REFLECTION TRAVELTIME TOMOGRAPHY

AND THE

MAXIMUM ENTROPY PRINCIPLE

P. M. WHITING B.Sc.(Syd.), M.Sc.(Macq.)

A thesis submitted in fulfilment
of the requirements for the degree of
Doctor of Philosophy

School of Mathematics and Statistics
(Applied Mathematics)

University of Sydney

JULY, 1993

TABLE OF CONTENTS

LIST OF PUBLICATIONS

Whiting, P.M., 1991, Practical reflection tomography and maximum entropy imaging, 61st Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 986-989.

Whiting, P.M., 1991, Maximum entropy reflection tomography, Exploration Geophysics, **23**, 459-464.

Whiting, P.M., 1993, Entropy and tomographic image reconstruction, Submitted to Geophysics.

Whiting, P.M., 1993, Practical reflection tomography for velocity field estimation, Submitted to Geophysics.

Whiting, P.M., 1993, Entropy and geophysical inverse problems, Submitted to Exploration Geophysics.

SUMMARY

Conventional reflection tomography creates an estimate of subsurface seismic velocity structure by inverting a set of seismic traveltime data. This is achieved by solving a least-squares optimisation problem that finds the velocity and depth model that minimises the difference between raytraced and measured traveltimes. Obtaining the traveltime data can be difficult as manual picking of reflection times is required and all picked reflection events must be associated with the reflector depths defined in the model. Even with good traveltime data the optimisation problem is very non-linear and the surface restriction of the sources and receivers makes the problem generally underdetermined. These issues result in severe ambiguity and local minima problems.

This thesis shows that modifications to the conventional reflection tomography algorithm can make it a more practical and reliable procedure that is less likely to be trapped by local minima. The ray tracing procedure is changed so that reflector depths are not necessary and automatic traveltime interpretation can be successful. Entropy constraints are introduced (after being justified) which prevent unwarranted velocity structure from appearing. This feature adds significant stability and reduces the ambiguity problems. Staged smoothing of the optimisation function helps avoid local minima.

Synthetic data examples show that the algorithm can be very effective on noise free data. Adding noise to synthetic data reduces the algorithms effectiveness, but inversions of real data sets produces updated velocity fields that result in superior pre-stack depth migrations.

§1 INTRODUCTION

Reflection seismic surveys are conducted with the aim of imaging subsurface geologic structures. Historically, seismic profiles have been used to aid detection of hydrocarbon reservoirs. These are most commonly found in sedimentary basins. Seismic profiles show reflection strength as a function of subsurface locations and acoustic traveltime. Ultimately, the explorationist requires an estimate of the actual depth structure.

Accurate depth imaging of seismic reflection data requires knowledge of the subsurface acoustic velocity field. During conventional seismic data processing, this subsurface velocity field is derived from CDP stacking velocities (see Telford et al., 1976) which are based on the normal (or hyperbolic) moveout assumption (see Taner and Koeler, 1969). The hyperbolic moveout assumption is only valid for velocity fields that do not vary appreciably in the lateral direction (hyperbolic moveout is exact only for constant velocity fields), but it has proved to be very successful as the sedimentary basins under investigation generally have little lateral variation on the scale of the recording aperture. Whenever significant lateral variations are present (for example, faulting, reef structures, channelling and low or high velocity zones), conventional processing can produce less than optimum results. In this case processing must be performed that is not limited by the hyperbolic moveout assumption.

The technique known as reflection tomography was demonstrated by Bishop et al. (1985). The aim of this technique is to derive the subsurface velocity field (in terms of interval velocities and depths) directly from the traveltimes of reflected seismic waves. No moveout assumption is made. The algorithm requires an array of reflected energy

traveltimes to be picked from the collected data, along with a set of reflector depths. The algorithm then iteratively updates the subsurface model in an attempt to minimise a constraint statistic which is basically the sum of the differences between the raytraced (modelled) traveltimes and the measured traveltimes. Extra terms are often added to the constraint statistic in an attempt to stabilise the inversion. The constraint statistic does not have to be based on traveltime residuals - the only requirement is that it is a useful measure of errors in the model.

Reflection tomography has not yet formed part of conventional processing. There are many reasons for this. Firstly, the required reflector depths are difficult to estimate and poor estimates can prohibit the inversion process from achieving useful results. Finding reflecting horizons can be a difficult task in itself - especially with complex structural geology (see Bording et al., 1987; Jacobs et al., 1992; Lailly et al., 1992). After estimating a set of reflectors, the traveltime data then have to be picked for energy reflected from these reflectors only. This is because the raytracing used in the inverse procedure (two-point raytracing), finds the traveltime between a surface source, a specified reflector and a surface receiver. Such restrictions imposed on the traveltime picking can make this step difficult and laborious (see Leger et al., 1989; Lailly et al., 1992).

Even when a reliable set of traveltimes and reflectors have been defined, a useful inversion result is not guaranteed. The surface restriction of the sources and receivers result in an inverse problem that is predominantly underdetermined. The iterative minimisation of the traveltime error is prone to entrapment by local minima and many areas of ambiguity exist (see van Trier, 1988). A solution to these problems must be found before reflection tomography will find more widespread use. An easily obtained,

reliable velocity field obtained from reflection tomography could be used to produce accurate seismic images in areas of complex structural geology.

The problems with reflection tomography, outlined above, have ~~lead~~led researchers to develop many related algorithms to extract velocity field information from the traveltimes of reflected energy (Biondi, 1988; van der Made and van Riel, 1988; Harlan, 1989; Sherwood, 1989; Toldi, 1989; Biondi, 1990). These algorithms have varying degrees of success and ease of use. Most of them simplify the problem in some way (for example, by defining velocity fields characterised by smooth spline functions only) in order to stabilise the inversion. Also, significant research has been conducted into the use of depth migration to simplify the traveltime data interpretation (van Trier, 1990; Harlan et al., 1991a, 1991b; Jacobs et al., 1992). Bording et al., 1987, used depth migration to help ensure that the reflector depths were accurate, without providing any picking assistance.

The aim of this thesis is to overcome the problems of conventional reflection tomography without limiting its potential. An algorithm will be developed that is very simple to use and practical, yet has the potential for improving the image quality obtained from the conventional algorithm. The reflector restrictions imposed on traveltime interpretation will be removed. A staged smoothing procedure will stabilise the inversion, and entropy constraints will help solve ambiguity problems and add further stability.

Sword (1986) demonstrated the advantages of removing reflector depths from the model parameters of Bishop's reflection tomography algorithm. This requires a raytracing procedure that does not need defined reflector locations. When the apparent dip of a reflection is measured on common source and common receiver gathers, it is possible to estimate the intercept angles of the rays at the surface. With this information, raytracing can be performed by tracing from both the source and receiver locations into the model

until the two ends meet. Reflectors do not have to be defined but apparent dips must be measured along with the traveltime. The advantage of this method is that there are no restrictions on the reflections that can be picked. Automatic data picking routines are a practical approach in this case. Fault plane reflections, diffracted energy and multiple arrivals (see Lailly et al., 1992) all become available as valid data with this style of raytracing. These extra data categories can provide wider angular coverage and hence better solutions.

The use of smoothing in inverse problems is a well known method for stabilising an inversion (Biondi, 1988; Menke, 1989, Williamson, 1990). It makes intuitive sense to use heavy smoothing early in the inversion, but to decrease the smoothing as the inversion improves the model - ultimately resulting in an unsmoothed velocity image. This type of approach initially determines the required low frequency velocity changes and gradually extracts the higher and higher frequency components of the modifications to the starting model. Williamson (1990) applied this type of smoothing approach to reflection tomography. Biondi (1988) used a similar approach in his beam-stack inversion. The algorithm which employs stages of decreasing smoothing, is effectively a means of damping the components of the inversion with small singular values (Jupp and Vozoff, 1975). Some local minima (caused by the small singular value components) are temporarily smoothed over, allowing the inversion to avoid them. The probability of finding the global minimum of the constraint statistic is increased.

Entropy, which plays a central role in this thesis, has a long history. Entropy constraints are commonly used to aid in the inference of a solution for undetermined inverse problems. Originally derived in the fields of thermodynamics (Jaynes, 1988a) and information theory (Shannon, 1948; Khinchin, 1957), entropy has recently been applied

in many other disciplines of science; for example, image processing, astronomy and geophysics (Reitch, 1977; Skilling and Gull, 1985; Bassrei, 1990; Whiting, 1991b). Maximum entropy solutions to inverse problems have the special property that they do not contain any structure other than that needed to obtain a solution. The maximum entropy solution is unique and, to the extent that the data is reliable, the solution is maximally unbiased in terms of structure. Entropy is maximised at every iteration, ensuring that no unjustifiable decisions are made at any stage of the iterative inversion. Since spurious anomalies are prohibited, maximising entropy has a stabilising effect on the inversion. However, it can be difficult to associate the current reflection tomography problem with applications of the maximum entropy principle within other disciplines. But with the aid of the notion of contingency tables and simple tomographic problems, an instructive association can be made (see Chapter 3).

With entropy constraints, the inverse problem becomes the simultaneous minimisation of a constraint statistic (error function) and the maximisation of the entropy function. The Gauss-Newton method of inversion used by Bishop et al. (1985) is replaced by a method using local quadratic approximations to the constraint and entropy functions within a 4 or 6 dimensional subspace of the original model space (with typically thousands of dimensions; see Kennett and Williamson, 1988, for more on subspace methods). Such an inversion algorithm was described by Skilling and Bryan (1984). Suitable modifications are necessary to incorporate stages of decreasing smoothing. The inversion process can also continuously estimate the extent of validity of the local quadratic approximations, and adjust the iteration dependent distance limit accordingly. This ensures the most rapid inversion possible without producing instability. The constraint statistic is the sum of the squared errors which are weighted to approximate an L1 norm so that erroneous data do

not have a detrimental effect on the inversion.

Synthetic data examples show that significant velocity anomalies can be resolved with the modified reflection tomography algorithm. The maximum entropy images obtained clearly have less spurious structure - the background noise[1] is suppressed. The strategy of decreasing smoothing successfully avoids some local minima problems.

When real seismic data is used, the interpreted traveltime data will be inherently noisy. By adding noise to some synthetic data it was found that 10% noise (added to the measured angles) was tolerable whereas 20% noise may restrict any useful velocity updates from being achieved. Reflection tomographic inversions of two real data sets show that significant improvements can be achieved in real situations. The validity of these updates is demonstrated by improved pre-stack depth migration results.

The spatial frequency of velocity variations obtainable from conventional means is low compared to that possible with reflection tomography. The most obvious role for reflection tomography is to verify and extend the best velocity model obtained by conventional means. With the modified reflection tomography algorithm proposed in this thesis, such updated velocity models are more reliable and obtained with little manual intervention.

Other important works relevant to this thesis are (alphabetical order) Ables (1974), Al-Yahya (1989), Berkhout (1987), Bracewell (1978), Burg (1967), Carrion et al. (1993), Catlin (1989), Chiu and Stewart (1987), Cornwell (1983), Delprat and Lailly (1990), de Vries and Berkhout (1984), Dusaussoy and Abdou (1991), Etgen (1988), Farra and Madariaga (1988), Finn and Backus (1991), Frieden (1972), Gill and Murray (1974),

---

[1] Since the synthetic data is noise free, the noise in the solution stems from the underdetermined nature of the inverse problem.

Good (1963), Gull (1988), Gull and Daniell (1978), Gull and Skilling (1983), Jackson (1979), Jaynes (1957, 1982, 1984, 1985, 1988b), Justice et al. (1989), Kennett (1988), Kennett et al. (1988), Kennett and Harding (1985), LaBrecque (1990), Lailly et al. (1990), Landa et al. (1989), Langan et al. (1985), Lines (1991), Lynn and Claerbout (1982), Marquardt (1963), McGillivray and Oldenburg (1990), Michelena and Harris (1991), Moser (1991), Oldenburg and Ellis (1991), Phillips and Fehler (1991), Press et al. (1989), Reitsch (1977, 1985), Scheuer and Oldenburg (1988), Shaw and Orcutt (1985), Shore (1984), Singh and Singh (1991), Skilling (1988), Skilling and Gull (1984), Smith et al. (1984), Stewart (1989), Stork (1992), Stork and Clayton (1991), Sword (1987), Tarantola (1987), Tikhonov and Arsenin (1977), Ulrych et al. (1990), Vidale (1989), White (1991) and Zhou et al. (1992).

§2 REFLECTION TRAVELTIME TOMOGRAPHY - PRACTICALITIES

2.1 Ray tracing with triangular cells

The development of a traveltime tomography algorithm begins with consideration of ray tracing routines. Ray tracing can easily be the most time consuming component of the overall algorithm and generally a trade-off between accuracy and speed is considered. One purpose of the current study is to improve reflection tomography in general, and rather than create a possible source of error with inaccurate ray tracing, computational speed was sacrificed for the sake of accuracy.

Bishop et al. (1985) used a ray tracing procedure as described by Langan et al. (1985). This procedure divides the image into square cells and computes approximate linear velocity functions of the form $V = a + bx + cz$ within each cell (a,b,c constants). The four nodes (see Figure 2.1-1) at the corners of the square cell will have velocities that do not generally satisfy such a linear relationship. However, a simple velocity structure is desired for rapid computation. So a linear velocity field is

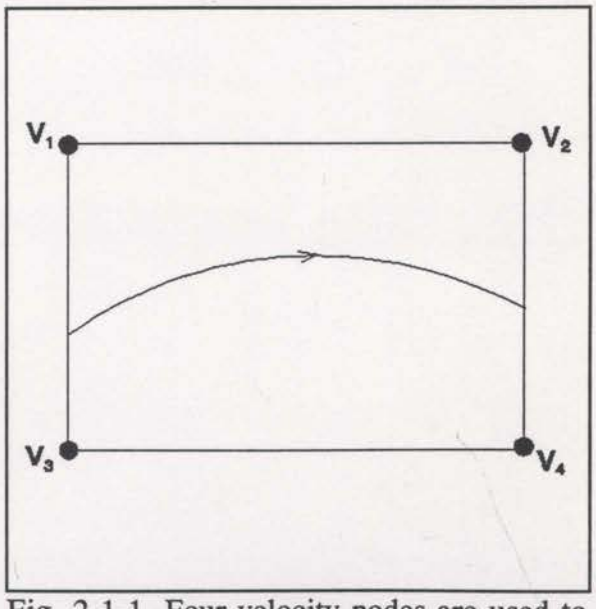


Fig. 2.1-1 Four velocity nodes are used to construct a square cell. A typical circular raypath is shown.

approximated to fit the four corner velocities, causing velocity discontinuities at the boundaries of the cells. Within the locally linear velocity fields, the raypaths are either circular or straight and the velocity discontinuities cause refraction at the boundaries. Langan et al. (1985) made further approximations to simplify the calculations of the circular arcs. The combination of the approximations used here may be insignificant, but for the present study any such possible source of error is to be avoided at the outset. More accurate ray tracing can be achieved with triangular cells.

The use of triangular instead of square cells has significant initial difficulties due to the existence of a sloping side (see Figure 2.1-2). When searching for raypath intersections with the sides of the cells, considerably more book keeping has to be done to cope with the sloping sides. However, there are significant advantages as well. Each triangular cell is defined by only three velocity nodes, uniquely specifying a linear velocity field. Therefore all velocities are continuous at cell boundaries, and each velocity node is honoured exactly. The resulting rays are circular within each cell, without refraction at the boundaries (see Figure 2.1-2). It is also intuitively easier to fit a triangular mesh to a surface than to try and fit a square mesh.

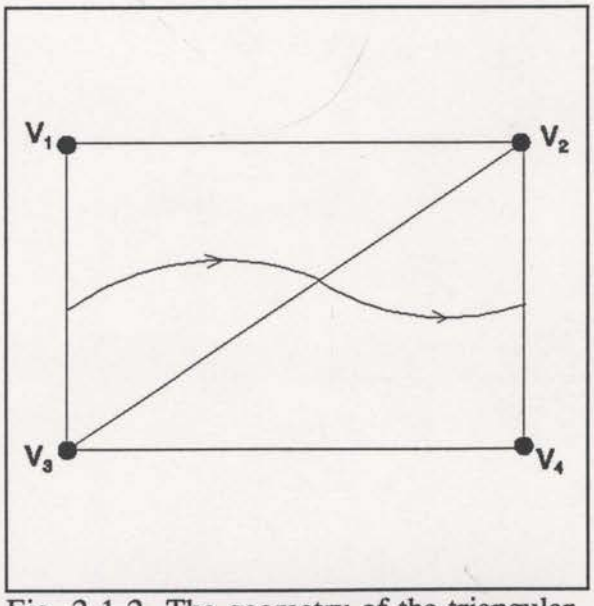


Fig. 2.1-2 The geometry of the triangular cell configuration and some typical circular arcs.

The ray tracing procedure was designed with a kernel program capable of tracing through any given triangular cell. This kernel program, named RAYTRACE

can be found in the appendix. The following is a summary of the flow of this program.

i) The input data are the spatial locations and velocities of the three grid nodes defining the triangular cell, along with the entry location and entry angle of the ray.

ii) Compute lateral and vertical velocity gradients by finite differencing across the corners of the triangle.

iii) Compute the velocity at the input ray location.

iv) Determine the angle between the velocity gradient vector and the reference axes and the magnitude of the gradient.

v) If the gradient is zero, flag the ray for straight ray tracing within this cell.

vi) Compute the radius of curvature for the ray's circular arc within this cell.

vii) If the radius is larger than a threshold value, flag the ray for straight ray tracing[1]. For the purposes of raytracing efficiently the threshold needs to be as low as possible, but surprisingly significant errors are easily created.

viii) Compute the centre of the circular arc.

Particular care must be taken to ensure that the circular arc is curving in the correct direction. That is, the centre must be chosen on the correct side of the ray[2].

---

[1] There will be more discussion on the required size of this threshold in a later section.

[2] The velocity at the correct centre must be zero, so it is easy to check the two choices.

ix) Intersect the circular arc with the sides of the triangular cell (see Figure 2.1-3), thus determining all possible exit points.

x) Determine the exit angle for each of the possible exit points[3].

xi) Compute the arc length from the entry point to each possible exit point around the circle (in the direction of energy transfer).

xii) The exit point with the shortest possible arc length is the required exit point.

The accuracy and integrity of this ray tracing code is of paramount importance for the traveltime inversion algorithm. Thus great care was taken to ensure that all possible ray configurations were correctly handled.

Most of the calculations performed by the subroutine RAYTRACE are performed with double precision calculations to avoid rounding errors. Originally, a threshold of 10^4 metres was set for circular rays. However, a serious problem was uncovered while analysing constraint statistic derivatives (see section 2.4). It was found that this was not accurate enough and that the threshold had to be increased to a radius of 10^6 metres.

