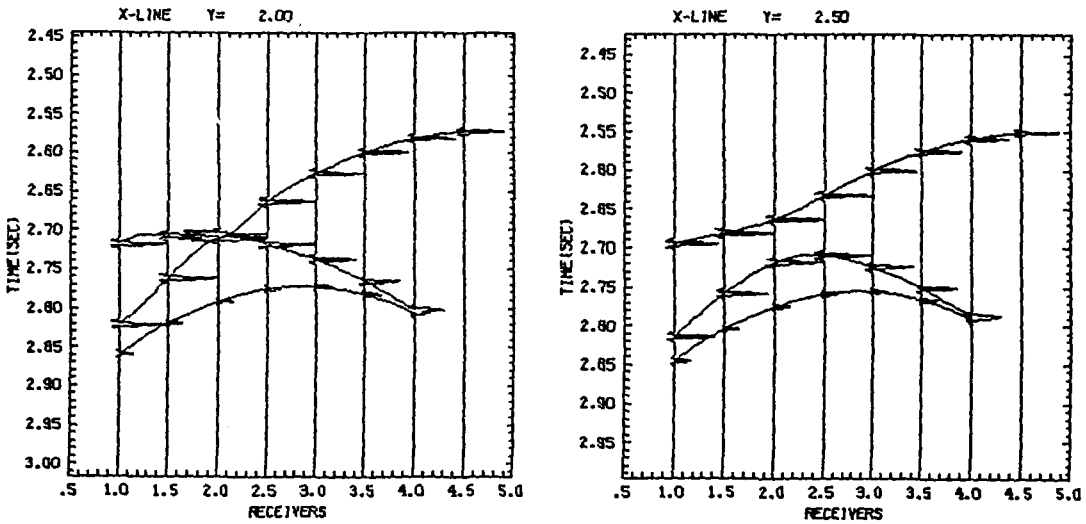

Fig. 13. Slices $y = 2, 2.5$ after editing the arrival surfaces.

focusing and passage through caustics):

$$S_R(t) = \sum_{i=1}^{\mathrm{NARR}} A_i * \exp\left[-\pi * f * (t - T_i)^2\right] * \cos\left[2\pi f * (t - T_i) - \theta_i\right],$$

where f is the frequency. In this calculation we have used a fairly sharp wavelet with $f = 100$ Hz. The advantage of using these type of wavelets is that no Hilbert transform is necessary in order to include phase shifts into the synthetic seismograms.

Figure 12 shows the different slices along constant y-lines. We can also calculate the slices along the normal direction (x constant), but they are entirely similar. We see that the picking has succeeded, with the exception of anomalies in the frames corresponding to $y = 2.0, 2.5$. Not unexpectedly, the trouble seems to be located in a neighborhood of $x = 2.0$, $y = 2.0$.

A new pass through MIGDAT2 allows us to edit the file, interchanging the three offending points in the two first arrivals, i.e. the points at (2.5, 2.0), (2.0, 2.5) and (2.5, 2.5). Recomputing the splines we obtain the corrected frames for $y = 2.0, 2.5$ as shown in Fig. 13. Line $y = 2.0$ still shows a strong anomaly at $x = 2.5$, suggesting that a higher local resolution may be indicated there. Although it is hard to see it in the picture for line $y = 2.0$, the two upper curves are tangential rather than intersecting, which is coherent with its neighboring y-slices.

2.7.1. Least-squares inversion results

For the next stage we input to TDEPMIG the time-amplitude output of MIGDAT2 in order to obtain points and tangent planes on the unknown interface I_2. This information is in turn fed into LSQVPR to calculate the least-squares fit with a surface of the form (2.10). In this case we have 4 linear and 3 nonlinear parameters.

In Table 2 we give the initial, final and exact values of the parameters, the residual sum of squares, number of iterations in VARPRO and number of observations for four different runs, namely,

R_1 data as picked originally with MIGDAT2 (Fig. 12);
stopped, because of too many iterations,

Table 2
Comparison of different outlier removal strategies; column r shows initial and final residual sum of squares

	a_1	a_2	a_3	a_4	α_1	α_2	α_3	r	Iterations	Observations
Initial	–	–	–	–	3.6	2.4	1.2	–	0	–
								5.93		
R_1	−246.5	1.01	0.99	250	−1.41	−1.32	47	2.4	50	462
								2.3		
R_2	5.0008	−0.0002	−0.0002	0.394	3.006	3.006	1.014	0.48	7	360
								2.69		
R_3	4.92	0.007	0.007	0.414	2.98	2.98	1.099	0.88	8	462
								2.56		
R_4	4.989	0.0005	0.0004	0.406	3.0007	3.0007	1.023	0.41	7	432
Exact	5.0	0.0	0.0	0.4	3.0	3.0	1.0	–	0	–

R_2 data edited (Fig. 13),
R_3 same as R_1, but outliers artificially removed,
R_4 same as R_2 with outliers removed.

To remove outliers we use the exact interface to test in TDEPMIG the difference between z_{exact} and $z_{\text{calculated}}$. The observation is removed if that difference is larger than 0.01 in relative value. This is of course an artificial procedure, but it shows the importance of weeding out erroneous data.

Obviously, solution R_1 is unacceptable, while solutions R_2 to R_4 are fairly close, since the editing of the data is in itself a way to eliminate outliers.

In what follows, we use the solution from R_2, which does not involve any artificial outlier removal. With VELDAT we produce the necessary amplitude data, which is then fed into VELLSQ to obtain the least-squares estimates of the material properties for the second layer.

As we show in Table 3, the results for the individual velocities v_p, v_s, and density ρ are not very good. However, as is well known [1,5], the impedances $\rho * v_p$, $\rho * v_s$ can be better determined. We include in Table 3 also the results of the optimization using the same procedure, but keeping the density fixed at its true value (as if it were known by independent means), which gives much better results.