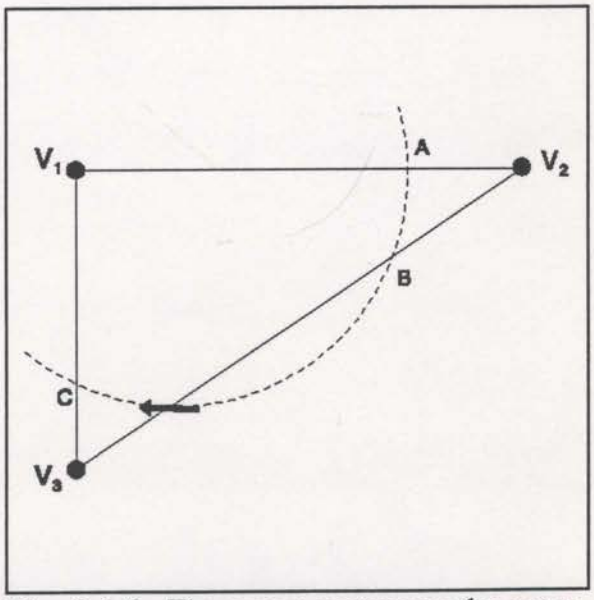


Fig. 2.1-3 The arrow represents the entry location and orientation of the ray. The dashed curve is the circular arc that the ray must traverse. A, B and C are possible exit points.

[3] One needs to be mindful of the direction of energy transfer around the circle, therefore being able to assign a 360° range of angles.

One potential problem with the ray tracing scheme as defined above, is an artificially introduced asymmetry. As can be seen from the example in Figure 2.1-2, the four velocity nodes can be split into triangles with a diagonal line sloping upwards to the right, or equivalently, upwards to the left. Irrespective of the sense of triangulation, such cells can make symmetric velocity models slightly asymmetric. However, as long as the cell size is small compared to the features being resolved, the induced asymmetry will be negligible.[4]

The ray tracing algorithm described above is computationally intense. Langan et al., 1985, go to considerable lengths to avoid the evaluation of trigonometric functions. No such compromises were made in the studies discussed in this thesis.

There are other approaches to the ray tracing problem which may prove adequate for reflection tomography applications, while possibly being more efficient. These approaches include finite difference traveltime modelling (eg. Vidale, 1989) and shortest path calculations (Moser, 1991). Currently, the ray tracing portion of the reflection tomography algorithm is the most time consuming component, taking up to 80% of the total computation time (depending on the size of the model; see section 4).

---

[4] The observations made about asymmetry here are reinforced by Bregman et al., 1989. Later in this thesis (Figures 2.2-4, 2.2-5 and 2.5-1) some synthetic examples display significant asymmetry even though the model and raypaths appear symmetric. This is not a result of the triangular cell shape. The model traveltime data were derived with exactly the same raytracing code and triangular cells. That is, the models themselves effectively have an asymmetric velocity distribution even though the structure of the anomalous velocity nodes is symmetric. The raypaths used for these examples are symmetrical with respect to the extent of the model, however, this means that they sample the edges of the (asymmetric) anomaly differently. The asymmetry of the solution is a result of this sampling and the subsequent response of the inversion algorithm to the underdetermined problem.

2.2 Removal of reflectors from problem parameterisation.

The conventional reflection tomography algorithm, as introduced by Bishop et al., 1985, requires a set of specific reflectors to be chosen and in turn, that all traveltime data picked must be for energy that has reflected from one of these reflectors. These restrictions were imposed to enable two-point ray tracing to compute theoretical travel times between known source and receiver locations. However, these restrictions introduce a lot of practical difficulties into the conventional algorithm. Firstly, it is often difficult to define an adequate set of reflectors in areas were reflection tomography could be useful (eg. structurally complex areas). It is very difficult to ensure that travel times for these reflectors are correctly interpreted from the pre-stack gathers. A further difficulty is that these chosen reflectors must have depths assigned before the tomographic inversion can begin. If no well data is available it is not easy to assign accurate depths. Leger et al., 1989, gave a graphic description of how difficult the travel time interpretation can be; they had to repick the data many times in order to get the correct inversion (luckily, they knew the type of results required). Finn and Backus (1991) made similar comments regarding the difficulty of traveltime picking during their presentation. They found the picking so difficult, they stopped picking travel time data for one corner of their prospect. Lailly et al. (1992) described how difficult the traveltime interpretation can be in areas with complex structural geology.

A lot of research effort in recent years has been aimed at reducing the effort required to extract the traveltime information from the data and prepare the initial

model. Here it is assumed that any algorithm designed to estimate the subsurface interval velocity field from the travel times of reflected seismic energy, is a form of reflection tomography. Along these lines, many researchers have developed algorithms to achieve the results of conventional reflection tomography, but with the emphasis on the ease of application.

Biondi (1988, 1990) developed an algorithm for subsurface interval velocity estimation with the help of "beam-stacks". The purpose of using beam-stacks was to remove the need to pick the traveltime data. An inversion procedure was set up to find the interval velocity distribution that maximised the resulting beam-stack energy. Beam-stacking is essentially a localised slant-stacking process across a gather of traces. Because it is localised, it is capable of resolving the non-hyperbolic moveout associated with complex velocity fields. Since a small group of traces are beam-stacked, the method is less troubled by noisy data, but its resolution is slightly decreased. After performing the beam-stacks, it is possible to find the velocity model that maximises this beam-stack energy. This velocity model is effectively a tomographically derived velocity field, and manual data picking was not necessary.

A somewhat similar approach was made by Toldi, 1989. He used CMP stacking semblance to derive an interval velocity model without the need for picking any data. This algorithm simplifies the practical aspects of reflection tomography, but its range of application is reduced. Given a proposed interval velocity model, Toldi used computed stacking velocities to evaluate the overall stacking semblance of the data (similar to overall beam-stack energy). Iterative changes to the velocity model were made to maximise the overall semblance. The technique is restricted by the use of stacking semblance values which are based on the hyperbolic moveout assumption.

Success can only be expected for velocity models that are smooth enough to produce roughly hyperbolic moveout.

Picking the required traveltime data after pre-stack depth migration has been proposed to simplify the conventional reflection tomography algorithm (Van Trier, 1990; Harlan et al., 1991a; Jacobs et al., 1992; Lailly et al., 1992; Stork, 1992). These researchers have acknowledged that traveltime data picking is very difficult and laborious, but they have noticed that it becomes easier if the data has been pre-stack depth migrated with an approximate velocity field. When picking the data from common-offset gathers, the reflection events are easier to interpret after an approximate migration. Accommodating traveltime data after pre-stack depth migration requires modifications to the procedure of Bishop et al., 1985, primarily for Frechet derivative computation[1]. In general, this type of algorithm has the computationally expensive pre-stack depth migration process between each iteration. The bulk of this expense can be avoided with approximate methods of residual migration (Van Trier, 1990). Traveltime data picking has not been avoided but it has been made easier.

Many other simplifications to the conventional reflection tomography algorithm (Bishop et al., 1985) have been published. Van der Made and Van Riel (1988) used stacking velocity values and zero-offset traveltimes to invert for the subsurface interval velocity field. Using stacking velocities without offset-dependent traveltimes is an extreme case of avoiding pre-stack data picking. Similarly, Sherwood (1989) generated hyperbolic traveltime data directly from stacking velocities. These methods are simple to use but they have difficulties in areas of complex velocity structure. Harlan

[1] Although Jacobs et al., 1992, and Lailly et al., 1992, suggest inverse raytracing of the picked migrated data to predict the pre-migration traveltimes.

(1989,1991b) simplified the data picking by incorporating stacking velocities and simplifying the data set to be interpreted. These research efforts demonstrate the difficulty of traveltime data picking for the conventional reflection tomography algorithm.

Two-point raytracing is used to compute theoretical traveltimes in the conventional reflection tomography algorithm. This approach to raytracing requires specifically defined reflectors and a fan of rays to be traced for every shot and each reflector (Figure 2.2-1). This is why the reflector depths must be specified as parameters of the inversion model - however, they are not known and must be allowed to vary. These extra parameters complicate the problem of determining the interval velocity field, introducing issues like velocity/reflector depth trade-off. Complications such as these are avoided when it is realised that two-point raytracing is not essential.

Conventional reflection seismic data is recorded in gathers of closely spaced traces. The array of receivers makes it possible to determine the angle of incidence of

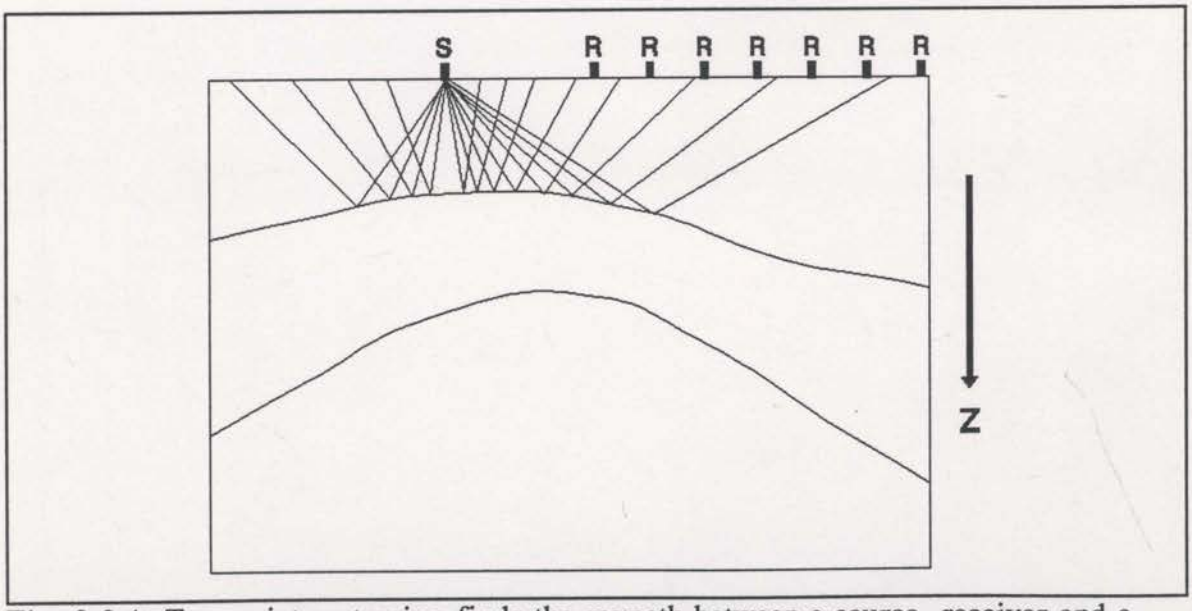


Fig. 2.2-1 Two-point raytracing finds the raypath between a source, receiver and a given reflector. A fan of rays is shot with interpolation to the actual receiver location.

the seismic energy with the surface. If the near surface velocity is known, then the apparent dip of some reflected energy on a common-source gather is all that is required to compute the emergence angle of the energy at the receivers (see Figure 2.2-2). Similarly, the apparent dip of the same energy on a common-receiver gather gives the impingement angle at the source array. Having both surface angles, it is possible to trace both ends of the ray back into the model until they meet (Figure 2.2-2). Reflected traveltimes are computed without any defined reflectors. This approach to raytracing will be called two-end raytracing in contrast to two-point raytracing.

Sword (1986) made use of two-end raytracing in reflection tomography. He also observed that errors associated with estimating the surface angles can be detrimental to the inversion procedure. Since the angles are given by dips measured from the data, there will always be some measurement error. Sword (1986) found that such errors could make the estimated traveltimes unstable - especially if the source and receiver are relatively close together (see Figure 2.3-3). Fortunately, Sword discovered another statistic, X_{err}, which is more stable than the traveltime. This new statistic is the spatial separation of the two ends of the ray at the depth Z where the total traveltime equals the actual traveltime. If the velocity model is correct, the two ends of the ray must meet at this depth and X_{err} will be zero. This statistic makes it possible to use two-end ray tracing in reflection tomography.

The X_{err} statistic was compared to the traveltime statistic in a simple synthetic reflection tomography experiment. The actual velocity model comprised grid points with a velocity of 2000 m/s, except for one anomalous point of 2050 m/s. A set of synthetic rays were generated in a symmetric pattern over the anomalous velocity.

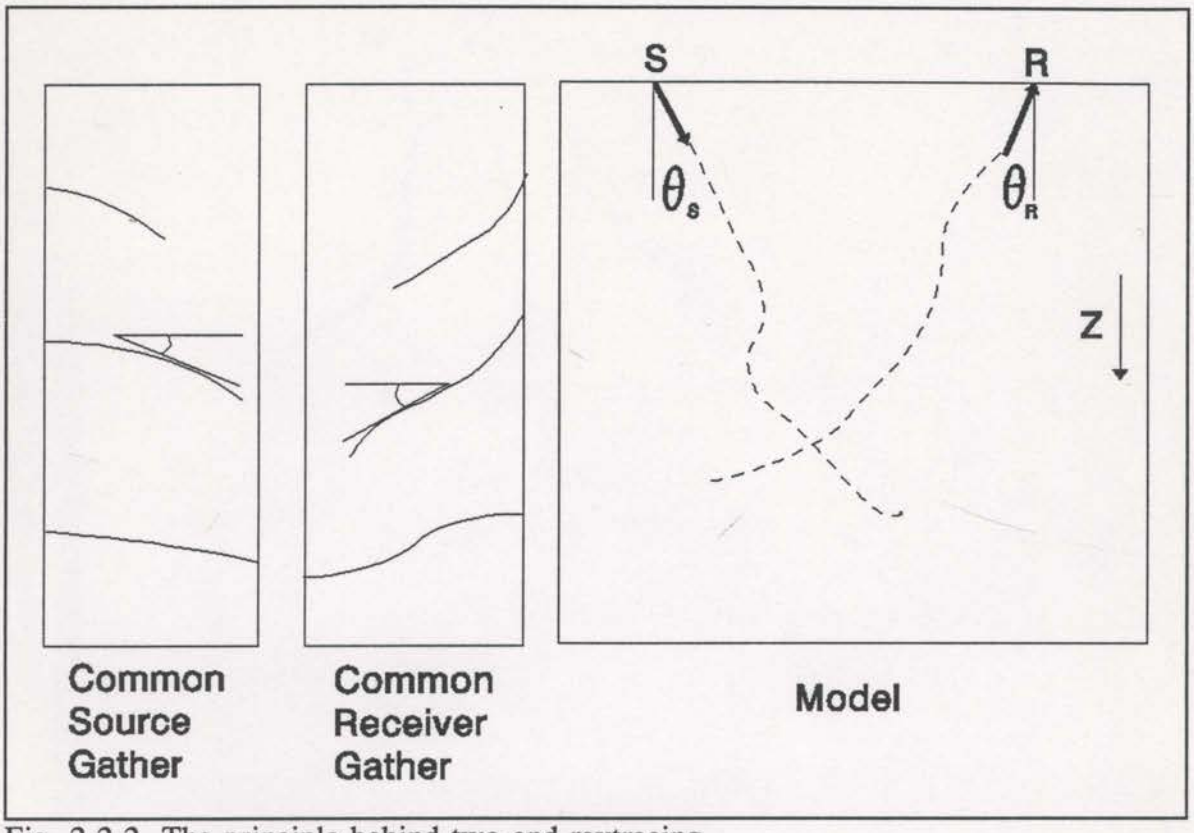


Fig. 2.2-2 The principle behind two-end raytracing.

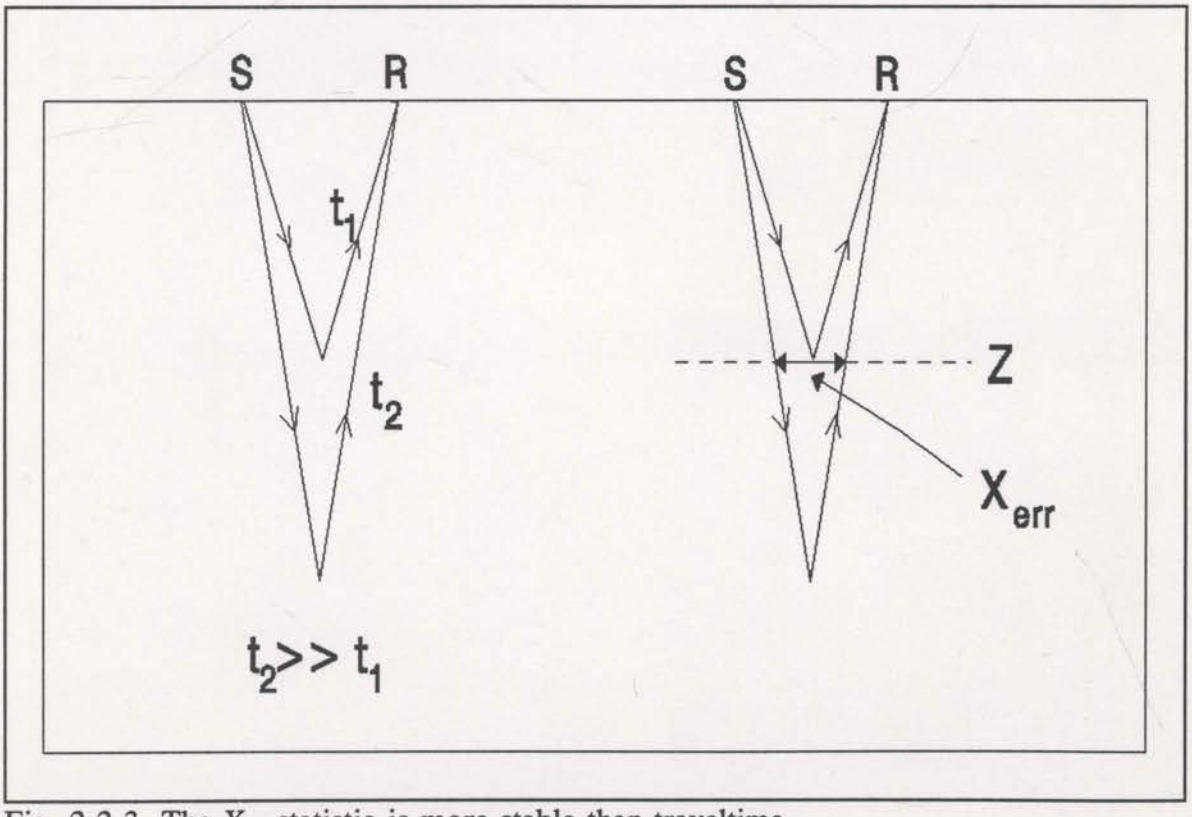


Fig. 2.2-3 The X_{err} statistic is more stable than traveltime.

Figure 2.2-4 shows the results of an inversion which used traveltimes as the constraint statistic. This inversion has performed poorly. Conversely, Figure 2.2-5 displays the successful results of the inversion of the same problem with the use of X_{err} as the constraint statistic.

2002	2000	2004	2017	1976	2017	1979	2004	2001	2003	2007
2000	1998	2013	2017	1981	2011	1982	2005	2007	2006	2002
2000	1999	2000	2027	1988	2001	1987	2011	2008	2001	2000
2000	1999	2004	2005	2010	2004	1996	2009	2002	2000	2000
2000	2000	2005	2014	2002	1996	2011	2004	2002	2000	2000
2000	2000	2004	2014	2020	1969	1996	2002	2002	2000	2000
2000	2000	2000	2009	2011	1966	1991	1995	1999	2000	2000
2000	2000	1998	2001	2026	1958	2002	1998	1996	2000	2000
2000	2000	2000	1999	2000	2001	1998	2000	1999	2000	2000
2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000
2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000

Fig. 2.2-4 Inversion results using traveltimes. 43 rays were used. The marked velocity should be 2050 m/s (2000 m/s for the rest). (See the footnote on page 15)

2003	1999	2004	2004	1998	2002	1998	2004	2000	2001	2003
2000	2000	2012	2000	1997	2000	1996	2001	2007	2003	2001
2000	1999	2002	2004	1988	1991	1996	2009	2010	1999	2000
2000	2000	2006	1995	2010	1990	2004	2008	2003	2000	2000
2000	2000	2008	2010	2001	2042	1997	2006	2006	2000	2000
2000	2000	2005	2017	2011	1994	1992	2005	2003	2000	2000
2000	2000	2000	2005	2001	1994	1992	1988	1998	2000	2000
2000	2000	1997	1996	1998	1997	1999	1994	1996	2000	2000
2000	2000	1999	1996	1999	1998	2000	1999	1998	2000	2000
2000	2000	2000	1999	2000	2000	2000	2000	2000	2000	2000
2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000

Fig. 2.2-5 An identical inversion to that of Fig. 2.2-4 except that X_{err}'s were used instead of traveltimes. (See the footnote on page 15)

It has been claimed that this method of raytracing, without reflectors, will suffer because it does not force all reflections from a single reflector to focus at the same depth (Sword, 1986). The depth location of the ends of two adjacent rays that reflect from the same reflector, may be slightly different due to errors in the surface angle estimation, for example. Conventional reflection tomography forces rays like these to reflect at the same depth. However, it is not clear whether this reflector constraint does not cause more problems than it solves. More difficult picking and velocity/reflector depth trade-off problems certainly appear to be very substantial problems.

An advantage of using two-end raytracing is the increased number of data that can be picked. Without pre-defined reflectors it is a simple matter to include any reflected/scattered energy as data, even from very small reflectors. Diffracted energy as well as energy that has reflected from fault planes are now naturally admissible. The result is more data with a greater range of angular coverage (from diffracted/fault plane energy), both of which will enhance the results of the inversion process. The extra rays that can be defined may well compensate for the focussing issues described earlier.

Two-end raytracing can also avoid the expenses of two-point raytracing. For accurate two-point raytracing, more rays have to be traced than there are receivers. That is, for each required raypath, more than one ray is traced because the final surface location of the ray cannot be predicted. The required traveltime is interpolated from the traced times. With two-end raytracing, exactly one ray is traced for each required raypath. A further saving is afforded since no reflection logic is required.

2.3 Automatic data picking using complex attributes.

Conventional reflection tomography requires specific events to be interpreted and careful, manual picking is a major and laborious step. Interactive workstation software has been proposed to make this step less arduous (Harlan et al., 1991a; Harlan et al., 1991b). However, two-end raytracing makes it possible to let a computer do all the work. Two-end raytracing (see section 2.2) makes traveltime data picking virtually unrestricted. Any reflected energy[1] can be picked as long as it is coherent enough to enable its dip to be estimated (on shot and receiver gathers). Automatic data picking is potentially simpler and more successful.

Sword (1986) used localised slant-stacks to identify reflection events and estimate their apparent dip. This approach applies a series of time shifts to a small number of traces from a common source or receiver gather, followed by summation. This slant-stacking process transforms the small group of traces from the time/offset $(t\text{-}x)$ domain to the centre trace intercept/ray parameter[2] $(\tau\text{-}p)$ domain;

$$\hat{y}(\tau,p_j) = \sum_{i=\frac{-(N-1)}{2}}^{\frac{(N-1)}{2}} y(\tau - ip_j\Delta x, i\Delta x)$$

where $y(t,x)$ is the original data set,

t is the travel time,

x is the offset from the centre trace of the group,

---

[1] Including diffracted events, fault-plane reflections and multiple arrivals.

[2] The ray parameter is the inverse of apparent horizontal velocity.

N is the number of traces in the group (odd),

τ is the centre trace intercept time,

p_j is the ray parameter equivalent to one of the j dips being searched for, and,

Δx is the spatial separation of the traces.

Once the data have been transformed in this way, the τ-p traces are searched for amplitude maxima. Each maximum defines a time on the centre trace and an apparent dip. These results are assigned to the centre trace location and the group of traces is rolled along by one trace, and a new τ-p transform is performed.

Once all traces of every common shot gather have been transformed and picked, the data are re-ordered into common receiver gathers. The τ-p transformation and picking process is repeated on each common receiver gather. In this way, both the apparent shot and receiver domain dips are obtained, as required for two-end raytracing. This process is successful (Sword, 1987), but it is also very computationally intense.

In 1988, Scheuer and Oldenburg described an algorithm that is capable of extracting the same information as achieved with localised slant stacks. This technique uses the complex attributes of the seismic traces. It is a simple process to compute the instantaneous frequency and instantaneous wavenumber at any point on a gather, and then take their ratio to give the apparent dip at that point. Obtaining the instantaneous attributes requires a single fast Fourier transform and its inverse for each trace. Such an algorithm will require considerably less computational effort than the previously described slant stack approach.

Complex trace attributes are derived from relatively simple theory. It can be assumed that any recorded trace is the real projection of some complex trace. Any imaginary component can be assigned to create such a trace, but only one particular imaginary component retains the characteristics of the original trace. Since the recorded trace is real and causal (zero amplitude for $t<0$), its Fourier transform is hermitian. Either the positive or negative half of such a Fourier spectrum constitutes enough information to recover the original total spectrum. It will be shown that zeroing the negative frequencies creates a complex trace with an imaginary part that is the Hilbert transform of the real part (the original trace). Such a complex trace allows for the evaluation of instantaneous attributes (see Bracewell, 1978).

If the recorded seismic trace is $x(t)$ then the complex trace, commonly referred to as the analytic signal, is

$$a(t) = x(t) + i\hat{x}(t)$$

where $\hat{x}(t)$ is the Hilbert transform of $x(t)$. The time domain Hilbert transform is defined as the time domain convolution with a digital filter $h(t)$ whose Fourier transform is $H(\omega) = -i\,\mathrm{sgn}\,\omega$. It is usually more efficient to apply the Hilbert transform in the frequency domain. Consider the Fourier transform of $x(t)$,

$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{i\omega t} dt \quad .$$

The negative frequencies can be suppressed by multiplying $X(\omega)$ by the unit step function[3], $U(\omega)$, and then

[3] The unit step function is zero for $\omega<0$, unity for $\omega>0$ and a 1/2 for $\omega=0$.

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} U(\omega).X(\omega)e^{-i\omega t}\,d\omega$$

is the inverse Fourier transform. The inverse Fourier transform of $U(\omega)$ is $u(t) = \frac{1}{2}\delta(t)+\frac{1}{i2\pi t}$ (see Berkhout, 1987), where $\delta(t)$ is the Dirac delta function. The convolution theorem gives[4],

$$\begin{aligned} f(t) &= \left(\frac{1}{2}\delta(t)+\frac{1}{i2\pi t}\right)*x(t) \\ &= \frac{1}{2}x(t)+\frac{1}{2}ih(t)*x(t) \\ &= \frac{1}{2}(x(t)+i\hat{x}(t)). \end{aligned}$$

Therefore, $f(t) = \frac{1}{2}a(t)$. In summary, the analytic signal is easily obtained by Fourier transformation, zeroing the negative frequency components and inverse Fourier transformation.

The analytic signal, or complex trace, allows the trace to be viewed as a phasor rotating about an axis (see Bracewell, 1978). The length of the phasor is the instantaneous amplitude and its rate of rotation gives the instantaneous frequency. The real projection of this complex trace is the recorded signal, $x(t)$, and the imaginary projection is $\hat{x}(t)$, known as the quadrature trace. The phasor is never allowed to rotate

---

[4] Note that the Fourier transform of $H(\omega) = -isgn\,\omega$ is $h(t) = \frac{-1}{\pi t}$.

backwards[5] - frequency is always positive. These attributes of the complex seismic trace allow for the rates of change to be evaluated in both the temporal and spatial dimensions, and can therefore be used for the computation of the local dip of the dominant plane wave energy.

The subroutine PICKSPC5 (see the appendix) was coded to compute the complex attributes for a common source gather and detect the apparent dips of the data within the gather. A separate subroutine, PICKCRP5 (also see the appendix), uses the picks made in the common source domain and extracts the corresponding apparent dips in the common receiver domain. The algorithm used by PICKSPC5 is outlined below.

a) Read a common source gather of traces.

b) Convert each trace to the frequency domain (one dimensional FFT's).

c) Zero all the negative frequency components.

d) Transform all traces back to the time domain.

e) Compute instantaneous amplitudes $\left(\sqrt{x^2(t)+\hat{x}^2(t)} = A(t)\right)$.

f) Pick maxima of instantaneous ~~of~~ amplitudes (not too close to each other).

g) Compute instantaneous phases $\left(\tan^{-1}\left\{\frac{\hat{x}(t)}{x(t)}\right\} = \phi(t)\right)$.

h) Extract eight nearest neighbour instantaneous phases around each maximum of instantaneous amplitude (see Figure 2.3-1).

i) Compute instantaneous frequencies for each of the three temporal triplets. Do this in two halves. For example, around point 2 (see Figure 2.3-1) compute $(\phi_3 - \phi_2)$

[5] Remember that the negative frequencies were zeroed.

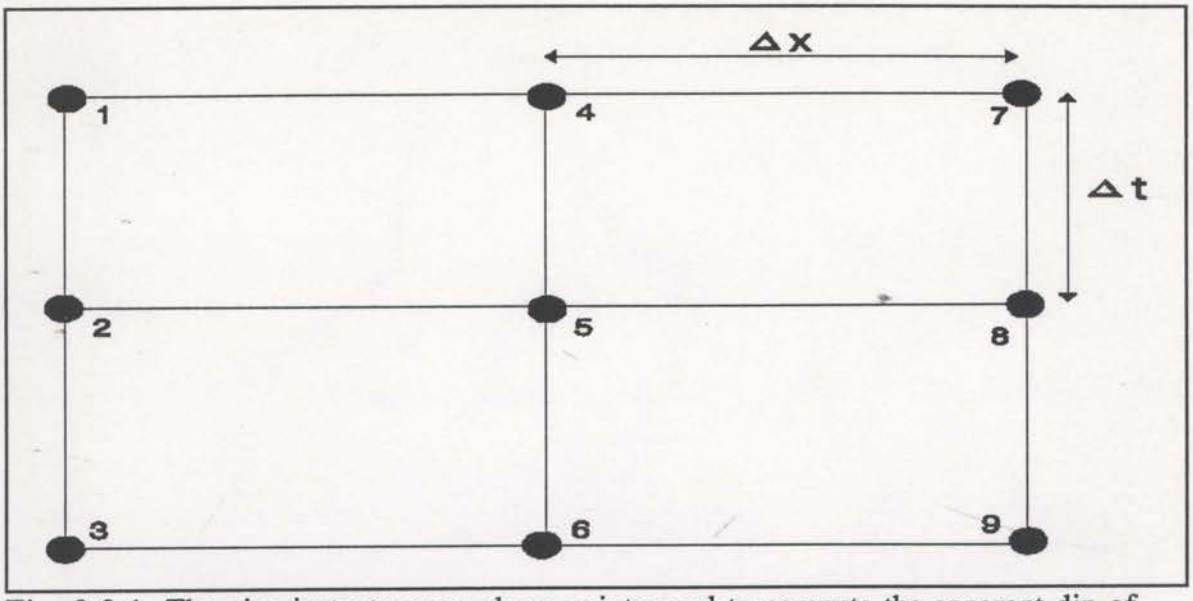


Fig. 2.3-1 The nine instantaneous phase points used to compute the apparent dip of the amplitude maxima at node 5.

and $(\phi_2 - \phi_1)$ and compare. If these values are quite different, declare the pick made at point 5 unstable, and discard it. Otherwise, average the two differences and divide by Δt to give the instantaneous frequency.

j) If the three estimated instantaneous frequencies are similar, average them.

k) Similarly compute the apparent wavenumber by using three consecutive horizontal points (see Figure 2.3-1). Note that the instantaneous wavenumber can be negative. It is this property that provides the sense of the apparent dip.

l) Compute the apparent velocity by dividing the computed instantaneous frequency by the apparent wavenumber.

m) Convert this to dip using the known surface velocity, V_{SURF},

$$DIP = \sin^{-1}\left(\frac{V_{SURF}}{V_{APPARENT}}\right)$$

n) Output this dip angle with the time of the instantaneous amplitude maximum

and the source/receiver locations to a file. This file will be used by PICKCRP5 (see the appendix).

o) Further consistency checks may be included before output. Such checks may, for example, only output if similar picks have been made nearby. Such extra checks are not always helpful.

Figure 2.3-2 shows the results of this algorithm when applied on some synthetic data. The picks seem to be generally representative of the data on the input gather.

All the initial testing of this complex attribute picking algorithm was performed on the synthetic data shown in Figure 2.3-2. Table 2.3-1 shows that the measured dips

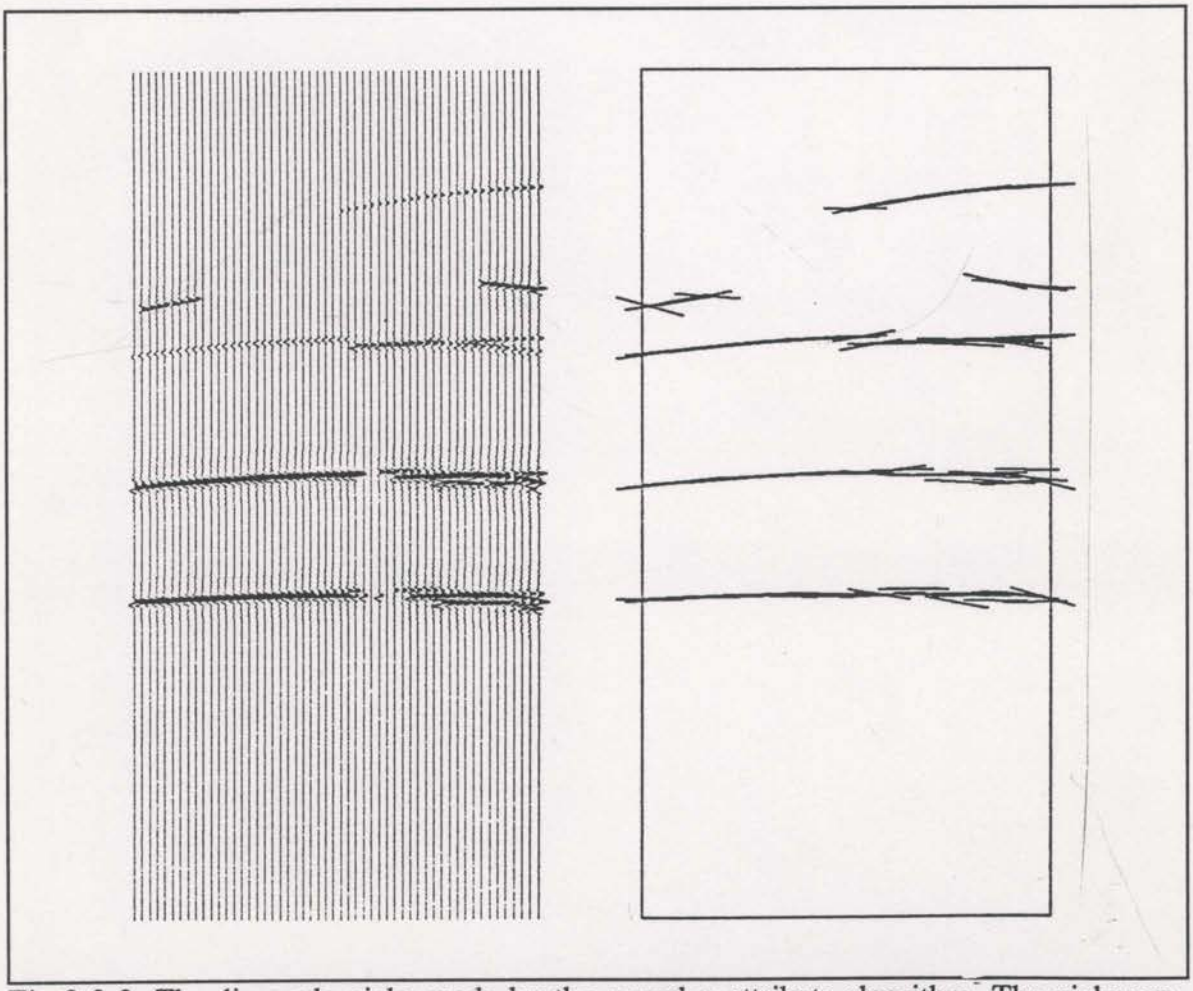


Fig 2.3-2 The dip angle picks made by the complex attribute algorithm. The picks are displayed as small dip bars centred on the pick location.

were clustered into groups of similar values separated by sharp jumps, whereas regularly changing dips were expected. This problem was eventually traced to the nearest millisecond accuracy of the synthetic trace generation program. It was surprising that nearest millisecond rounding was clearly observed by the complex attribute picker. The accuracy of the algorithm promises to be high. Table 2.3-1 also records another potential problem since the picked maximum amplitude points are output to the nearest sample period (4 milliseconds in this case). More on this later.