In Fig. 14 we show side by side the two-point rays traced with the exact interface by AUTO3DRT and those traced by VELDAT with the least-squares recovered interface. In Fig. 16 we show the contour plot of the vertical deviation between the exact and the least-squares interfaces, which gives a measure of the fit quality. This contour plot is based on the values of the deviation calculated at the points of intersection of the rays with the least-squares interface,

Table 3
Inversion of material properties using amplitude data

	v_p	v_s	ρ	$\rho * v_p$	$\rho * v_s$
Initial	7.0	2.0	2.3	16.1	4.6
Variable	6.98	3.31	2.12	14.8	7.02
Fixed	6.41	2.89	2.3	14.74	6.81
Exact	6.5	2.89	2.3	14.95	6.65

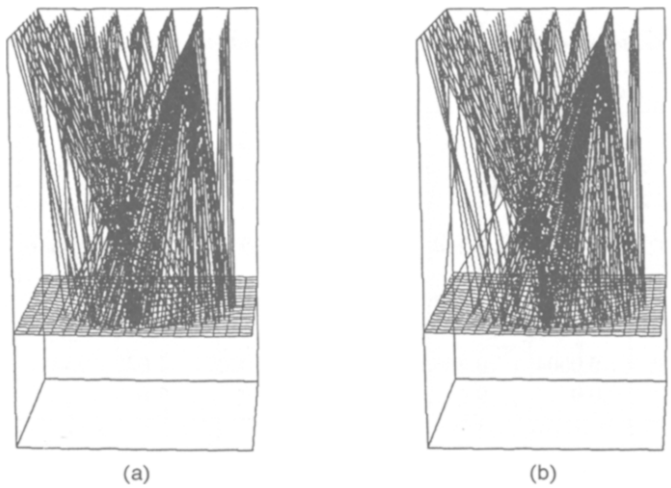

Fig. 14. (a) Two-point rays (exact); (b) Two-point rays after recovery of interface.

whose (x, y) coordinates are shown in Fig. 15 as the vertices of the triangulation. Their distribution of course, is far from uniform, so some of the features seen in the contour plot may be artifacts. In any case, the maximum vertical deviation of 8.5 ms on a total depth of over 5 km is quite acceptable.

2.7.2. Second layer

With the structure partially recovered, we proceed to work on a deeper layer. The whole process is repeated, using the known values for the parameters of interface I_2, the material properties of the first two layers, and time-amplitude data corresponding to primary reflections from interface I_3 (see Table 1). We keep for simplicity the same receiver and source configuration, although that is not required.

Because of the interaction of the two curved layers we obtain now only one full coherent arrival to all stations, and a few additional arrivals at the lower right-hand corner of the receiver array (see Fig. 17), for a total of 80 rays. Suppressing observations with relative vertical error greater than 0.01, we get for the parameters of I_3 the results shown in Table 4. We see a marked

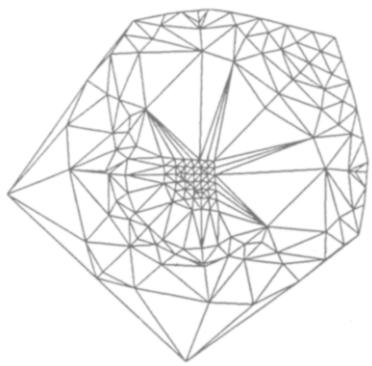

Fig. 15. Triangulation of (x, y) coordinates of recovered interface reflection.

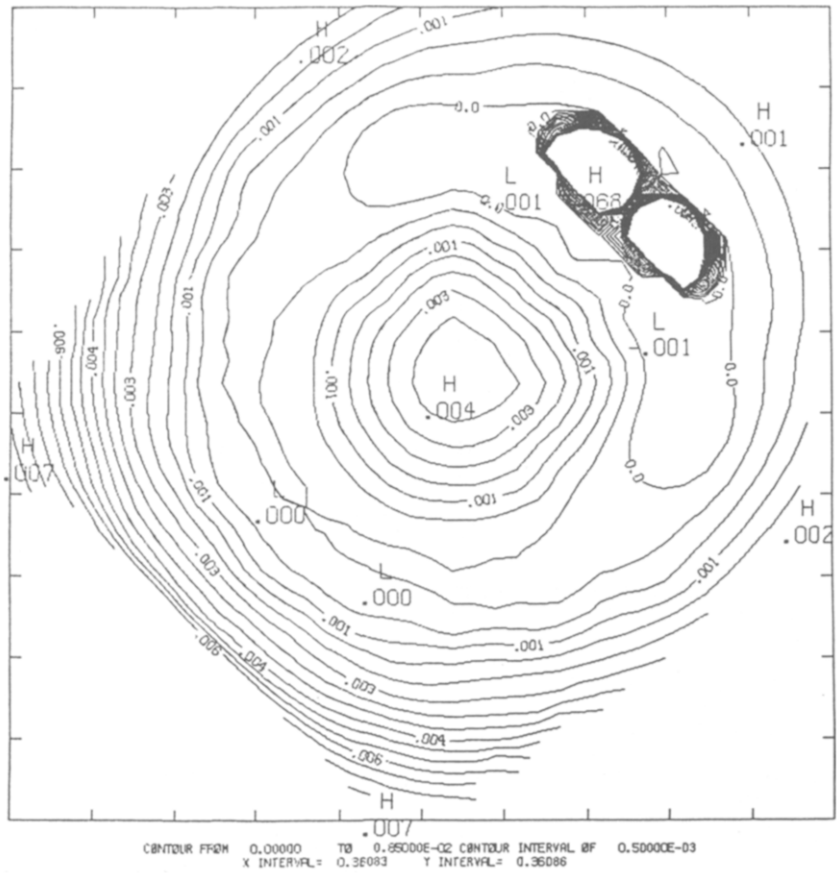

Fig. 16. Contours of vertical error between exact and recovered reflection points.

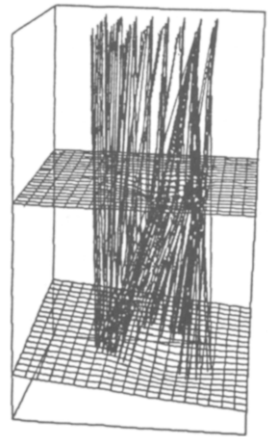

Fig. 17. Exact two-point rays. Primary reflections on interface *I*.

Table 4
Least-squares recovery of interface I_3

	a_1	a_2	a_3	a_4	α_1	α_2	α_3	r	Iterations	Observations
Initial values	–	–	–	–	4.2	2.96	1.8	0.6	–	–
Recovery	10.005	0.084	0.088	0.33	3.7	3.81	1.71	0.37	8	210
Exact	10.0	0.1	0.1	0.30	3.5	3.7	1.5	–	–	–
Relative	0.0005	0.16	0.12	0.05	0.06	0.03	0.14	–	–	–

deterioration with respect to the previous layer, as it was to be expected, but we are still getting fairly reasonable accuracy on some of the parameters, considering the small data set being used. These results indicate that increasing the resolution and data redundancy, by considering more shots and a more dense receiver array, will produce very adequate results with a moderate amount of work.

3. Dynamic velocity and interface determination

3.1. Introduction

In this section we consider the problem of dynamically inverting travel time data with ray tracing in three-dimensional media. The term 'dynamic' indicates that now the ray tracing will be contained in a nonlinear least-squares loop, in contrast to the 'static' approach taken in Section 2. Potentially this will be a more expensive procedure, but also considerably more powerful and accurate than the ones described in Section 2, which still can be very useful as initializers.

We include some preliminary numerical results demonstrating the feasibility of the approach and the need for more powerful computing tools than the small minicomputer previously used. Consequently, we include also some results of runs on the CRAY XMP/48 at the San Diego Supercomputer Center, where the whole system was ported without difficulties.

This work extends to three dimensions previous two-dimensional results [27,30]. Besides overcoming the added complexity of the increased dimensionality, we have considerably improved the earlier techniques, thanks to the more sophisticated automatic ray tracer described in Section 1 and the least-squares algorithms employed.

Velocity and interface information is recovered from travel time data, separatedly or jointly, using two different nonlinear least-squares techniques. Comparative runs were made on our PRIME 2250 minicomputer and on one processor of the CRAY XMP/48, showing speedups of over a 100 times (for a code that has not been optimized for vectorization). This speedup brings the typical run time to a level that makes the development feasible, and indicates that interactive three-dimensional modeling by these techniques is possible within current technology.

3.2. Travel time inversion

Given a set of observed travel times (T_j^o), corresponding to rays joining a source and an array of receivers with a given signature, we are interested in determining a parameterized model

described by velocities v_k and interfaces I_k:

$$v_k(\eta;\ \alpha^k), \qquad z - I_k(x,\ y;\ \beta^k) = 0,$$

such that travel times (T_j^c) calculated by three-dimensional ray tracing best fit the observed data in the least-squares sense.

As it has been established in [27,30], this problem can be stated and solved in the generality of a heterogeneous multilayered model with multiple sources and ray signatures. However, for the sake of intelligibility we will describe first how to invert for one layer velocity, then how to invert for an interface, and at the end it will be clear how to assemble these pieces together to obtain a general procedure.

3.2.1. Recovery of a layer velocity

For $\eta(x,\ y,\ z)$, $u(\eta) = 1/v(\eta)$, $w = u(\eta)\ \mathrm{d}\eta/\mathrm{d}s$, where s is arc length along the ray, the three-dimensional ray equations that we solve for each layer are

$$\frac{\mathrm{d}\eta}{\mathrm{d}s} = v(\eta)w(s), \quad \frac{\mathrm{d}w}{\mathrm{d}s} = \nabla u(\eta), \quad \frac{\mathrm{d}T}{\mathrm{d}s} = u(\eta), \qquad s \in [0,\ S], \tag{3.1}$$

where S is the unknown total arc length. We have added an equation for the partial travel time:

$$T(s) = \int_0^s u(\eta(s))\ \mathrm{d}s,$$

since $T(S)$ is the quantity of interest.

Clearly, if $v = v(\eta;\ \alpha)$, then $T(S) = T(S;\ \alpha)$, and we can obtain $\partial T/\partial\alpha_k$ by differentiating equations (3.1). Of course, the situation in general will be somewhat more involved since there is a different v and α for each layer, as well as boundary and interface conditions that we have yet to mention. For the case of rays starting from the free surface, reflecting on a known interface I_2 and arriving at an array of receivers on the free surface, we observe that the only condition that involves v is the so-called 'arc length condition':

$$v^2(\eta(0)) * \| w(0) \|_2^2 - 1 = 0. \tag{3.2}$$

Differentiation of (3.1) and (3.2) with respect to the parameters α_k produces the linearized or variational equations

$$\frac{\mathrm{d}}{\mathrm{d}s}\begin{bmatrix} \partial\eta/\partial\alpha_k \\ \partial w/\partial\alpha_k \\ \partial T/\partial\alpha_k \end{bmatrix} = J(\eta(s),\ s)\begin{bmatrix} \partial\eta/\partial\alpha_k \\ \partial w/\partial\alpha_k \\ \partial T/\partial\alpha_k \end{bmatrix} + g_k(s) \tag{3.3}$$

where

$$g_k(s) = \begin{bmatrix} w\partial v/\partial\alpha_k \\ \nabla_\eta(\partial/\partial\alpha_k) \\ \partial u/\partial\alpha_k \end{bmatrix}, \tag{3.4}$$

and where $J(\eta(s),\ s)$ is the Jacobian matrix of the right-hand side of (3.1). If $L(\eta)$ represents the linearized version of (3.2), then we also have as a side condition

$$L(\eta(0)) + 2 * v(\eta(0)) * v_{\alpha_k}(\eta(0)) * \| w(0) \|_2^2 = 0. \tag{3.5}$$

The other linearized boundary and interface conditions are homogeneous, since there is no explicit v dependence.