When the complex attribute picker is applied to real data, many more mispicks

Time	Measured Dip (radians)	Theoretical Dip (radians)
0.324	0.5956	0.5667
0.320	0.5005	0.5504
0.316	0.5895	0.5337
0.312	0.5915	0.5167
0.312	0.4126	0.4993
0.308	0.4125	0.4817
0.304	0.4952	0.4636
0.304	0.4986	0.4460
0.300	0.4155	0.4266
0.296	0.4069	0.4076
0.296	0.4133	0.3883
0.292	0.3274	0.3687
0.292	0.3271	0.3488
0.288	0.3267	0.3286
0.288	0.3275	0.3081
0.284	0.3267	0.2873
0.284	0.2441	0.2663
0.280	0.2405	0.2450
0.280	0.2453	0.2234
0.280	0.1617	0.2018
0.276	0.1588	0.1799
0.276	0.1619	0.1578
0.276	0.1630	0.1355

Table 2.3-1 Measured versus theoretical dip angles for the shallowest reflection of the synthetic data example (Model 4)

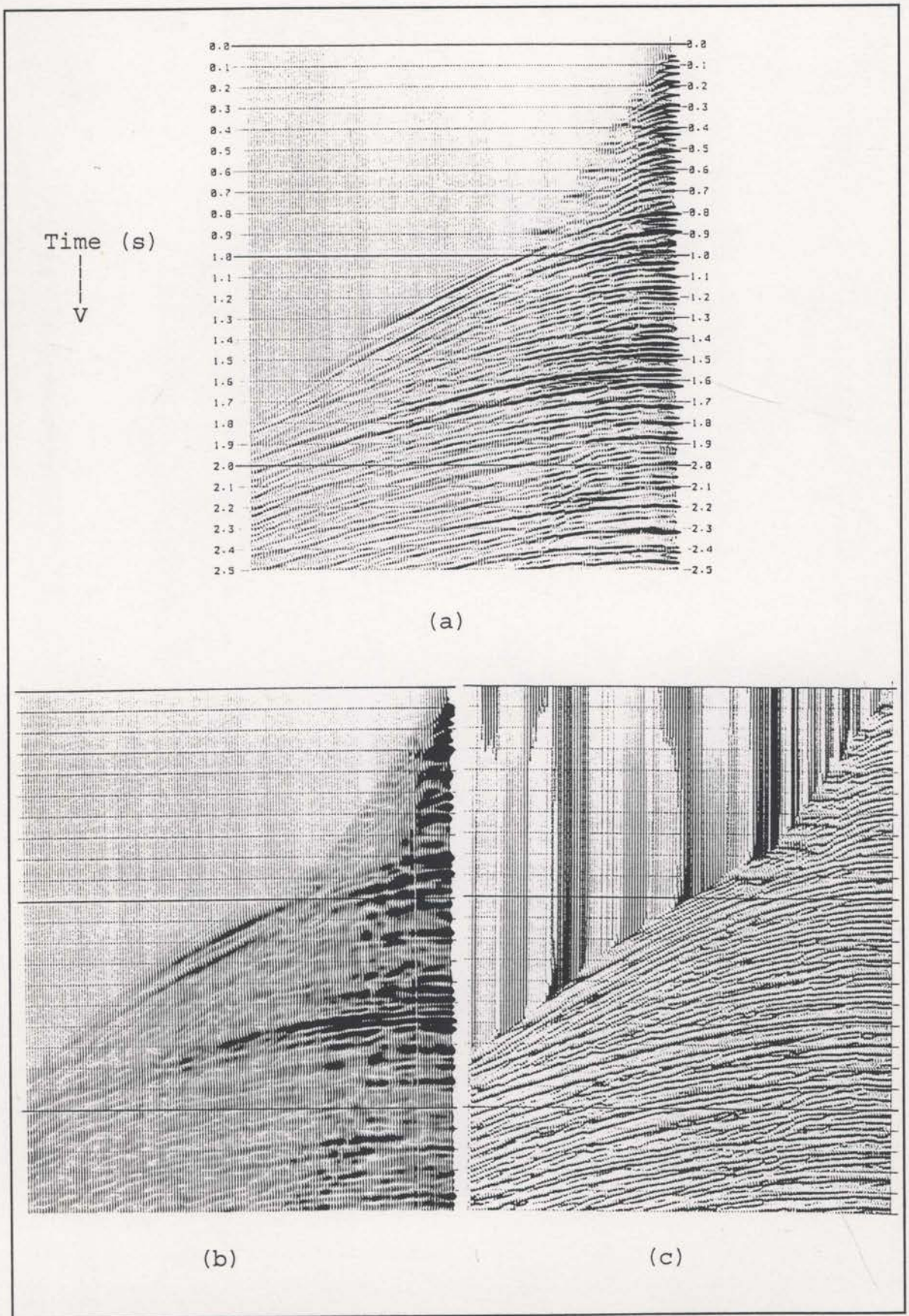


Fig 2.3-3 A common source gather (a) and its instantaneous amplitude (b) and instantaneous phase (c) representations.

are made since the events are not as distinct as on the synthetic data set. Figure 2.3-3 displays an example of a common source gather and its instantaneous amplitude and instantaneous phase representations. Figure 2.3-4 is an example of the picks made by the complex attribute algorithm on real data. It is normal for the phase values to change rapidly from values near $+\pi$ to values near $-\pi$. This is commonly known as "phase wrap-around". One has to be very aware of phase wrapping when computing the instantaneous frequency.

Two of the major reasons for the increased number of mispicks made on real

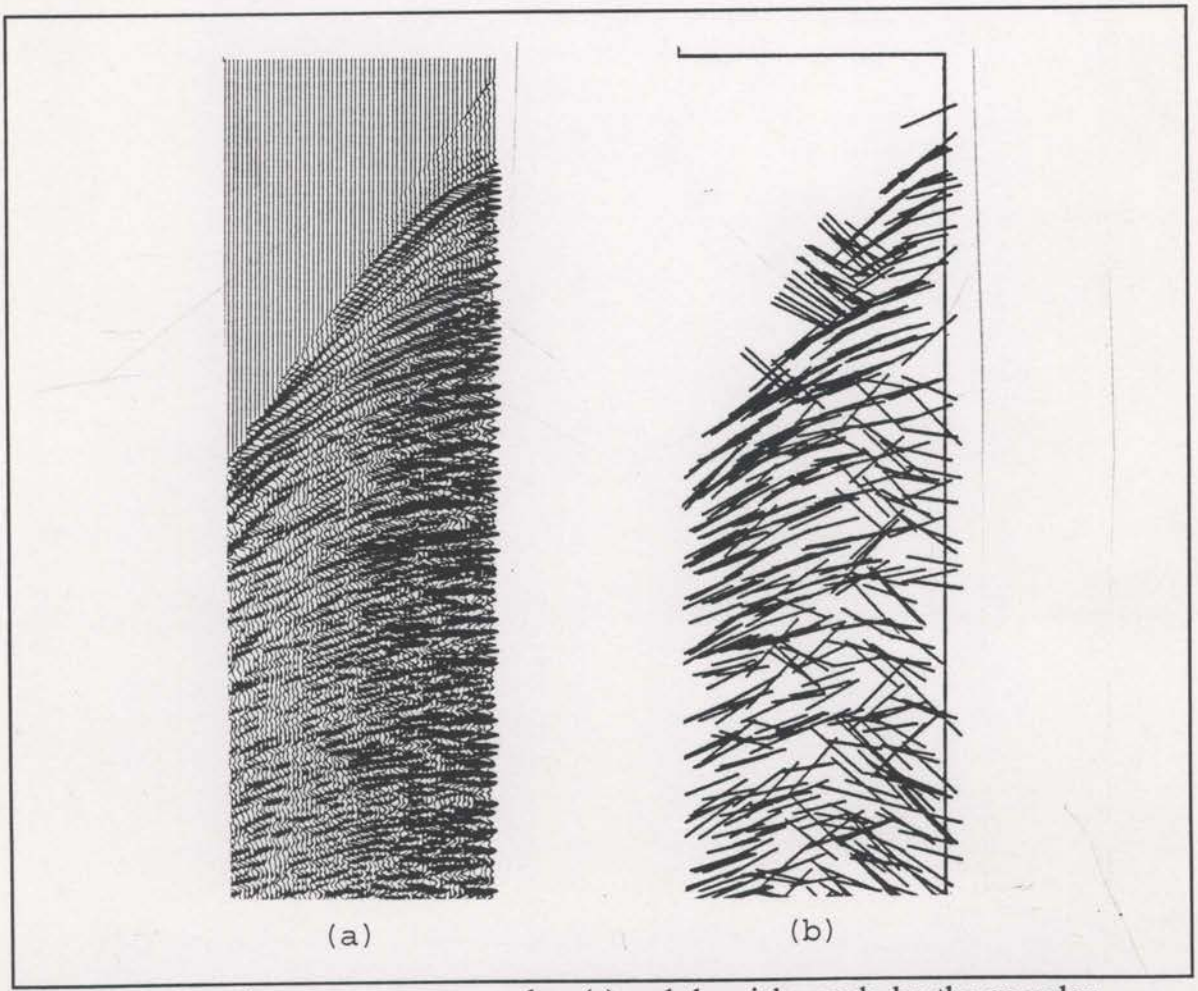


Fig. 2.3-4 A real common source gather (a) and the picks made by the complex attribute picker (b), presented as small dipping bars. These bars are centred on the pick location and they dip with the measured apparent dip.

[6] Scheuer and Oldenburg (1988) preferred velocity filtering for their applications. Velocity filtering is avoided here because it totally removes steeply dipping energy that may provide some useful information. High cut filtering provides the opportunity for such energy to be preserved. Velocity filtering is also avoided as it requires the added computational expense of a 2D FFT whereas high cut filtering requires only 1D FFT's.

data are crossing events and noise. Both of these can make the frequency and wavenumber estimates unstable. These problems can be reduced by careful preliminary processing of the seismic data. Filtering in the *f-k* domain can be used to remove linear, coherent noise and multiple reflections. Prediction error filtering in the *f-x* domain has been used in all real examples to attenuate random noise. Finally, because maxima in instantaneous amplitude are being picked, it is imperative that the data be processed to produce a wavelet with approximately zero phase.

With the complex attribute picker it is possible for events to be aliased. That is, steeply dipping events may be picked with a dip opposite to the actual dip. As an example, see the picks made on the steeply dipping coherent energy of Fig 2.3-4. Aliasing can occur because the instantaneous wavenumber can be positive or negative (between $-\pi$ and $+\pi$). When the dip of an event becomes large enough so that its instantaneous wavenumber is slightly greater than $|\pm\pi|$, the sign and magnitude of the computed wavenumber will be wrong. The only way to avoid aliasing, apart from preliminary velocity filtering, is to high cut filter the input data.[6] This reduces the rate of change of phase between adjacent traces and helps to keep the magnitude of the instantaneous wavenumber below π. Figure 2.3-5 has been high cut filtered at 35 Hz and shows reduced aliasing compared to Figure 2.3-4. However, high cut filtering also results in a decrease of resolution and tends to cause more problems with interfering events.

Scheuer and Oldenburg (1988) used a 2D FFT to compute the complex traces required for this type of data picking. It is more computationally intensive to use a 2D FFT instead of a 1D FFT for each trace, and because only the negative temporal frequencies are zeroed (see earlier in this section), the results will be identical. Scheuer

data are crossing events and noise. Both of these can make the frequency and wavenumber estimates unstable. These problems can be reduced by careful preliminary processing of the seismic data. Filtering in the *f-k* domain can be used to remove linear, coherent noise and multiple reflections. Prediction error filtering in the *f-x* domain has been used in all real examples to attenuate random noise. Finally, because maxima in instantaneous amplitude are being picked, it is imperative that the data be processed to produce a wavelet with approximately zero phase.

With the complex attribute picker it is possible for events to be aliased. That is, steeply dipping events may be picked with a dip opposite to the actual dip. As an example, see the picks made on the steeply dipping coherent energy of Fig 2.3-4. Aliasing can occur because the instantaneous wavenumber can be positive or negative (between $-\pi$ and $+\pi$). When the dip of an event becomes large enough so that its instantaneous wavenumber is slightly greater than $|\pm\pi|$, the sign and magnitude of the computed wavenumber will be wrong. The only way to avoid aliasing, apart from preliminary velocity filtering, is to high cut filter the input data.[6] This reduces the rate of change of phase between adjacent traces and helps to keep the magnitude of the instantaneous wavenumber below π. Figure 2.3-5 has been high cut filtered at 35 Hz and shows reduced aliasing compared to Figure 2.3-4. However, high cut filtering also results in a decrease of resolution and tends to cause more problems with interfering events.

Scheuer and Oldenburg (1988) used a 2D FFT to compute the complex traces

[6] Scheuer and Oldenburg (1988) preferred velocity filtering for their applications. Velocity filtering is avoided here because it totally removes steeply dipping energy that may provide some useful information. High cut filtering provides the opportunity for such energy to be preserved. Velocity filtering is also avoided as it requires the added computational expense of a 2D FFT whereas high cut filtering requires only 1D FFT's.

[7] This simpler algorithm will actually pick the group velocity and not the phase velocity. This distinction is in principle immaterial for the marine data being considered in this thesis (since the near-surface is non-dispersive). However, it may be significant for seismic data acquired on land. In this case a correction to the dips may be required. Also, dip estimation based on amplitude may be more problematic in the presence of crossing events.

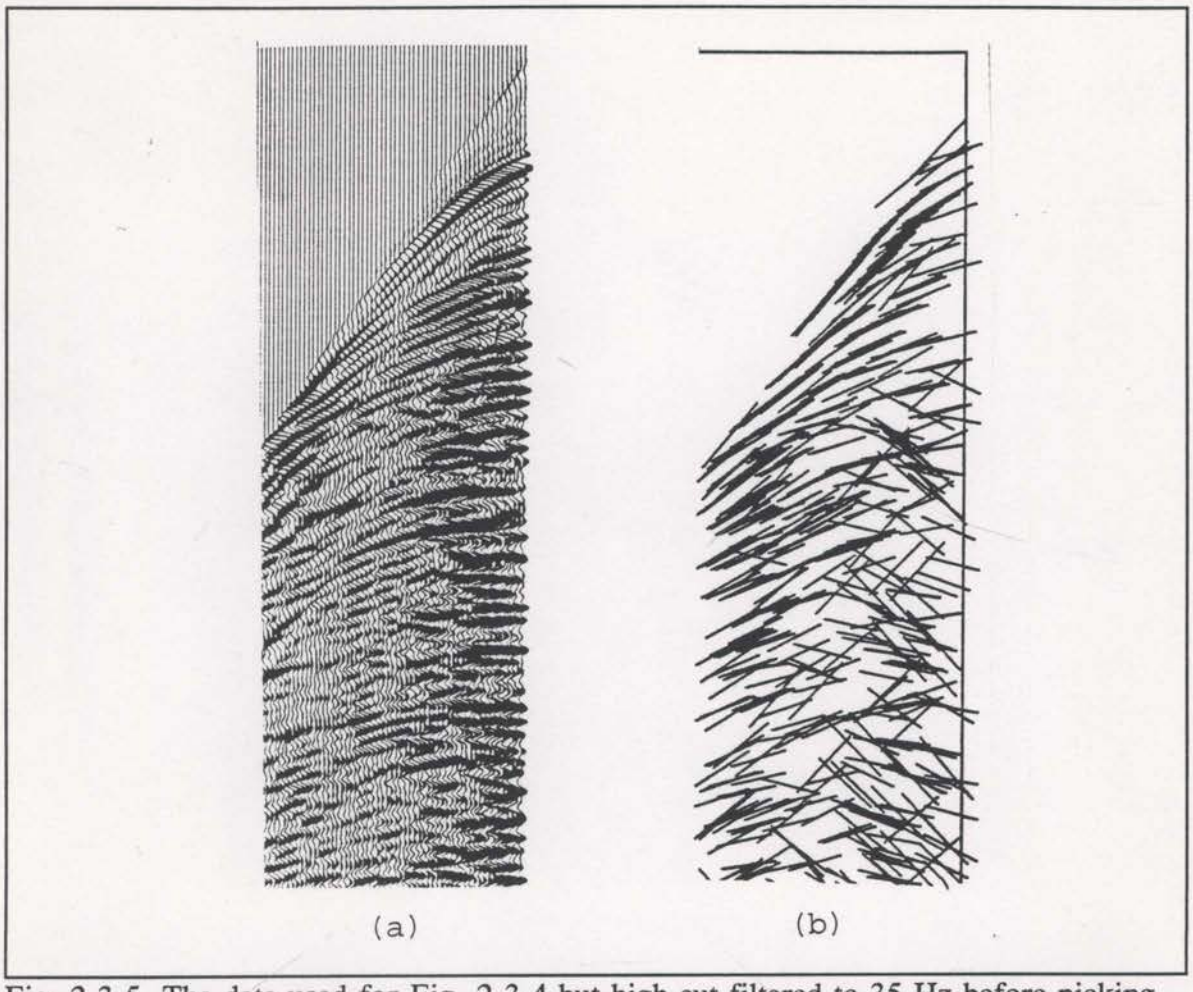


Fig. 2.3-5 The data used for Fig. 2.3-4 but high cut filtered to 35 Hz before picking.

and Oldenburg made use of the *f-k* domain for velocity filtering at the same time. All picks performed in this study used 1D FFT's.

The picking required here, for reflection tomography purposes, is only concerned with areas of maximum amplitude. Studying the rates of change of instantaneous phase may well be more detailed than is necessary. A much simpler approach would be to locate an amplitude maximum, search for the same maximum on adjacent traces, and compute the apparent dip.[7] The promise of such an algorithm is further simplicity and efficiency. A program called NPCKSPC2 (see the appendix) was coded for the purpose and the flow of this algorithm is;

a) Read a common source gather of traces.

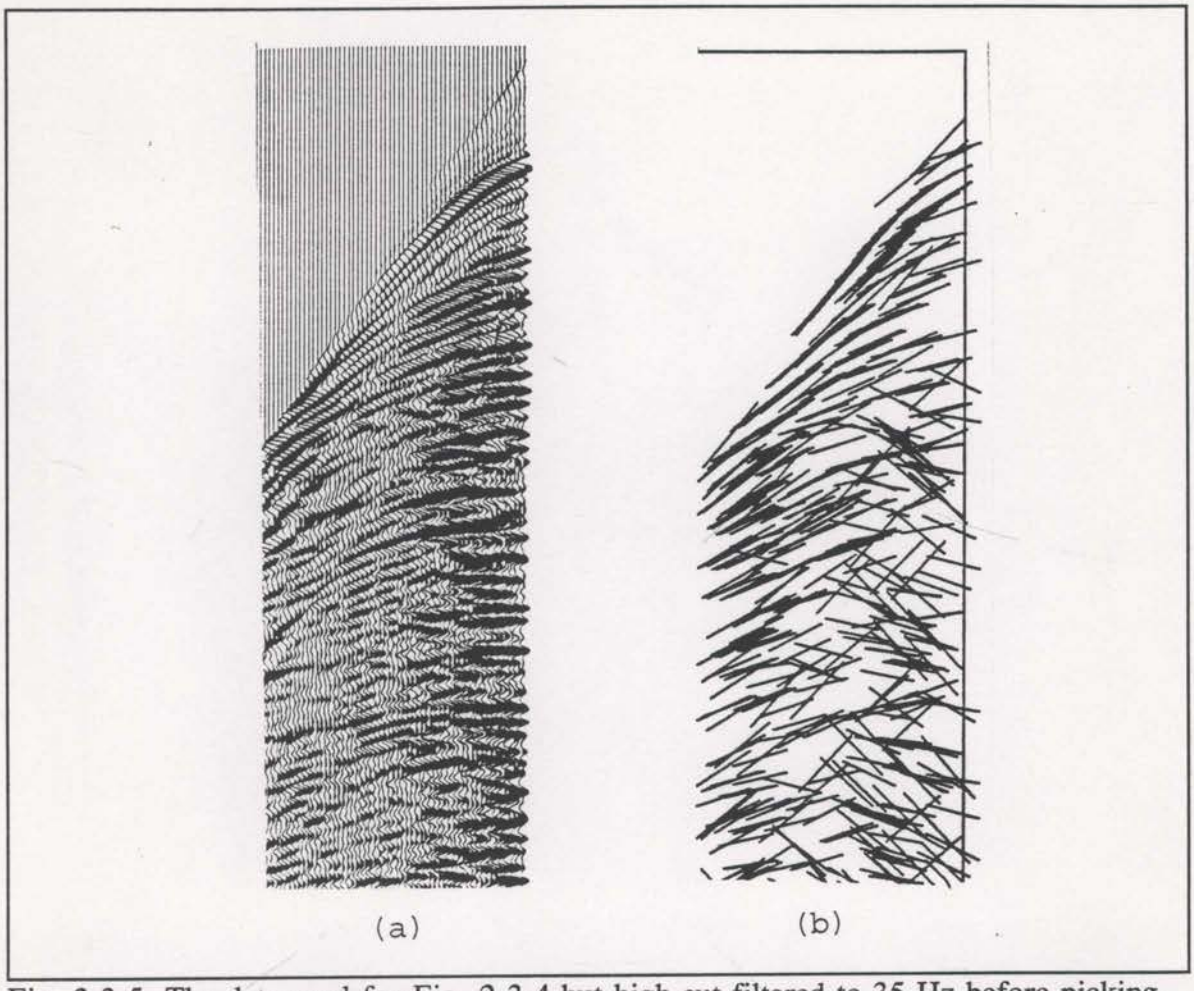


Fig. 2.3-5 The data used for Fig. 2.3-4 but high cut filtered to 35 Hz before picking.

and Oldenburg made use of the *f-k* domain for velocity filtering at the same time. All picks performed in this study used 1D FFT's.

The picking required here, for reflection tomography purposes, is only concerned with areas of maximum amplitude. Studying the rates of change of instantaneous phase may well be more detailed than is necessary. A much simpler approach would be to locate an amplitude maximum, search for the same maximum on adjacent traces, and compute the apparent dip.[7] The promise of such an algorithm is

[7] This simpler algorithm will actually pick the group velocity and not the phase velocity. This distinction is in principle immaterial for the marine data being considered in this thesis (since the near-surface is non-dispersive). However, it may be significant for seismic data acquired on land. In this case a correction to the dips may be required. Also, dip estimation based on amplitude may be more problematic in the presence of crossing events.

b) Compute the complex traces and the instantaneous amplitudes. The instantaneous amplitude trace defines the envelope of the input trace. When dealing with amplitudes, it is easier and simpler to use the amplitude of the trace envelope.

c) Search for amplitude maxima (not too close to each other).

d) Extract 12 instantaneous amplitude values around the maximum (see Figure 2.3-6).

e) Use quadratic interpolation to find the exact time of the maximum detected near sample 7. Given amplitude values, A_1, A_2 and A_3, with A_2 in the centre, the stationary point of the quadratic function passing through these is at

$$t_2 - \Delta t \frac{(A_1 - A_3)}{2(A_1 + A_3 - 2A_2)}$$

where t_2 is the travel time of the amplitudeA_2 . The stationary point is a maximum if $2A_2 > (A_1 + A_3)$.

f) For the adjacent traces, use quadratic interpolation to find the maximum, but be aware that the maximum may have moved out of the bounds of the three samples. This is the reason for extracting five sample from the adjacent traces (see Figure 2.3-6).

g) Compute the two dips implied by the maxima times on

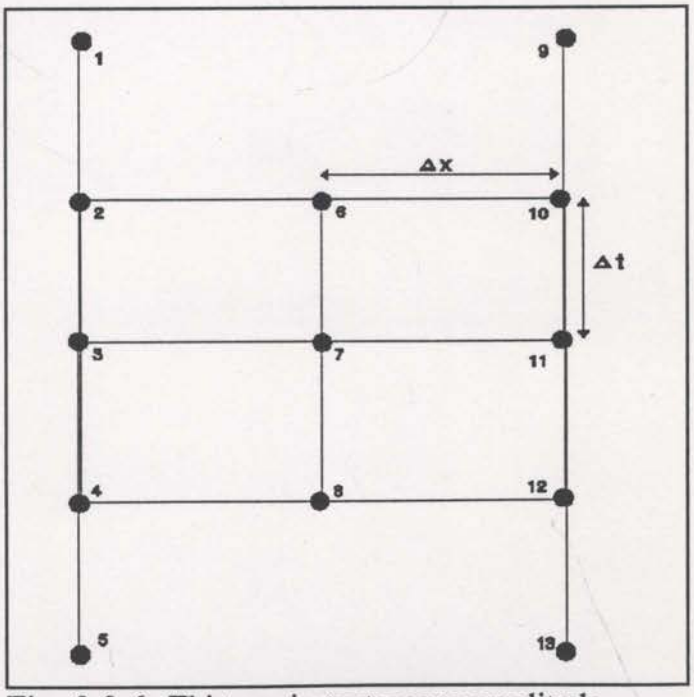


Fig. 2.3-6 Thirteen instantaneous amplitude values extracted around an amplitude maximum detected at sample 7.

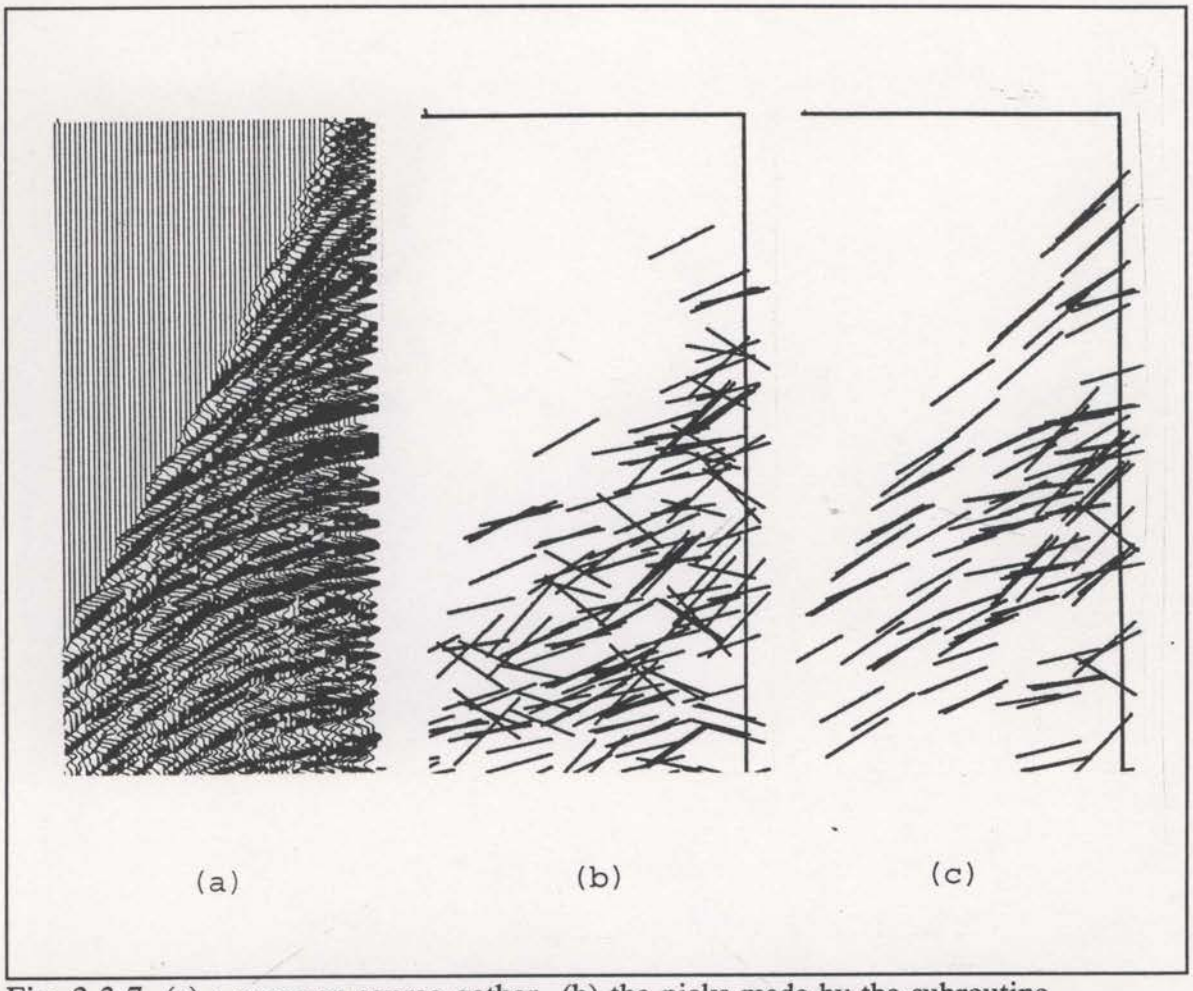


Fig. 2.3-7 (a) a common source gather, (b) the picks made by the subroutine PICKSPC5 and, (c) the picks made by the simple amplitude picker (NPCKSPC2).

either side of the target trace. If these dips are very different, ignore the point.

h) Convert the dips to velocities with the known surface interval velocity and average.

i) Output velocity, along with source and receiver locations, to file for use by a similar receiver domain subroutine (NPCKCRP2).

An example of the picks made by this algorithm can be seen in Figure 2.3-7. The new picks displayed here appear more stable than those made by PICKSPC5 (Figure 2.3-7(b)).

2.4 Semi-analytical derivative estimates.

A key factor in the success of an inversion procedure is the accurate estimation of the sensitivity of the constraint statistic to changes of the model parameters. Here the constraint statistic is the sum of the squares of all the X_{err} (see section 2.2) values,

$$C = \sum_k X_{err(k)}^2 \quad . \tag{2.4-1}$$

The model parameters are the discrete grid of interval velocities, V_{ij}. The required sensitivities are,

$$\frac{\partial C}{\partial V_{ij}} = \sum_k 2 X_{err(k)} \frac{\partial X_{err(k)}}{\partial V_{ij}} \quad , \tag{2.4-2}$$

where the derivatives on the right hand side are commonly called Frechet derivatives. These sensitivities define the local linear approximations of the n-dimensional C surface that are critical for the iterative inversion of the data (see section 4).

In both crosswell and reflection tomography, it is common to determine the Frechet derivatives analytically. Consider a raypath travelling through a velocity model defined by cells of constant velocity. The traveltime of this ray is,

$$t = \sum_k l_k s_k \quad ,$$

where l_k is the path length of the ray in the kth cell and s_k is the corresponding slowness[1]. Conventional tomography uses traveltime error as the constraint statistic,

---

[1] Slowness is the inverse of velocity, $s_k = \frac{1}{V_k}$.

and the Frechet derivatives are,

$$\frac{\partial t}{\partial s_k} = l_k \quad .$$

In this way, if slownesses are used as the model parameters, the Frechet derivatives are exactly the cell path lengths. However, this method assumes that the raypath does not change as the velocities change (ie. $\partial l_k / \partial s_k = 0$). This assumption can be justified by invoking Fermat's principle which states that the raypath between any fixed points will be that with minimum traveltime. The raypath is stationary and will not change with small changes in velocity.

With the strategy of two-end raytracing without reflectors (see section 2.2), the reflecting surface is not defined at a fixed location and Fermat's principle does not apply. The reflection point is allowed to move arbitrarily and $\partial l_k / \partial s_k$ cannot be assumed to be zero. Analytic derivative estimates become very difficult, if not impossible. It is important that the Frechet derivatives are accurately estimated but computational efficiency is also paramount.

It is possible to continue with analytic derivative estimates if the range of velocity variations is restricted. Zhou et al., 1992, used analytic derivative estimates with a first-order surface fitted to the four nodes of a square cell. Biondi (1988, 1990) simplified the derivative estimation by allowing only velocity fields that can be defined by B-splines.

If analytic derivative estimates are not possible, finite-difference estimates can be used. This procedure requires the velocity at one of the nodes to be perturbed slightly, after which the ray is retraced and it is observed how the traveltime (or X_{err})

changes. This has to repeated for every velocity node and every ray. It quickly becomes very expensive in terms of computation time. Sword (1986) reduced the expense by performing fewer perturbations and propagating the results with a transfer matrix. With the triangular cells and arbitrary velocity variations of the current study, analytic derivative estimates are not possible, but accurate derivative estimates can be obtained with a modified finite-difference scheme.

Consider a single ray passing near a given velocity node, as in Figure 2.4-1. When considering the effect of a perturbation of this velocity node, it is recognised that the travel path in a six cell area of influence will be directly affected. An efficient method of estimating the sensitivity of X_{err} with respect to velocity ($\partial X_{err}/\partial V_{ij}$) will be

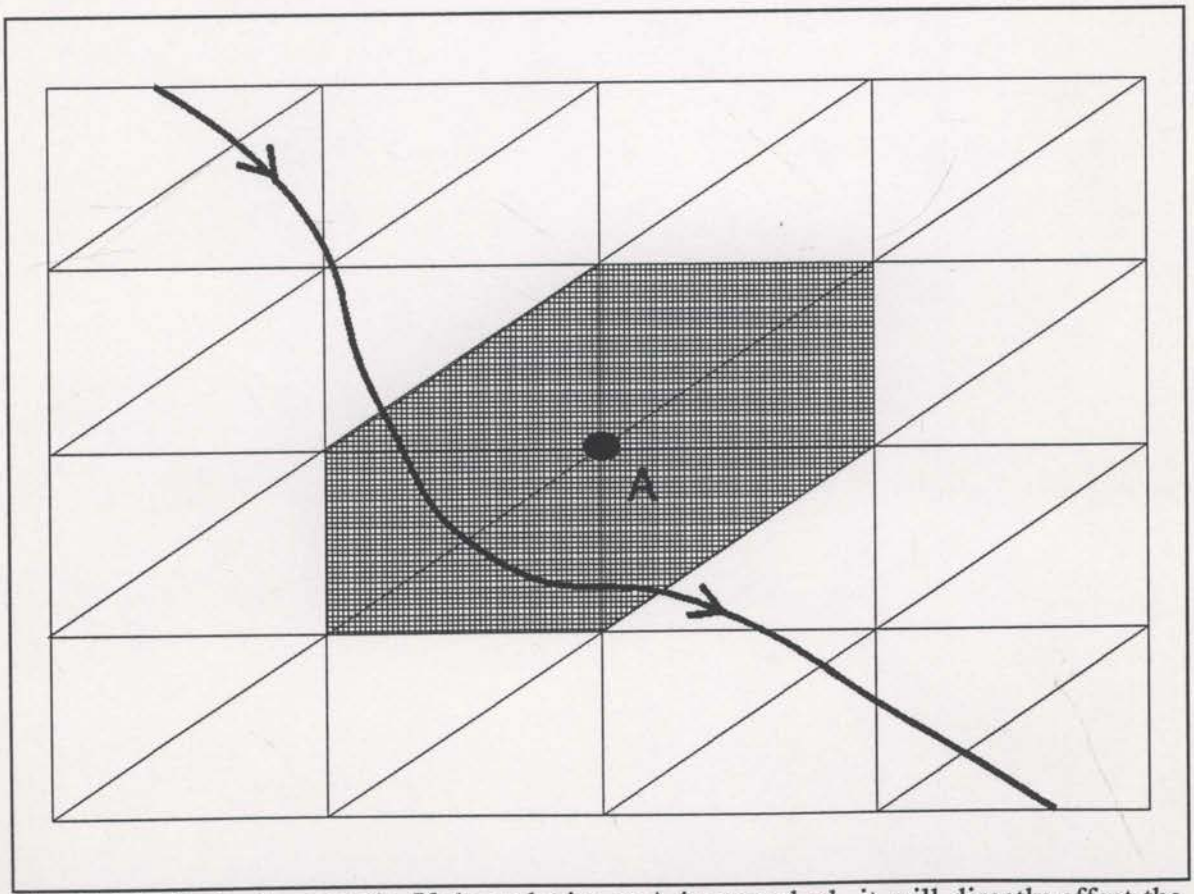


Fig. 2.4-1 A single raypath. If the velocity at A is perturbed, it will directly affect the travel path within a six cell zone of influence (shaded).

achieved if the raypath outside the area of influence is assumed to maintain its shape and is only re-oriented by the raypath changes within the area of influence. In this way the only extra ray tracing required will be within the six cell area of influence. The complete raypath will not have to be retraced as is necessary for finite-difference estimates.

Figure 2.4-2 demonstrates the principle of the proposed Frechet derivative estimation algorithm. The raypath shape from any cell to the end of the ray is assumed fixed, and is effectively defined by a vector $\mathfrak{R}$. The velocity perturbation of the node in question alters the raypath within the zone of influence, giving a new exit location and angle. These new exit parameters can be used to re-orient $\mathfrak{R}$, enabling a new X_{err}

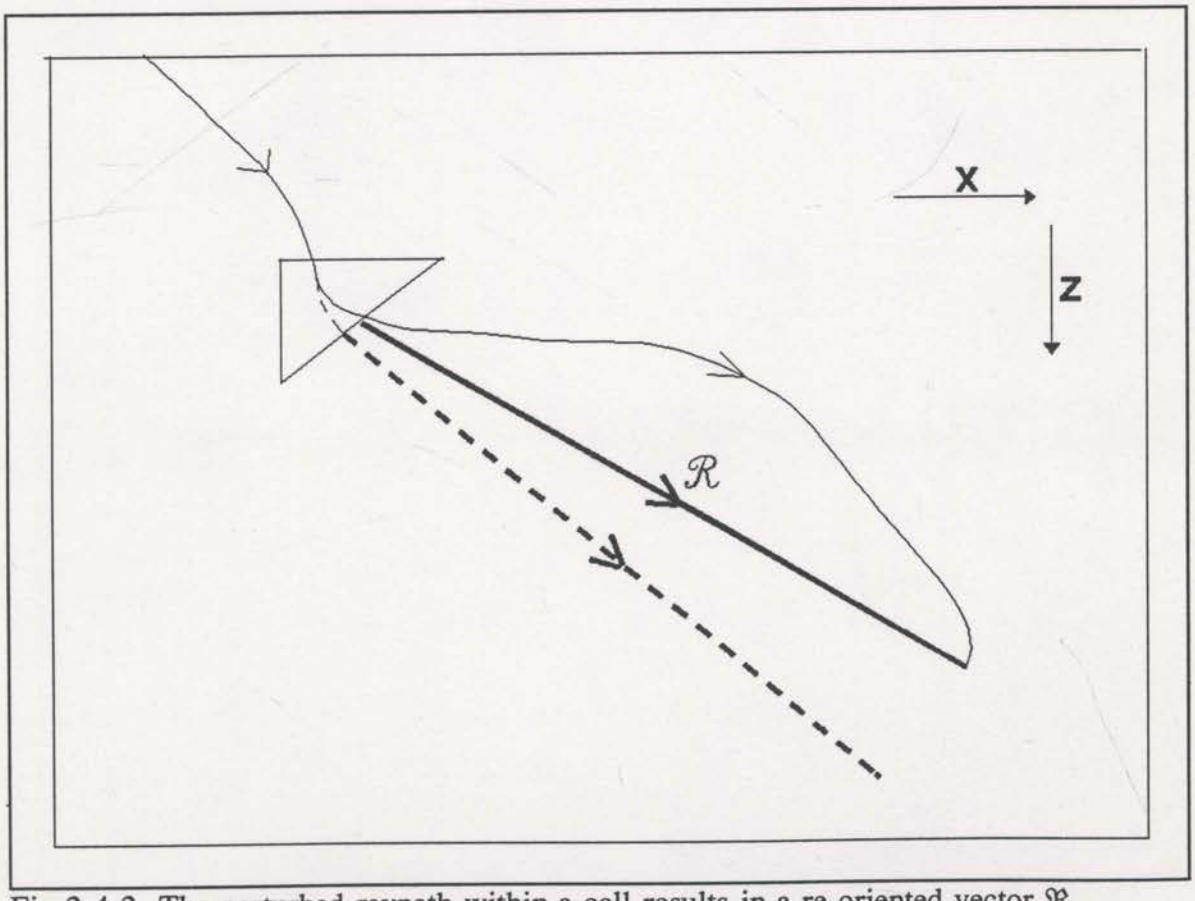


Fig 2.4-2 The perturbed raypath within a cell results in a re-oriented vector $\mathfrak{R}$.

to be computed. Along with the original X_{err}, this value is used to estimate the required Frechet derivative. It is expected that these estimates will be accurate if the perturbation of the subject velocity node is small. A small perturbation also allows each cell of the area of influence to be handled separately with the results added together (as indicated in Figure 2.4-2). If the velocity perturbations are too large, the change in exit location and angle may be large enough to significantly change the shape of the rest of the raypath.