Given the exact model $v(\eta;\ \alpha^*)$, (3.1) is solved for each arrival to obtain the synthetic data $T_j^o(\alpha^*) \equiv T(S;\ \alpha^*)$. Starting from an initial set of parameters α, we use a nonlinear least-squares procedure to find the α^*, that solves

$$\min_{\alpha} \sum_j \left(T_j^c(\alpha) - T_j^o\right)^2.$$

If we choose a method that does not use derivatives, then (3.1) is all that is required. However, finite differences or other similar ways of producing derivative approximations are expensive in terms of function evaluations. Considering that a 'function evaluation' here means tracing a complete set of rays, we prefer to turn to methods that make use of derivatives of T with respect to α. We can obtain those derivatives by solving (3.3) as many times as there are parameters. This seems in itself quite costly and complicated. However, and this is one of the main bonuses of the two-point approach and method of solution, upon convergence to the solution of (3.1), a discrete approximation to the linearized equations is available:

$$\Omega \delta\eta = \boldsymbol{g},$$

where the sparse matrix Ω is already in LU form. Therefore, solving the required systems is actually quite inexpensive, since it involves essentially back substitution with a sparse LU matrix.

3.2.2. Recovery of an interface

We consider now the same problem and data as above, but with v known, and attempt to determine the parameterized interface $z - I(x,\ y;\ \beta) = 0$. There are now two places where the interface appears explicitly.

(a) *Snell's law.*

$$\boldsymbol{w}^t = \boldsymbol{w}^i - \left(\pm\sqrt{u_t^2 - u_i^2 + \langle \boldsymbol{w}^i,\ \boldsymbol{\nu}\rangle^2} - \langle \boldsymbol{w}^i,\ \boldsymbol{\nu}\rangle\right)\boldsymbol{\nu}, \tag{3.6}$$

where i and t in $\boldsymbol{w}$ and u symbolize incident and either transmitted or reflected values, while the sign in front of the square root should be + for transmission and − for reflection. The vector $\boldsymbol{\nu}$ is the unit normal to the interface at the point of contact:

$$\boldsymbol{\nu} = \begin{bmatrix} -I_x \\ -I_y \\ 1 \end{bmatrix} \Big/ \sqrt{I_x^2 + I_y^2 + 1}\,. \tag{3.7}$$

In the reflection case $u_t = u_i$ and (3.6) simplifies to

$$\boldsymbol{w}^t - \boldsymbol{w}^i + 2\langle \boldsymbol{w}^i,\ \boldsymbol{\nu}\rangle\boldsymbol{\nu} = 0. \tag{3.6r}$$

(b) *Contact condition.*

$$z - I(x,\ y;\ \beta) = 0. \tag{3.8}$$

Thus, to calculate $\partial T/\partial\beta_k$, $k = 1,\ldots, q$, using the same technique as before requires differentiating (3.1) and all the boundary and interface conditions with respect to β_k. On one side we

obtain again the linearized equations, and on the other side we get nonzero forcing terms whenever there is an explicit occurrence of β_k: namely in (3.6) or (3.6r) and (3.8). From (3.7)

$$\frac{\partial \boldsymbol{v}}{\partial \beta_k} = \begin{bmatrix} -I_{x\beta_k} \\ -I_{y\beta_k} \\ 0 \end{bmatrix} \Big/ \sqrt{I_x^2 + I_y^2 + 1} - \begin{bmatrix} -I_x \\ -I_y \\ 1 \end{bmatrix} \frac{I_{x\beta_k} I_x + I_{y\beta_k} I_y}{\left(I_x^2 + I_y^2 + 1\right)^{3/2}}, \tag{3.9}$$

and therefore, differentiation of (3.6) yields the linearized equation:

$$\begin{aligned} w_{\beta_k}^{\mathrm{t}} - w_{\beta_k}^{\mathrm{i}} - \Big(\pm R^{-1} \big[u_{\mathrm{t}} \langle \nabla u_{\mathrm{t}}, \eta_{\beta_k} \rangle - u_{\mathrm{i}} \langle \nabla u_{\mathrm{i}}, \eta_{\beta_k} \rangle + \langle w_{\beta_k}^{\mathrm{i}}, \boldsymbol{v} \rangle + \langle \boldsymbol{w}^{\mathrm{i}}, \boldsymbol{v}_{\beta_k} \rangle \big] \\ - \langle w_{\beta_k}^{\mathrm{i}}, \boldsymbol{v} \rangle - \langle \boldsymbol{w}^{\mathrm{i}}, \boldsymbol{v}_{\beta_k} \rangle \Big) \boldsymbol{v} - \left(\pm R - \langle \boldsymbol{w}^{\mathrm{i}}, \boldsymbol{v} \rangle \right) \boldsymbol{v}_{\beta_k} = 0, \end{aligned} \tag{3.10}$$

where $R = \sqrt{u_{\mathrm{t}}^2 - u_{\mathrm{i}}^2 + \langle \boldsymbol{w}^{\mathrm{i}}, \boldsymbol{v} \rangle^2}$.

Observe that all the terms containing w_{β_k} or η_{β_k} have already been taken into account in the variational equations, so the actual new forcing term is simply

$$g_{1(10)} = \left(\pm \langle \boldsymbol{w}^{\mathrm{i}}, \boldsymbol{v}_{\beta_k} \rangle / R - \langle \boldsymbol{w}^{\mathrm{i}}, \boldsymbol{v}_{\beta_k} \rangle \right) \boldsymbol{v} + \left(\pm R - \langle \boldsymbol{w}^{\mathrm{i}}, \boldsymbol{v} \rangle \right) \boldsymbol{v}_{\beta_k}. \tag{3.11}$$

For (3.6r) this simplifies even further to

$$g_{1(10r)} = -2 \langle \boldsymbol{w}^{\mathrm{i}}, \boldsymbol{v}_{\beta_k} \rangle \mu - 2 \langle \boldsymbol{w}^{\mathrm{i}}, \boldsymbol{v} \rangle \boldsymbol{v}_{\beta_k}. \tag{3.11r}$$

For (3.8) we get as the forcing term simply I_{β_k}.