The code written to compute these semi-analytic[2] derivative estimates is called SADE2 and can be found in the appendix. In order to make these estimates, SADE2 requires a number of values to be computed and placed in 'common blocks' for the raytracing subroutine XERRRT (also see the appendix). These values are;

1) upon exit of each cell - the cell number, the exit coordinates, the exit angle and the travel time at the moment of exiting,

2) the final depth (where the total travel time matches the data travel time) - *finalz*,

3) the final x-coordinates of both the source and receiver ends of the ray - *finalx1* and *finalx2*,

4) the rates of change of x-coordinate with respect to depth at the end of each half of the ray - *dxdz1* and *dxdz2*,

5) the rates of change of time with respect to depth at the end of each half of the ray - *dtdz1* and *dtdz2*, and

---

[2] Called semi-analytic because only part of the ray is retraced. The rest is an analytical approximation.

6) the local velocity at the end of each half of the ray - *finalv1* and *finalv2*.

All of this information is used by the SADE2 subroutine for the estimation of the derivative $\partial X_{err}/\partial V$, by re-orienting the vector $\Re$ (see Figure 2.4-2) and computing the new depth at which the X_{err} will be evaluated.

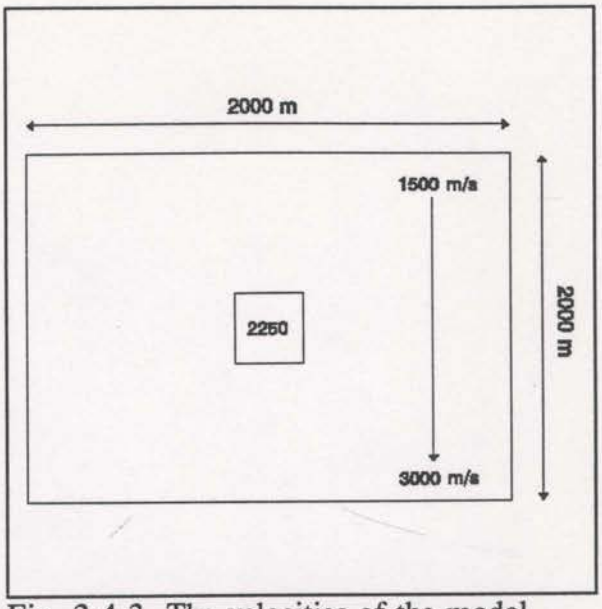


Fig. 2.4-3 The velocities of the model used to check the semi-analytic derivative estimates. The node spacing is 50x50 metres.

A synthetic data set was used to verify the accuracy of these semi-analytic derivative estimates. The interval velocities of this model consist of a linear background increasing from 1500 m/s to 3000 m/s, with a square anomaly of 2250 m/s in the centre (see Figure 2.4-3). Rays were traced though this model with regular source and receiver locations and a range of surface angles (see Figure 2.4-4). The ray tracing resulted in 1312 valid rays (only 184 of these shown in Figure 2.4-4) within this model of 41x41 velocity nodes.

The most concise way to evaluate the semi-analytic derivative estimates is through the gradient of the constraint statistic, ∇C. This gradient is calculated using the Frechet derivatives and is used to drive the inversion of the data (see equation (2.4-2)). The semi-analytic estimates are compared to fully ray traced finite-difference estimates in Figures 2.4-5 and 2.4-6. The finite-difference estimates were computed using a subroutine called FRECHX (see the appendix). Figures 2.4-5 and 2.4-6 are quite similar with one of the major differences being the magnitude of the derivative

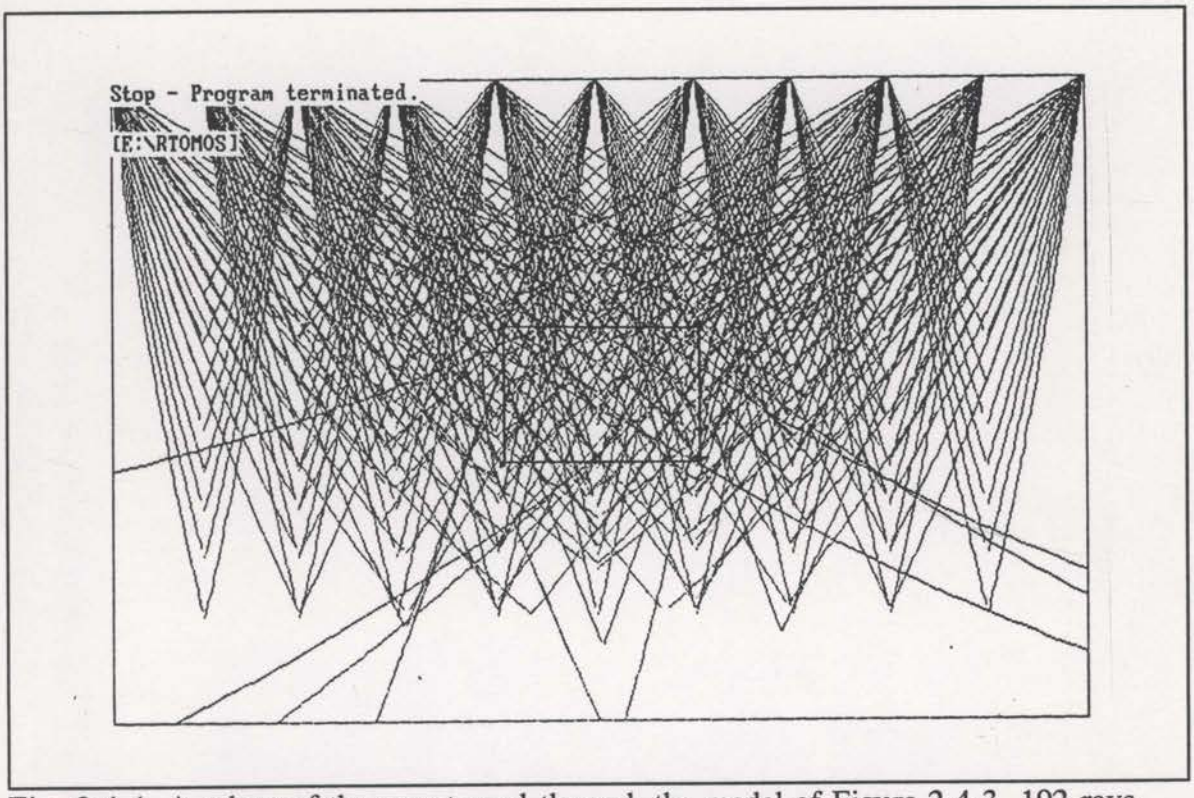


Fig. 2.4-4 A subset of the rays traced through the model of Figure 2.4-3. 192 rays are displayed but only 184 of these were accepted as data.

estimates. The finite-difference estimates (Figure 2.4-5) are generally larger than their semi-analytic counterparts (Figure 2.4-6), especially for the shallower velocity nodes (small velocity node numbers). The semi-analytic estimates appear to be reasonable.

Both the finite-difference and semi-analytic estimates were used to complete an inversion of the synthetic data described above. The final images of these inversions are displayed in Figures 2.4-7 and 2.4-8. Surprisingly, the semi-analytic derivative estimates have resulted in a much improved inversion. The comparison of the inversion results can be found in Table 2.4-1. The semi-analytic derivative estimates have resulted in a smaller final C value, in less iterations. It was expected that the semi-analytic estimates would not be any better than the finite-difference ones; this test was only performed to see how close the semi-analytic estimates were to the finite difference estimates.

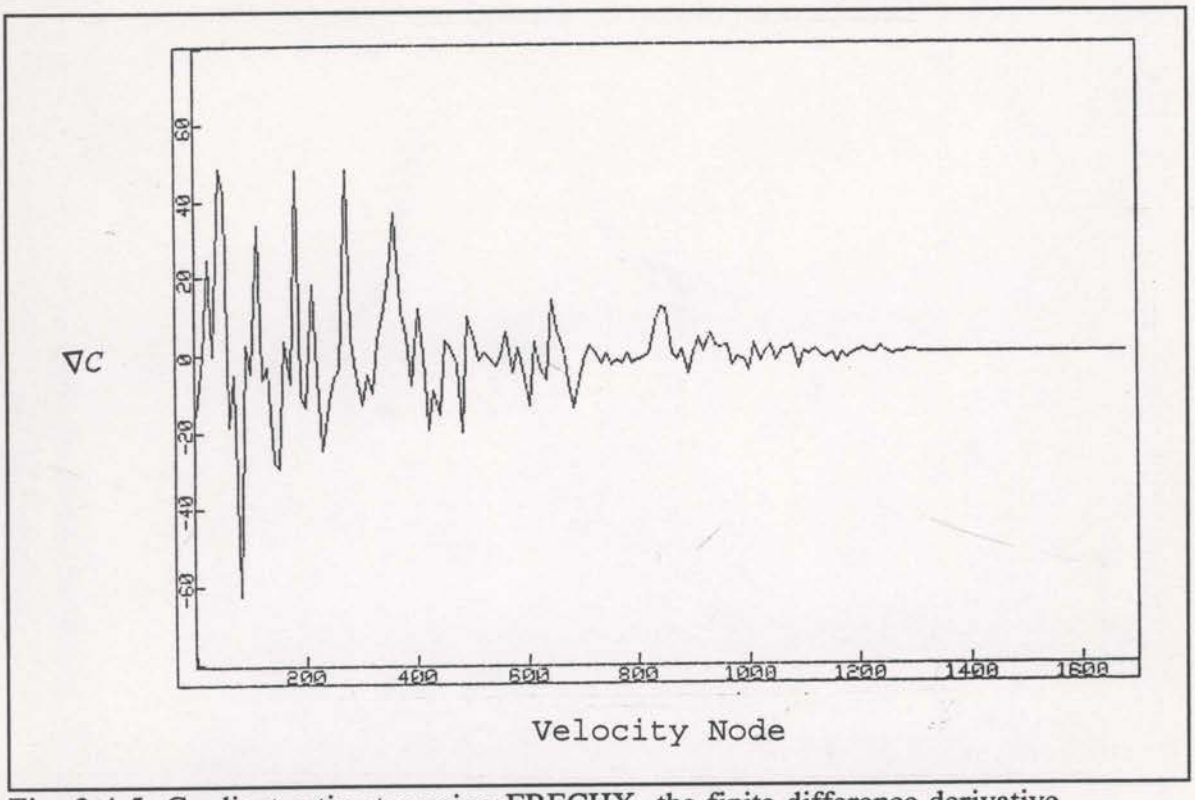


Fig. 2.4-5 Gradient estimates using FRECHX, the finite-difference derivative estimator.

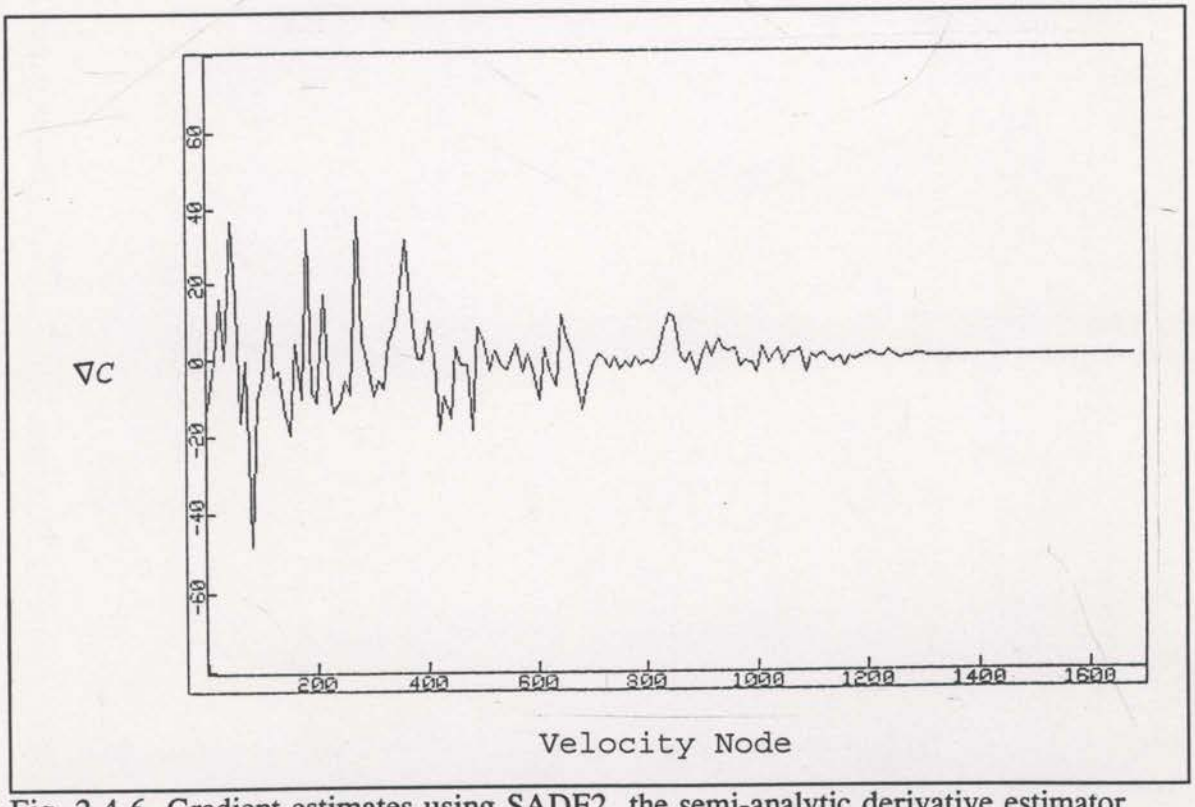


Fig. 2.4-6 Gradient estimates using SADE2, the semi-analytic derivative estimator.

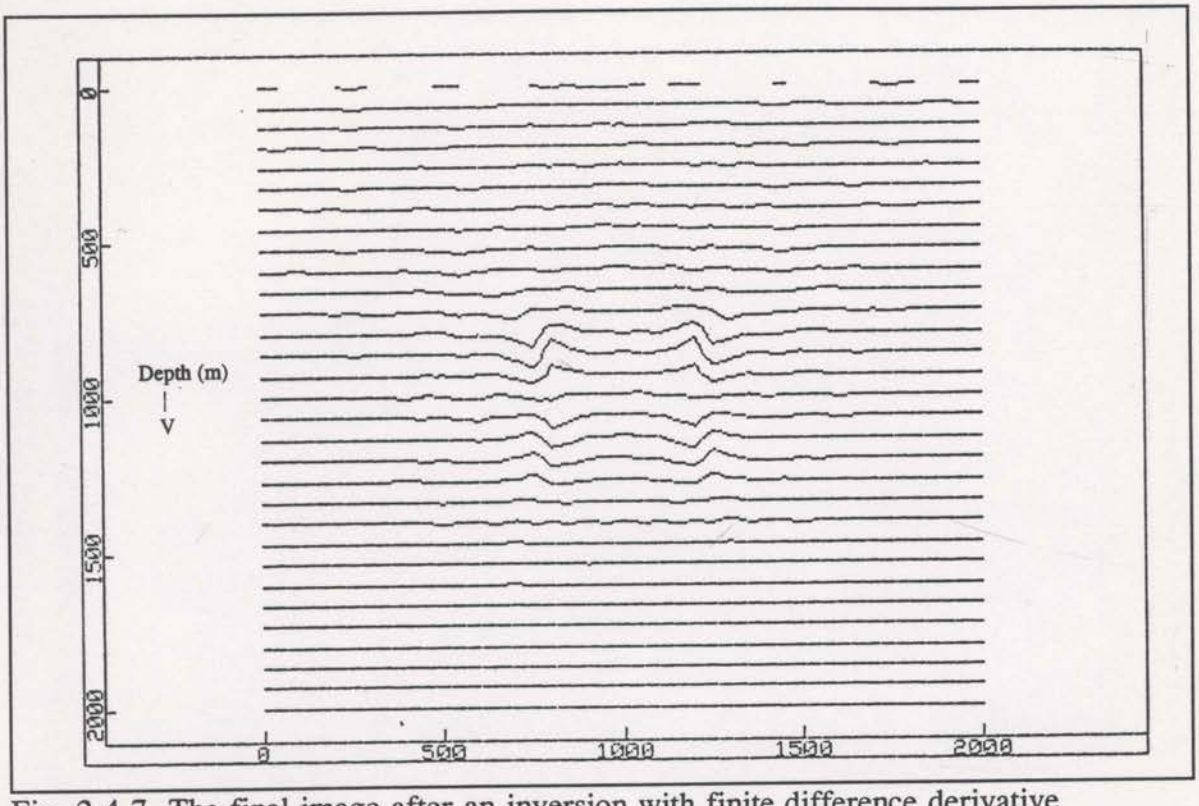


Fig. 2.4-7 The final image after an inversion with finite difference derivative estimates.

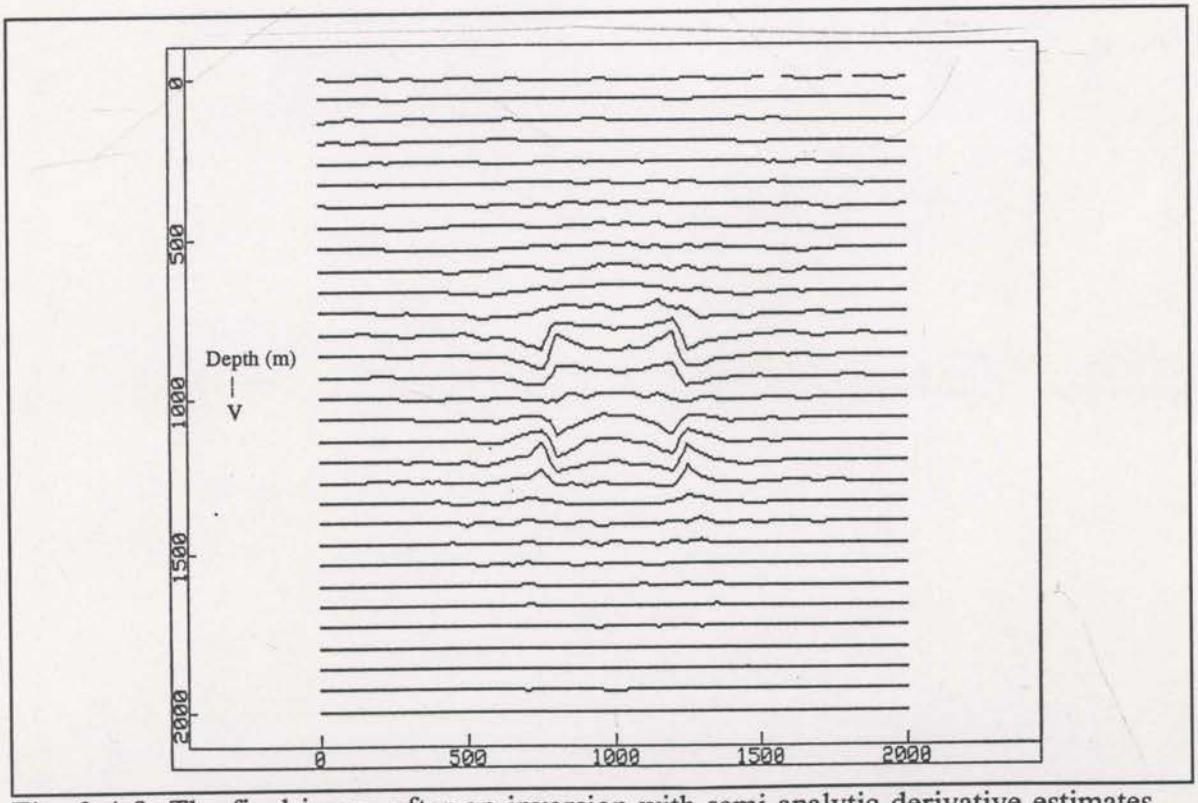


Fig. 2.4-8 The final image after an inversion with semi-analytic derivative estimates (SADE2).

ITERATION	FINITE-DIFFERENCES C Value / Expected C	SEMI-ANALYTIC C Value / Expected C
1	974515 / 809486	974613 / 611467
2	846018 / 684663	611756 / 421692
3	741426 / 601354	421505 / 294016
4	598125 / 477003	292685 / 217063
5	502037 / 428042	206046 / 162009
6	432897 / 376930	159798 / 135241
7	377288 / 275676	130242 / 116015
8	280804 / 251546	117523 / 106245
9	251746 / 214859	103885 / 95554
10	217855 / 206648	97089 / 90108
11	206733 / 197700	89065 / 83378
12	197790 / 191154	84428 / 79377
13	191685 / 188650	78817 / 74132
14	188519 / 180016	75214 / 71153
15	186743 / 178709	71243 / 67595
16	178885 / 165109	68617 / 65491
17	165724 / 161598	65655 / 62735
18	161781 / 156352	63625 / 61045
19	156854 / 154517	61166 / 58806
20	154527 / 145857	59608 / 57480
21	146436 / 143476	57377 / 55394
22	147074 / 144237	56214 / 54403
23	144391 / 140852	54192 / 52495
24	141116 / 139480	53304 / 51730
25	139531 / 136425	51529 / 50060
26	136874 / 135265	50873 / 49499
27	135396 / 133137	
28	133416 / 131460	

Table 2.4-1 The results of the inversion based on semi-analytic derivative estimated compared to those based on finite-difference estimates. The expected C value is that which the inversion predicts will be achieved in the next iteration.

A couple of features have been introduced to the semi-analytic derivative estimator (SADE2) that may affect its performance compared to the finite-difference estimator (FRECHX). The first feature was introduced to guard against problems that may arise if the vector $\mathfrak{R}$ (see Figure 2.4-2) has to be re-oriented too far from its original location. If the change in X_{err} is more than 10 metres, a zero is returned for the derivative involving that particular ray and velocity node. The need for this was rare with the current model, however, ignoring unstable points may have helped stabilise the inversion. Another less important feature has been added to account for the rare possibility of the new exit location of the ray actually being deeper than the required depth for the evaluation of X_{err}. Once again, these occurrences were ignored by returning a zero derivative.

One last problem for the semi-analytic derivative estimator is the possibility of the new exit point from the cell being a long way from the original exit point. Figure 2.4-9 demonstrates one set of circumstances where this could happen. In this case, the ray will now pass through a cell not traversed previously. It is difficult to maintain the assumption that the rest of the raypath can be defined by the vector $\mathfrak{R}$. Once again, this occurrence would be rare but it might be better not to attempt estimates

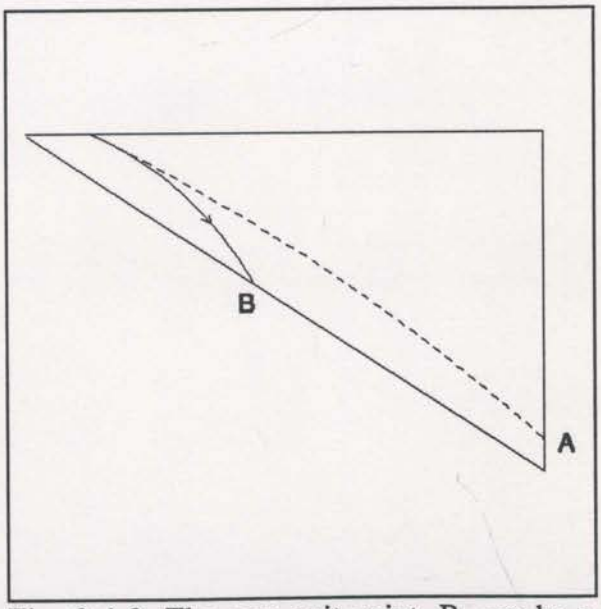


Fig. 2.4-9 The new exit point, B, can be a long way from the original exit point, A. Even to the point of entering a cell not previously traversed.

in these cases.

The motivation for developing the semi-analytic derivative estimator was improved computational efficiency. The intuitive ideas underlying its implementation seem to have worked satisfactorily in practice. The 1312 ray example described earlier used 1700 seconds of CPU time[3] per iteration with finite-difference approximations but only 75 seconds for the semi-analytic estimates. A twenty fold reduction in run time has been achieved. It is interesting to note that of the 75 seconds per iteration, 65 seconds were taken up by ray tracing and derivative estimation - only 10 seconds for the inversion. Further efficiency improvements must come from the ray tracing software. This is a rather small model though; the proportional expense changes dramatically for larger models.

An interesting technique that may also be useful for the current problem is "approximate inverse mapping" (Oldenburg and Ellis, 1991). This uses a relatively simple approximate inverse mapping from the data space to the model space and determines correction vectors to account for the approximation. Linearisation is not necessarily required. This type of algorithm has not been tested within the current work.

---

[3] On an IBM 3090 mainframe computer, with vector facility.

2.5 Stages of decreasing smoothing.

Once the traveltime data has been obtained, an initial velocity model is required for the commencement of the iterative inversion of these data. There is rarely full confidence in the initial velocity model. Usually some aspects of the model cause concern, for example, the best velocity for a particular layer that is assumed to be homogeneous. With such concerns, it seems inconsistent for the inversion process to consider individual changes to each of the closely spaced, discrete velocity values. It would seem more appropriate to change the velocities of complete geologic units initially, and only later allow for higher spatial frequency updates. This would be tantamount to using the data to improve the starting model before allowing the standard inversion to begin.

Consider a synthetic velocity model consisting of an 11x11 grid of velocity values. The velocities assigned to this grid are 2000 m/s except for the 3x3 block of velocities in the centre which are 2050 m/s. That is, the model has a fast square shaped anomaly in the centre. A total of 192 raypaths were traced though this model to generate the traveltime data to be inverted. These raypaths were generated to symmetrically span the horizontal extent of the model with a uniform range of short to long offsets, along with a uniform range of surface angles to ensure good coverage at all depths. An initial constant velocity model of 2000 m/s was used to begin the inversion that resulted in the image of Figure 2.5-1. This inversion did not successfully uncover the anomaly.

A potential problem with an inversion as described above is that the initially

2021	2000	1991	1998	2003	2010	2008	2003	1993	1998	2010
2009	2005	1989	2004	2007	2008	2011	2002	1988	2007	2006
2002	2023	1984	1992	2012	2008	2024	1997	1991	2006	2002
2000	2014	2003	1976	2007	2008	2029	1992	1997	2008	2000
2000	2005	2020	1956	2024	2000	2016	1977	2008	2003	2000
2000	2003	2022	1991	1988	1989	1981	1987	2012	2001	2000
2000	2001	2013	2012	1956	1979	1963	1993	2009	2000	2000
2000	2001	2007	2022	1962	1960	1954	1987	2005	2000	2000
2000	2000	2003	2009	1985	1944	1958	1992	2003	2000	2000
2000	2000	2000	2000	1995	1969	1984	1998	2000	2000	2000
2000	2000	2000	2000	2000	1996	1999	2000	2000	2000	2000

Fig. 2.5-1 An inversion of the 11x11 synthetic model data. The initial model was 2000 m/s for every grid node. The values within the rectangle should be 2050 m/s.

(See the footnote on page 15)

traced raypaths may, in some cases, be far from the actual raypaths. This is because the anomaly is not defined by the initial model. Figure 2.5-2 demonstrates how the traced and actual ray locations can differ appreciably. It makes little sense to attempt complete correction of the velocity model based on such incorrect ray locations. An effort must be made to improve the ray locations before the final solution can be derived. As mentioned earlier in this section, the velocities of whole geologic units could be changed until the best simple velocity model is found. Such a new simple model should position the rays closer to their actual location and the high-fidelity inversion can be allowed to proceed with more confidence. Effectively, the traveltime data are used to improve the initial velocity model. However, as with the current synthetic example, the existence of geologic units (in this case, a square region of fast velocity) is not always known. It is desirable to find a procedure that will help improve the ray locations and is independent of the velocity field.

The standard inversion will consider changes to each discrete velocity value independently. Intuitively, high spatial frequency velocity updates will be of little use if

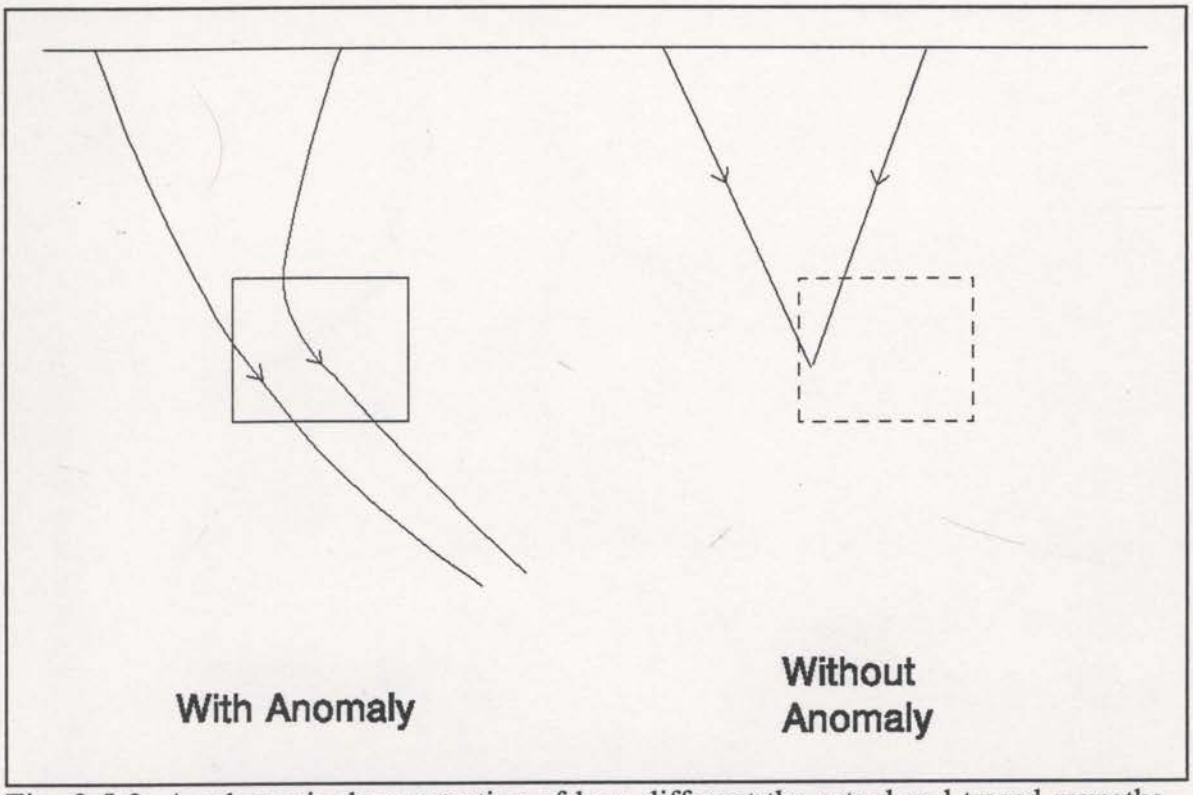


Fig. 2.5-2 A schematic demonstration of how different the actual and traced raypaths can be when the anomaly is not defined in the initial velocity model.

the rays are not near their correct locations. High spatial frequency changes will have little effect on the location of the rays and it will be difficult for the iterative inversion to approach the correct solution with such changes. Low spatial frequency, or smooth, velocity changes will have more effect on the locations of the rays. One can imagine an algorithm that allows only smooth velocity changes at first, effectively locating the raypaths, and only then allows higher frequency changes.

One such algorithm consists of blocking the velocity values for the early iterations (this type of procedure has been described by Williamson, 1990). Consider the 11x11 synthetic model described earlier. If it is extended by an extra row and column to a discrete 12x12 model, then 4x4 groups of velocity nodes can be "blocked" together for the early iterations of the inversion. Each of the sixteen velocities within a block will be changed equally. The sensitivities will be computed for velocity changes

of the block as a whole. The inversion initially becomes a 3x3 problem in which each cell consists of 16 sub-cells. The initial model will still be a uniform velocity of 2000 m/s, but the early blocked iterations effectively find a better low frequency model that will define the ray locations more accurately.

After nine iterations of such a blocked procedure, the image of Figure 2.5-3 results. Notice that each of the 4x4 blocks have the same velocity. The block containing the anomaly now has a velocity of 2037 m/s which is approaching the true velocity of the anomaly, 2050 m/s. Blocking has produced a noticeable improvement over the unblocked inversion (Figure 2.5-1) even for this simple model. The raypaths generated in the blocked inversion should be better located. A further eight iterations were performed without any blocking (but with the model shown in Figure 2.5-3 as the starting model), the results of which can be seen in Figure 2.5-4. The velocity values within the marked anomalous zone are close to the required 2050 m/s but the residual effect of the blocked zone (dashed line) is still apparent. This example has shown that it can be very beneficial to attempt to correctly locate the rays before proceeding with the inversion. However, there is also evidence that the blocking algorithm may not be the best approach.

Williamson (1990) also explored this approach of blocking but referred to it as a variable parameterisation scale-length technique. His original motivation was to force the inverse problem to be temporarily over-parameterised, and hence stable. After noticing similar residual velocity problems as those demonstrated by Figure 2.5-4, Williamson developed a translation invariant approach with the same benefits as the blocking approach. He introduced a model covariance matrix to effect smoothing of the constraint statistic, C. The smoothed constraint statistic for the current problem can be

1986	1986	1986	1986	1977	1977	1977	1977	1972	1972	1972	1972
1986	1986	1986	1986	1977	1977	1977	1977	1972	1972	1972	1972
1986	1986	1986	1986	1977	1977	1977	1977	1972	1972	1972	1972
1986	1986	1986	1986	1977	1977	1977	1977	1972	1972	1972	1972
1986	1986	1986	1986	2037	2037	2037	2037	2002	2002	2002	2002
1986	1986	1986	1986	2037	2037	2037	2037	2002	2002	2002	2002
1986	1986	1986	1986	2037	2037	2037	2037	2002	2002	2002	2002
1986	1986	1986	1986	2037	2037	2037	2037	2002	2002	2002	2002
2000	2000	2000	2000	1987	1987	1987	1987	2002	2002	2002	2002
2000	2000	2000	2000	1987	1987	1987	1987	2002	2002	2002	2002
2000	2000	2000	2000	1987	1987	1987	1987	2002	2002	2002	2002
2000	2000	2000	2000	1987	1987	1987	1987	2002	2002	2002	2002

Fig. 2.5-3 After 9 iterations of the "blocked" inversion procedure, the anomaly is better defined. The anomalous zone is marked by the solid line.

1990	1989	1989	1988	1980	1979	1980	1980	1975	1975	1974	1972
1987	1989	1989	1990	1981	1981	1982	1981	1975	1975	1973	1972
1986	1988	1988	1989	1980	1981	1982	1981	1974	1974	1972	1972
1986	1987	1988	1988	1980	1984	1981	1978	1973	1973	1972	1972
1986	1987	1988	1988	2046	2037	2049	2039	2003	2002	2002	2002
1986	1987	1987	1985	2040	2038	2037	2037	2003	2002	2002	2002
1986	1986	1987	1987	2034	2037	2036	2037	2003	2002	2002	2002
1986	1986	1986	1997	2029	2040	2048	2039	2002	2002	2002	2002
2000	2000	2000	2000	1978	1996	1988	1984	2002	2002	2002	2002
2000	2000	2000	2000	1987	1987	1987	1987	2002	2002	2002	2002
2000	2000	2000	2000	1987	1987	1987	1987	2002	2002	2002	2002
2000	2000	2000	2000	1987	1987	1987	1987	2002	2002	2002	2002

Fig. 2.5-4 A further eight iterations with no blocking does not remove the effect of the earlier blocked iterations. The anomalous zone is marked with a solid line. The blocked zone is marked with a dashed line.

expressed as,

$$C = \mathbf{X}_{err}^{T}\mathbf{C}_{d}^{-1}\mathbf{X}_{err} + (\mathbf{v}-\mathbf{v}_0)^{T}\mathbf{C}_{m}^{-1}(\mathbf{v}-\mathbf{v}_0) \qquad (2.5\text{-}1)$$

where $\mathbf{X}_{err}$ is the column vector of $X_{err(i)}$ values for each ray, $\mathbf{C}_d$ is the data covariance matrix (currently assumed to be the identity matrix), $\mathbf{v}$ are the new velocities derived by this iteration, $\mathbf{v}_0$ are the current velocities and $\mathbf{C}_m$ is the model covariance matrix. Williamson (1990) added two-dimensional Gaussian shaped functions to the model covariance matrix, effectively forcing the velocity updates made by each iteration to be as smooth as these functions. This approach treats every velocity value equally. The velocities are not grouped arbitrarily. Smooth changes will be enforced with limited side-effects.

The new constraint statistic of equation (2.5-1) was adopted in the inversion process described in this thesis. The first and second derivatives of C will be needed for this purpose. The first derivative is given by the gradient vector,

$$\nabla C = 2\nabla\mathbf{X}_{err}^{T}\mathbf{X}_{err} + 2\mathbf{C}_{m}^{-1}(\mathbf{v}-\mathbf{v}_0) \quad , \qquad (2.5\text{-}2)$$

and the corresponding second derivative is,

$$\nabla\nabla C = 2\nabla\mathbf{X}_{err}^{T}\nabla\mathbf{X}_{err} + 2\nabla\nabla\mathbf{X}_{err}^{T}\mathbf{X}_{err} + 2\mathbf{C}_{m}^{-1} \quad . \qquad (2.5\text{-}3)$$

Each of equations (2.5-1), (2.5-2) and (2.5-3) are identical to the conventional expressions except for their final terms. In each case, the last term suggests that it may be necessary to calculate the inverse of $\mathbf{C}_m$. However, this computational expense is avoided since the above vectors are in a space dual to the model space. The dual space is the space of all linear forms of the original space (see Tarantola, 1987, section

4.4.2). Here, the above vectors are in the dual space by the action of $\mathbf{C}_m$. To obtain the vector locations in the model space, the above forms must be operated on by $\mathbf{C}_m$ (see Tarantola, 1987, equation (4-68)). Therefore,

$$\nabla C' = 2\mathbf{C}_m \nabla \mathbf{X}_{err}^T \mathbf{X}_{err} + 2(\mathbf{v}-\mathbf{v}_0) \tag{2.5-4}$$

and,

$$\nabla\nabla C' = 2\mathbf{C}_m \nabla \mathbf{X}_{err}^T \nabla \mathbf{X}_{err} \mathbf{C}_m + 2\mathbf{C}_m \nabla\nabla \mathbf{X}_{err}^T \mathbf{X}_{err} \mathbf{C}_m + 2\mathbf{C}_m \,, \tag{2.5-5}$$

(the symmetry of the covariance matrices has been used here). The second term of equation (2.5-5) is usually ignored (see Tarantola, 1987 (page 194), and Kennett et al., 1988) because $\nabla\nabla \mathbf{X}_{err}$ diminishes as the data fit improves and not least because it is also difficult to compute. The actual equation used is then,

$$\nabla\nabla C' = 2\mathbf{C}_m \nabla \mathbf{X}_{err}^T \nabla \mathbf{X}_{err} \mathbf{C}_m + 2\mathbf{C}_m \quad . \tag{2.5-6}$$

The Gaussian functions used by Williamson are characterised by a half-width, L. The value of the function at a distance d from the subject point is then,

$$g = \exp\left[-\frac{1}{2}\left(\frac{d}{L}\right)^2\right] \quad . \tag{2.5-7}$$

This function is calculated by the subroutine GAUSCAL3 (see the appendix) and the results are stored in a compressed format before the commencement of the inversion. Most of the work involved with smoothing application is controlled by the group of subroutines that perform the inversion and which are driven by the subroutine MEMNL$8 (see the appendix)[1].