3.3. Two-point ray-tracing / least-squares procedure

For each successfully traced ray, derivatives are computed with respect to the parameters to be recovered by the procedure described in Section 3.2. That is, we compute the forcing terms, place them in the appropiate spots as right-hand sides of the linearized, discrete boundary value problem solver, make a subroutine call (to SYSNEW), and obtain η_p, w_p, T_p, where p symbolizes the particular parameter being considered. From this, we pick $T_{jp}^{\mathrm{c}} \equiv T_p(S)$, and place it in the jth row, pth column of the Jacobian matrix of T with respect to the parameters $\boldsymbol{p}$. Doing this for each ray and parameter in turn yields the least-squares Jacobian matrix

$$\boldsymbol{T}_p = \left(\partial T_j(\boldsymbol{p}) / \partial p_k \right), \quad j = 1, \ldots, \mathrm{Nobs}, \; k = 1, \ldots, \mathrm{Nparams}$$

used by a nonlinear least-squares solver to update the current estimates of the parameters $\boldsymbol{p}$.

This is schematically the process to solve iteratively

$$\min_{\boldsymbol{p}} \Phi(\boldsymbol{p})$$

where $\Phi(\boldsymbol{p}) = \| T^{\mathrm{o}} - T^{\mathrm{c}}(\boldsymbol{p}) \|_2^2$ and T^{o}, T^{c} are the vectors of observed and calculated travel times, respectively. Any of a number of available codes, some of which have already been mentioned in Section 2, can be used for this purpose.

Observe that, in principle, several strategies are possible to determine a parametric model of a three-dimensional region which has been appropriately sampled by a set of rays. Considering always the case of complete, single valued, smooth curved layers, let us assume that we have

identified P-arrivals corresponding to direct reflections from interfaces $I_2(\beta^2),\ldots,I_f(\beta^f)$, with associated P-velocities $v_1(\alpha^1),\ldots,v_{f-1}(\alpha^{f-1})$. Let us also assume that we have initial guesses for all the parameters α^{k0}, β^{k0}, obtained for instance by the procedures of Section 2. Then we can proceed in various ways:

(a) *Global inversion*. The most expensive mode of operation is to try to invert for all the layers at once. This requires tracing rays for the whole set of observations, at each iteration of a potentially large, nonlinear least-squares problem. We think this option should be used only as the final step of a more progressive approach.

(b) *Stagewise inversion*. This is similar to the approach of Section 2, except that now we consider dynamic inversion, and obtain simultaneously an interface and the corresponding layer velocity directly from the arrival time data. This is of course a smaller problem than (a), both in the number of parameters and in the number of rays to be traced, since only those corresponding to reflections in the current deepest interface need be considered. To avoid the accumulation of errors observed in the corresponding static inversion algorithm, a combined strategy may be advisable.

Steps (b) and (a) can be alternated, where (a) is used as a global procedure down to the current deepest interface. Thus, we would start with a step of type (b) to determine the best parameters for the first layer velocity v_1, and interface I_2. Then, since for the first layer (a) and (b) coincide, we would continue to use (b) to obtain v_2, I_3, and then we would start to alternate the use of (a) (to correct v_1, v_2, I_2, I_3) and (b), as we continue down to deeper layers and interfaces.

For procedure (a), the ray traces corresponding to families reflecting from different interfaces are totally independent, and can therefore be effected in parallel, if appropriate hardware is available. Thus, given enough processors, the time necessary to compute the reflections from all layers would be essentially that corresponding to the deepest one. This is parallelism at the module level, which is easily implementable in current supercomputer multiprocessor architectures.

A similar procedure has just been developed independently in two dimensions by a research group at Amoco Production Company [37].

3.4. Implementation and numerical results

We have implemented procedure (b), for the dynamical recovery of a variable velocity layer in two stages. The first stage assumes that the interface is known. As before the synthetic exact data is generated with program AUTO3DRT from Section 1. In this case we only use arrival times, and since there is no need for derivatives of the travel time surface, coherence is not a major issue. However, we still must make sure of a correct matching between 'observed' and calculated travel times. Although we have not used it in this preliminary implementation, a coherence pass with program MIGDAT2 could be useful to insure that a correct matching is effected, especially in the case of multiple valued time surfaces.

In this test run we use the rays out of AUTO3DRT to start the ray tracing in the nonlinear least-squares iteration. Of course, the initial values of the parameters are different from the true ones, but we expect that the deformation in the ray paths are not sufficient to create any anomalies. This is another reason to combine these procedures with the static ones described in Section 2, since with real data we could not employ the start-up procedure just indicated, but we could use instead the rays generated by TDEPMIG or VELDAT (see Section 2.7). The rays of

iteration i are used to start the ones for iteration $(i+1)$; thus, if the variations in the velocity parameters are small, this ray tracing should be quite fast as compared to the one in AUTO3DRT that may involve an extensive additional search. This is born out by the computer times obtained.

We see in the test example below that, because of the variable layer velocity we cannot use the fast shooting initialization in AUTO3DRT. A shooting initialization is still used, but it requires RAYT3D-PASVA4, and therefore is as costly as a two-point solve. This shows in the CPU time necessary on a PRIME 2250 mini-computer, even for a simple one-layer problem, and it strongly suggests more powerful hardware. We have successfully transported AUTO3DRT and all its necessary subroutines to the CRAY XMP/48 at the San Diego Supercomputer Center. No part of this code was written with vectorization in mind, so its performance is far from optimal; however, the speedup is sufficient to make practical our current computations. With some modifications to improve vectorization and with the use of multiprocessors we believe that these codes can successfully attack realistic complex three-dimensional problems.

Example 3.1. Let us consider a geological model consisting of one layer over a half space, with a curved interface of the form considered in Section 2.7. The velocity $v_1(\eta)$ is now variable and given by

$$v_1(\eta) = 4.0 + 0.1 * (x + y + z)$$

while the interface I_2 is defined by

$$z = 5 + 0.4 * \exp\left(-\left[(x-3)^2 + (y-3)^2\right]\right).$$

The data consists of arrival times for rays starting at So = (4, 4, 0), traveling downwards through layer 1, reflecting on interface I_2, and arriving at a 6×6 array of receivers located on the free surface at

$$x = 1.0 + 0.5 * (j-1), \qquad y = 1.0 + 0.5 * (i-1).$$

Using a 33×33 pixel control screen with XMIN = YMIN = 3.5, XMAX = YMAX = 4.14, DELX = DELY = 0.02, AUTO3DRT obtains one full set of arrivals and 23 additional multiples, for a total of 59 observations. Each curved ray is composed of 16 legs, thus the nonlinear systems solved contain 114 equations and unknowns. Other statistics of interest are

- constant velocity rays traced (used to initialize the variable velocity shooter): 407,
- successfully shot variable velocity rays: 527,
- failed shot rays (mostly because of shooting off the window): 1054,
- failed 2P rays: 30,
- successful 2P rays: 59.

The CPU time on the PRIME 2250 was 1 hr. 50 min. (6600 s), while the same run on the CRAY XMP/48 took 52 s, for a speedup of 127. Since we have observed in highly vectorized finite element codes speedups of up to a factor of 1000, we still have plenty of room for improvement.

For the least-squares iteration we use again VARPRO, but in its nonseparable, Gauss–Newton mode. Program VELVPR reads initial ray and observed travel time data from a file supplied by AUTO3DRT, together with initial values α^0 for the parameters in v:

$$v = \alpha_1 + \alpha_2 x + \alpha_3 y + \alpha_4 z.$$

Table 5
Dynamic recovery of layer velocity

Iteration	α_1	α_2	α_3	α_4	Residual	Time (s)
0	4.8	0.12	0.12	0.12	2.51	–
6 PRIME	4.054	0.1013	0.1013	0.1013	0.20	2800
6 CRAY	4.054	0.1014	0.1014	0.1015	0.22	30
Exact	4.0	0.1	0.1	0.1	–	–

Two-point rays are traced by RAYT3D, and the corresponding travel times are calculated. For each ray, partial derivatives of the travel time with respect to the parameters are computed by successive calls to subroutine SYSNEW with appropriate right-hand sides. All this information is provided to VARPRO which effects the correction of the model parameters, and then the process is iterated. In each iteration, the rays of the previous iteration are used to start the two-point ray tracing with the updated model parameters. The fact that there is no shooting search performed, and that for small changes in the model parameters the initial rays will be fairly accurate, makes these iterations considerably less expensive than the original ray traces performed by AUTO3DRT. In this particular problem, even counting the additional work used by VARPRO, each iteration takes 1/10 of the time employed by AUTO3DRT.

We give in Table 5 results corresponding to runs for this problem on the PRIME 2250 and the CRAY XMP/48.

In this run we observe a speedup of 93 between the PRIME 2250 and the CRAY XMP/48, and an excellent accuracy in the recovered parameters.

Example 3.2. In attempting to solve the complete layer problem, i.e. the simultaneous recovery of the velocity v_1 and the interface I_2, we run into difficulties with VARPRO. There are three different problems that we have identified and dealt with:

(a) When computing the rays in an iteration there is the possibility (and actual occurrence) of failure. For a failed ray the corresponding observation should be deleted.

(b) A more subtle error occurs when in the course of an iteration a ray bifurcates to a different arrival (for the same station). In this case, attempts to match up the calculated time to the observed one will be futile. This is an example of incoherence: the computed ray is in a different travel time sheet than the observed one. Again a quick fix here is to delete the corresponding observation when the situation is detected.

In order to identify this last kind of anomaly, an outlier detection strategy can be employed. We have chosen a simple one that consists of computing a running average of the residuals, and deleting an observation if its residual is several times larger than the current average. More elaborated techniques, like those used in robust estimation [8], will be tested in the future.

Unfortunately, most current software for solving nonlinear least-squares problems does not have a provision for varying the number of observations during the iteration. In particular, VARPRO uses information from the previous iteration in an intricate way, which makes it quite difficult to allow a change in the number of observations from iteration to iteration. Of course,

one can exit and restart every time there is such a change, with the hope that asymptotically the process would stabilize to a fixed set of observations. Usually a costly overhead penalty must be paid by such restarts.

(c) The third type of difficulty we have encountered is one that is well known in least-squares inversion of geophysical data: ill-conditioning.

In order to study and deal with these three sources of problems in a more controlled fashion, we have implemented a simple nonlinear least-squares Marquardt type iteration, based on the Singular Value Decomposition, following the excellent ideas of Jupp and Vozoff [20]. See also Gill et al. [14] as a general reference, and Lines and Treitel [25]. Within this code we can easily control any side effects produced by the elimination of observations, so that takes care of (a) and (b) above. In order to explain how the ill-conditioning is dealt with, we describe the formulation of a typical iteration of the Marquardt-like procedure.

Let T_p be the $m \times n$ Jacobian matrix of the least-squares residual vector, and let

$$T_p = USV^{\mathrm{T}} = \sum_{i=1}^{n} s_i \boldsymbol{u}_i v_i^{\mathrm{T}},$$

be its Singular Value Decomposition, where the singular values s_i are assumed ordered:

$$s_1 \geqslant s_2 \geqslant \cdots \geqslant s_n \geqslant 0.$$

Let $k_i = s_i/s_1$ be the normalized singular values. Ill-conditioning shows up as small k_i, and a crucial problem in least-squares inversion is the determination of the numerical rank of the linearized problem. Setting to zero singular values below a preassigned threshold is one possibility, which leads ultimately to iterates confined to a subspace of the parameter space, and therefore relinquishes the estimation of part of the model. With real data this may be appropriate, since it can be interpreted as an indication that some of the parameters cannot be resolved with the available data, and the known characteristics of the observational errors can then be used to establish a meaningful cutoff level.

A more conservative approach, akin to Marquardt's modification of the Gauss–Newton algorithm, consists of introducing a diagonal damping matrix $D = \mathrm{diag}(d_i)$, in the pseudoinverse of T_p:

$$B^{+} = VDS^{+}U^{\mathrm{T}}.$$

This produces the damped correction

$$\delta \boldsymbol{p} = B^{+}\boldsymbol{\epsilon} = \sum_{i=1}^{r} d_i \rho_i v_i / s_i,$$

where $\boldsymbol{\epsilon}$ is the residual vector $T_p^{\mathrm{T}}\Phi(\boldsymbol{p})$, $\rho = U^{\mathrm{T}}\boldsymbol{\epsilon}$, and r is the rank of T_p, i.e. $s_i = 0$, $i > r$.

For the classical Marquardt algorithm, a relative threshold $\mu > 0$ (with respect to s_1) is chosen, and

$$t_i = s_i^2/\left[s_i^2 + s_1^2\mu^2\right].$$

Jupp and Vozoff [20] have also used successfully other mollifiers which provide sharper cutoffs.

In our implementation we have chosen an initial value μ_0 for the threshold; this value is diminished whenever a sufficient decrease in the root mean squared relative residual is not achieved. If RESOLD and RESNEW are two successive such residuals, $\mu \leftarrow 0.1 * \mu$ if RESLEV

Table 6
Dynamic recovery of a variable velocity and a curved interface for various initial parameters

α% Off	Iterations	Res	$\|\text{Error}\|_\infty$	Res_f	Time (s)	μ_0	RESLEV
+5	7	0.007	0.003	1.9×10^{-6}	112	0.001	0.15
+10	10	0.013	0.004	1.9×10^{-6}	154	0.001	0.3
+20	14	0.029	0.006	1.9×10^{-6}	185	0.015	0.6

$\leqslant$ RESNEW/RESOLD < 1. If RESNEW/RESOLD $\geqslant 1$, the step in the direction $\delta \boldsymbol{p}$ is reduced to enforce descent, except in the case when a change in μ was effected in the previous iteration.

The parameters μ_0 and RESLEV are quite important in obtaining convergence for this ill-conditioned problem, where the algorithm has a tendency to jam at extraneous local minima. Larger μ_0 and RESLEV correspond to a more conservative approach, while too small values may lead to premature divergence. This simple strategy is enough to overcome the ill-conditioning of this nine-parameters problem, corresponding to the determination of the velocity v_1 and the interface I_2, even when we start from parameters which are up to +20% away from the true parameters, as we show in Table 6. These results were obtained on the CRAY XMP/48.

We have taken as the model for the interface

$$z = \alpha_5 + \alpha_5 \exp\bigl(-[x-\alpha_7]^2 - [y-\alpha_8]^2\bigr)/\alpha_9^2.$$

The synthetic data generation for this model, using AUTO3DRT on the CRAY computer took 62 s for

704 constant velocity shot rays,
660 successful variable velocity shot rays,
1320 failed shot rays,
130 successful two-point rays,
31 failed two-point rays.

The condition of the least-squares Jacobian at the solution was $s_1/s_9 = 13{,}846$. The total number of two-point rays computed for the 5% off case during the optimization was 1081, not including 12 failures. Thus we see that, fortunately, the ray tracing within the modeling is substantially faster than in the generation phase.

4. Conclusions

In this work we have made a preliminary scan through some of the most important aspects of the inverse modeling of complex three-dimensional geological regions by means of seismic ray-tracing and nonlinear least-squares techniques.

The first step was to show that a viable forward modeling capability was available. One of the main themes in our approach has been the use of boundary value techniques, and we showed in Section 1 how a general, automatic ray tracing code could be devised, and how its implementation performed in some test problems. The techniques described there were extensions of what we learned earlier in developing two-dimensional counterparts, but they also required the solution of some new problems that are particular to three dimensions. For instance, the display

of the information in an appropriate way is quite crucial for practical applications. Our experience in the last year with the code AUTO3DRT indicates that we have achieved a robust, versatile implementation, capable of tracing source-receiver rays with minimal user interaction in fairly complex piecewise continuous inhomogeneous media with curved interfaces. Some of the features of this code that we like to emphasize are:

- automatic initialization and restarting upon failure,
- efficient receiver continuation in two-point mode,
- complete elastic amplitudes, including geometrical spreading,
- automatic calculation of multipath arrivals, such as those generated by a syncline.

These features are absolutely necessary, even at this research level, in order to concentrate in all the other problems involved in the inversion, without being diverted by difficulties in the ray tracing.

In Section 2 we considered the problem of time-to-depth migration of pre-stacked reflection data. We implemented algorithms for step-by-step migration of successive travel time horizons, emphasizing the two-point and boundary value approach. Both interface and material properties were recovered with the algorithms described, which we entitled 'static inversion', to indicate that the ray tracing was outside the least-squares loop.

This type of migration becomes more inaccurate as one proceeds to deeper layers, but it provides an economical way of generating an initial model which can then be refined with more comprehensive techniques. This is the subject of Section 3, where we consider 'dynamic inversion', since the ray tracing is an integral part of the least-squares iteration. These techniques are considerably more expensive than the static ones, especially in the case of variable velocity (inhomogeneous layers), where the ray tracing becomes relatively time consuming. Thus, an algorithm combining in an alternate fashion the static and dynamic approach suggests itself naturally.

At this point in our study it was natural to ask how these codes would fare on a supercomputer environment. Speedups of two orders of magnitude were achieved with respect to the minicomputer in which the early development was done. We foresee that with vectorization and multitasking one additional order of magnitude in speed could be obtained on the same computer.

We have strived to obtain an orderly growth in this complex prototype system, emphasizing modularity and transportability. This has paid in facilitating the development, debugging and maintenance of the code, and in allowing painless and fast migration to the CRAY. We have also foreseen in our design the possible use of different advanced computer architectures. We think that the stage is set for using these modules in more challenging and complex problems, both with synthetic and real data, extending these preliminary results and schemes as it becomes necessary.

Acknowledgment

This work has been facilitated by using electronic mail for rapid communication with colleagues and most important, for receiving valuable public domain software through the automatic system described in [7]. We thank Professor Gene H. Golub of Stanford University for his effort in making these facilities available to the Numerical Analysis community, and also for discussions about some of the least-squares issues.

References

[1] K. Aki and P.G. Richards, *Quantitative Seismology 2* (Freeman, San Francisco, 1980).

[2] R.H. Bartels, J.C. Beatty and B.A. Barsky, An introduction to the use of splines in computer graphics, University of Waterloo, Canada, Tech. Rept. CS-83-09, 1985.

[3] V. Cerveny, Ray synthetic seismograms for complex two-dimensional and three-dimensional structures, *J. Geophys. 58* (1985) 2–26.

[4] V. Cerveny, I.A. Molotkov and I. Psencik, Ray method in seismology, University Karlova, Praha, 1977.

[5] J.F. Claerbout, *Imaging the Earth Interior* (Blackwell, Palo Alto, 1985).

[6] D. Coleman, P. Holland, N. Kaden, V. Klema and S.C. Peters, A system of subroutines for iteratively reweighted least squares computations, *ACM Trans. Math. Software 6* (1980) 327–336.

[7] W.M. Coughram Jr., E. Grosse and D.J. Rose, Variation diminishing splines in simulation, *SIAM J. Sci. Statist. Comput. 7* (1986) 696–705.

[8] J. Dennis, Nonlinear least squares, in: D. Jacobs, ed., *The State of the Art in Numerical Analysis* (Academic Press, New York, 1977) 269–312.

[9] J.E. Dennis, D.M. Gay and R.E. Welsch, An adaptive nonlinear least squares algorithm, *ACM Trans. Math. Software 9* (1983) 369–383.

[10] J.J. Dongarra and E. Grosse, Distribution of mathematical software via electronic mail, NA Manuscript 85-2, AT & T Bell Labs, 1985.

[11] G. Forsythe, M. Malcolm and C. Moler, *Computer Methods For Mathematical Computations* (Prentice-Hall, Englewood Cliffs, NJ, 1977).

[12] F.N. Fritsch and J. Butland, A method for constructing local monotone piecewise cubic interpolants, *SIAM J. Sci. Statist. Comput. 5* (1984) 300–304.

[13] B.S. Garbow, K.E. Hillstrom and J.J. More, MINPACK project, Argonne National Laboratories, 1980.

[14] P.E. Gill, W. Murray and M.H. Wright, *Practical Optimization* (Academic Press, New York, 1981).

[15] G.H. Golub and V. Pereyra, The differentiation of pseudoinverses and nonlinear least squares problems whose variables separate, *SIAM J. Numer. Anal. 10* (1973) 413–432.

[16] H. Gjoystdal and B. Ursin, Inversion of reflection time in 3-D, *Geophysics 46* (1981) 972–983.

[17] P. Hubral and T. Krey, Interval velocities from seismic reflection time measurements, SEG Monograph, 1981.

[18] J.M. Hyman and B. Larrouturou, The numerical differentiation of discrete functions using polynomial interpolation methods, in: J.F. Thompson, ed., *Numerical Grid Generation* (Elsevier, Amsterdam, 1982) 487–506.

[19] J.M. Hyman and M.J. Naughton, Static rezone methods for tensor-product grids, *Lectures in Applied Mathematics 22* (Amer. Math. Soc., Providence, RI, 1985) 321–343.

[20] D.L.B. Jupp and K. Vozoff, Stable iterative methods for the inversion of geophysical data, *Geophys. J. Roy. Astr. Soc. 42* (1975) 957–976.

[21] L. Kaufman, Variable projection methods for solving separable nonlinear least squares problems, *BIT 15* (1975) 49–57.

[22] L. Kaufman and V. Pereyra, A method for separable nonlinear least squares problems with separable nonlinear equality constraints, *SIAM J. Numer. Anal. 15* (1978) 12–20.

[23] H.B. Keller, Global homotopies and Newton methods, in: C. de Boor and G.H. Golub, eds., *Recent Advances in Numerical Analysis* (Academic Press, New York, 1978).

[24] M. Lentini and V. Pereyra, PASVA4: An ordinary boundary solver for problems with discontinuous interfaces and algebraic parameters, *Mat. Apl. Comput. 2* (1983) 103–118.

[25] L.R. Lines and S. Treitel, Inversion with a grain of salt, *Geophysics 50* (1985) 99–109.

[26] V. Pereyra, PASVA3: An adaptive finite difference FORTRAN program for first order nonlinear, ordinary boundary problems, *Lecture Notes in Computer Science 76* (Springer, Berlin, 1979) 67–88.

[27] V. Pereyra, Two-point ray tracing in heterogeneous media and the inversion of travel time data, in: R. Glowinski and J.L. Lions, eds., *Computational Methods in Applied Science and Engineering* (North-Holland, Amsterdam, 1980) 553–570.

[28] V. Pereyra, Deferred correction software and its application to seismic ray tracing, *Comp. Supp. 5* (1984) 211–226.

[29] V. Pereyra, Modeling with ray tracing in two-dimensional curved homogeneous layered media, in: A. Wouk, ed., *ARO Workshop on Microcomputers in Large Scale Scientific Computation* (SIAM, Philadelphia, 1986).

[30] V. Pereyra, H.B. Keller and W.H.K. Lee, Computational methods for inverse problems in geophysics: Inversion of travel time observations, *Physics of the Earth and Planetary Interiors 21* (1980) 120–125.
[31] V. Pereyra and G. Scherer, Efficient computer manipulation of tensor products with applications in multi-dimensional approximation, *Math. Comp. 27* (1973) 595–605.
[32] V. Pereyra and G. Sewell, Mesh selection for discrete solution of boundary value problems in ordinary differential equations, *Numer. Math. 23* (1975) 261–268.
[33] A. Ruhe and P.A. Wedin, Algorithms for separable nonlinear least squares problems, *SIAM Rev. 22* (1980) 318–337.
[34] R.D. Russell and J. Christiansen, Adaptive mesh selection strategies for solving boundary value problems, *SIAM J. Numer. Anal. 15* (1978) 59–80.
[35] B. Ursin, Time to depth migration using wave front curvature, *Geophys. Prosp. 30* (1982) 261–280.
[36] H.R. Nelson, *New Technologies in Exploration Geophysics* (Gulf Pub. Co., Houston, 1983).
[37] R.P. Bording, L.R. Lines, J.A. Scales and S. Treitel, Principles of seismic travel time tomography, *Geophys. J. Roy. Astr. Soc.*, to appear.
[38] V. Pereyra, Improved automatic two-point ray tracing in inhomogeneous three-dimensional media, Weidlinger Associates Inversion Project Rept. 87-01, 1987.