[1] For more on MEMNL$8, see section 4.

Instead of arbitrarily blocking groups of velocities, Williamson (1990) suggested that the early iterations be performed with Gaussian smoothers that have a half-width of the order of the dimensions of the model. Once these iterations make no further C reductions, Williamson reduced the half-width by a factor of two and allowed the next stage of the inversion to proceed. This approach is continued until no smoothing at all is applied. The result is to initially extract only the very low frequency features of the model and then slowly extract higher and higher frequency features. The effect of this algorithm is to impose a bias towards positioning of the raypaths early in the inversion and then gradually allowing the details of the velocity anomalies to be extracted.

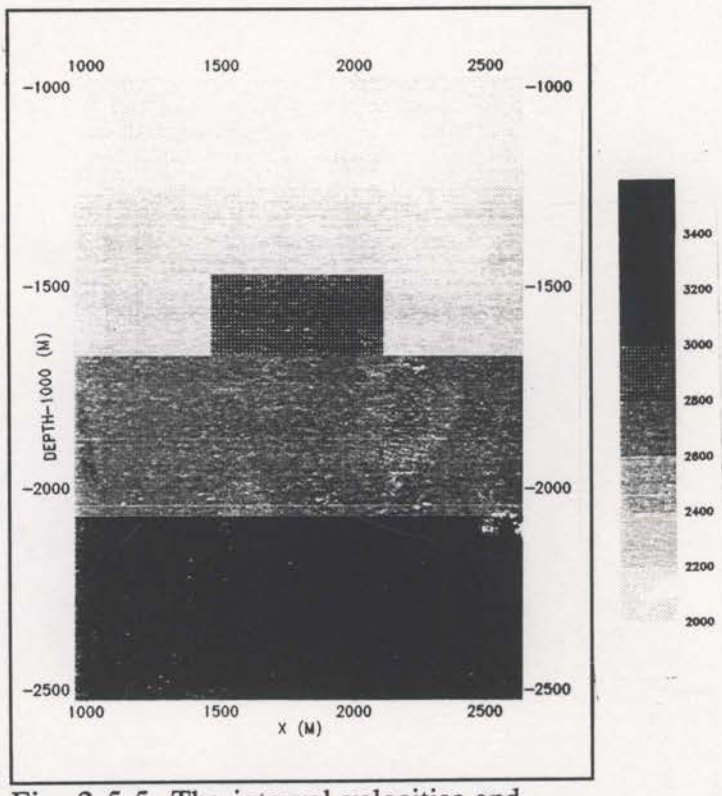


Fig. 2.5-5 The interval velocities and dimensions of the model used to create the synthetic data.

It is useful to evaluate this technique on some synthetic data. The interval velocities and dimensions of the model to be used are shown in Figure 2.5-5. A random number generator was used to randomly select source and receiver locations as well as source and receiver surface angles, for a total of 800 rays. These rays were traced through the model to derive the traveltime data[2]. The velocities were defined on

[2] Many of these rays exited the boundaries of the model. These were ignored and another ray traced until 800 valid rays were left.

a 50x50 metre grid. A starting velocity model was prepared with the correct velocity for the first layer (ie. 2000 m/s) and with a linear velocity increase of 0.9 m/s per metre from the base of this layer (see Figure 2.5-6). This starting model implies almost total ignorance of the model. In general, if no other information is at hand, a simple linear increase with depth would be used as the starting model.

An inversion of these data was performed using the stages of decreasing smoothing approach described earlier. Figures 2.5-7 through 2.5-12 display the inverted image after each of the stages of smoothing. The respective Gaussian half-widths are 1600, 800, 400, 200 and 100 metres with the final stage having no smoothing. The features of the model are gradually extracted, with the final image (Figure 2.5-12) displaying a good expression of the anomaly[3], especially considering the poor starting velocity model. Fig 2.5-13 shows a graph of the decrease of the constraint statistic during the 201 iterations required to achieve the final result. The effect of the stages of decreasing smoothing is clear. The smoothing width is reduced when the value of C stops decreasing. This gives Figure 2.5-13 its unusual step-like appearance.

The same data were also inverted without any stages of decreasing smoothing. Figure 2.5-14 shows the final image of this inversion. It is difficult to see an expression of the velocity anomaly in these results. A graph of the change in constraint statistic, C, is shown in Figure 2.5-15. Notice that the final C value is almost double that of the inversion with stages of smoothing (Fig. 2.5-13). The inversion without smoothing has undoubtedly encountered a local minimum of C and converged onto

[3] This anomaly has a velocity that is 25% higher than the velocity of its host layer.

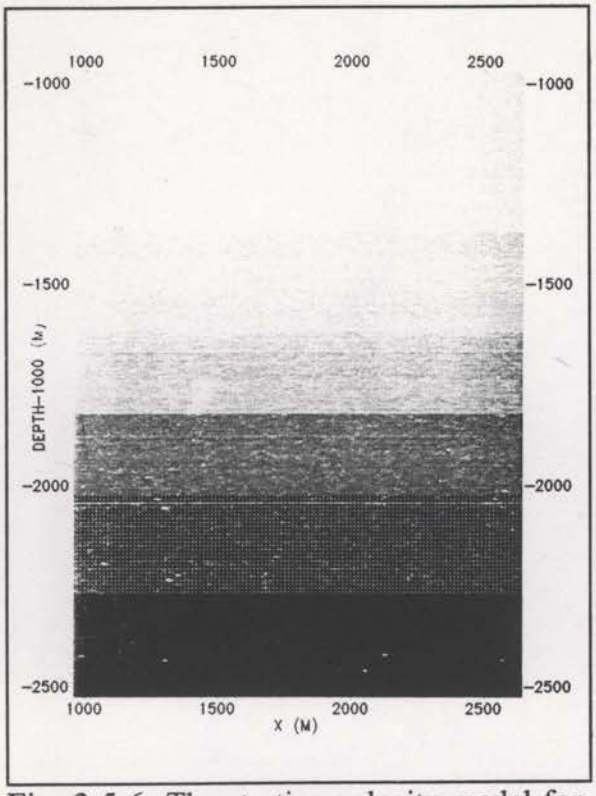


Fig. 2.5-6 The starting velocity model for the inversion of the synthetic data derived from the model of Figure 2.5-5

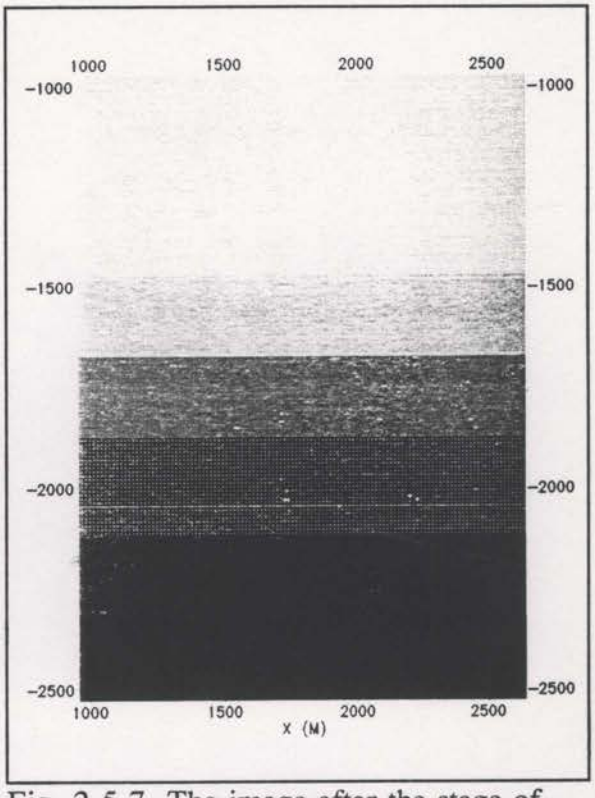


Fig. 2.5-7 The image after the stage of the inversion that used Gaussian smoothers with 1600 metre half-widths.

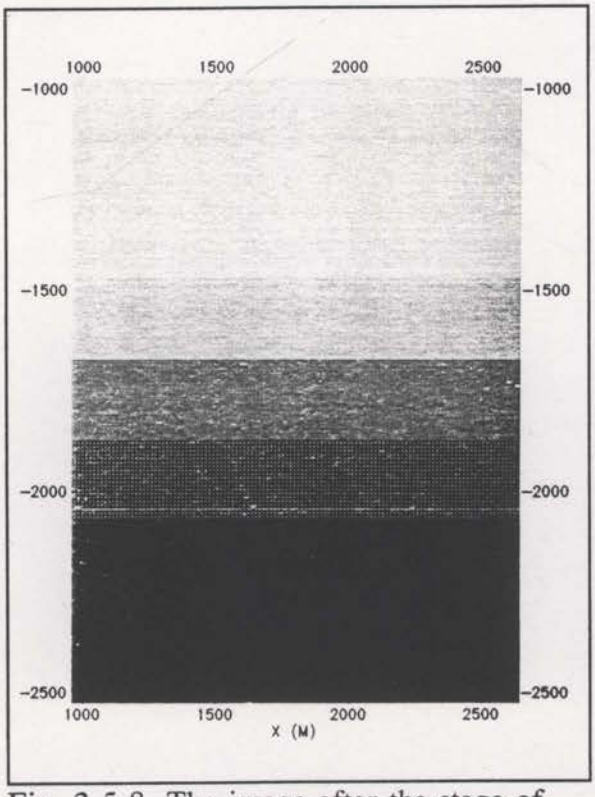


Fig. 2.5-8 The image after the stage of the inversion that used Gaussian smoothers with 800 metre half-widths.

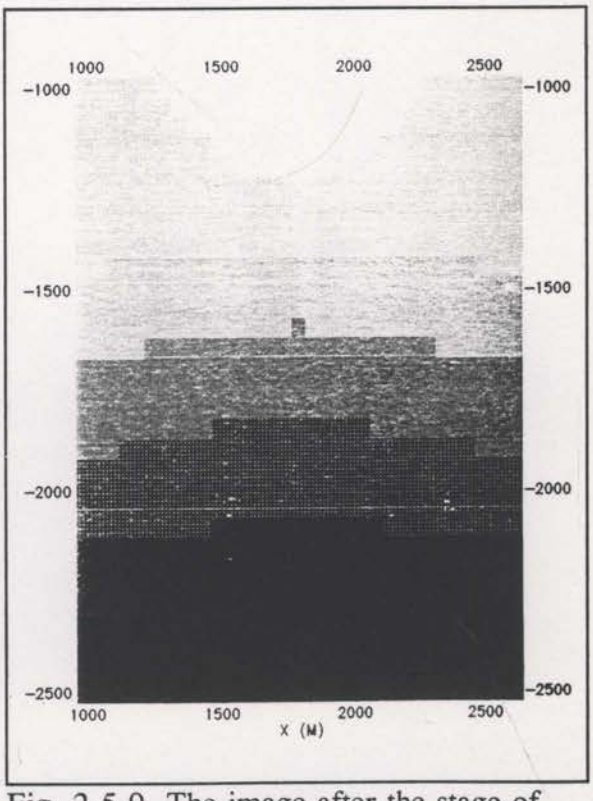


Fig. 2.5-9 The image after the stage of the inversion that used Gaussian smoothers with 400 metre half-widths.

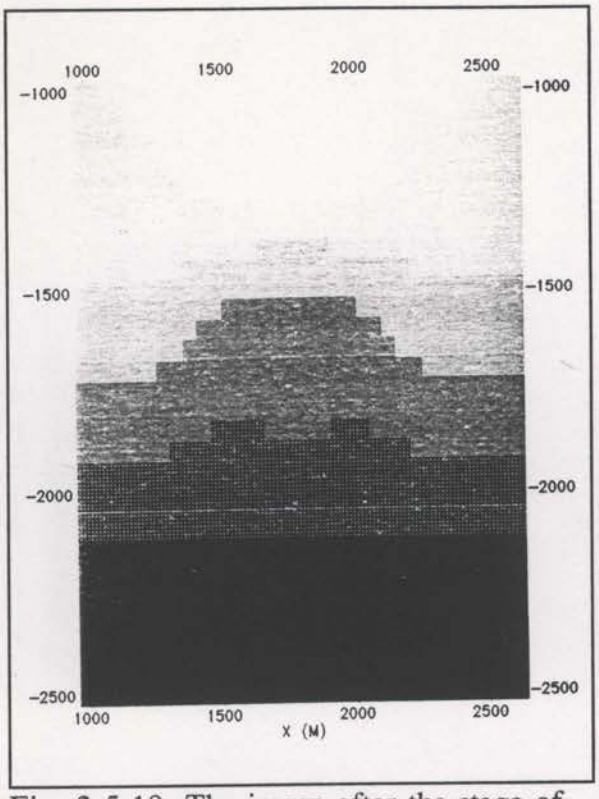


Fig. 2.5-10 The image after the stage of the inversion that used Gaussian smoothers with 200 metre half-widths.

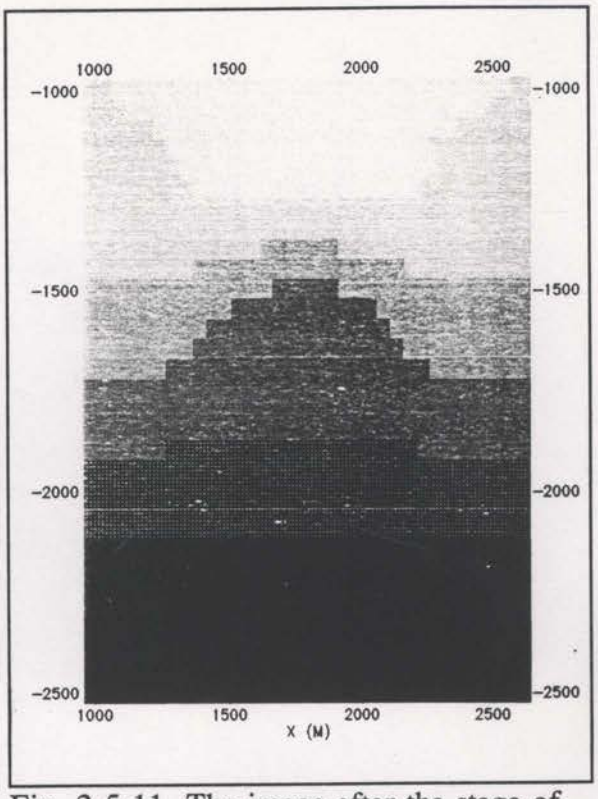


Fig. 2.5-11 The image after the stage of the inversion that used Gaussian smoothers with 100 metre half-widths.

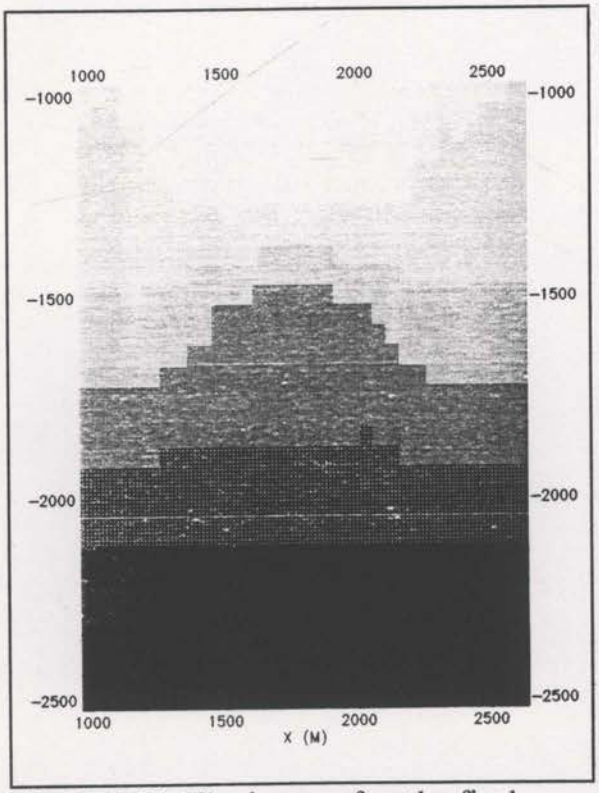


Fig. 2.5-12 The image after the final stage of the inversion that free from smoothing.

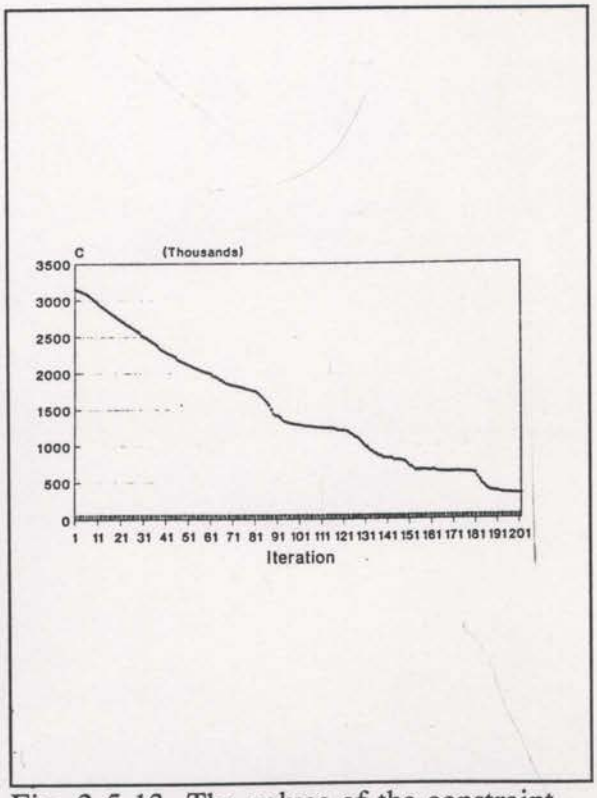


Fig. 2.5-13 The values of the constraint statistic. The stepped appearance is caused by the stages of decreasing smoothing.

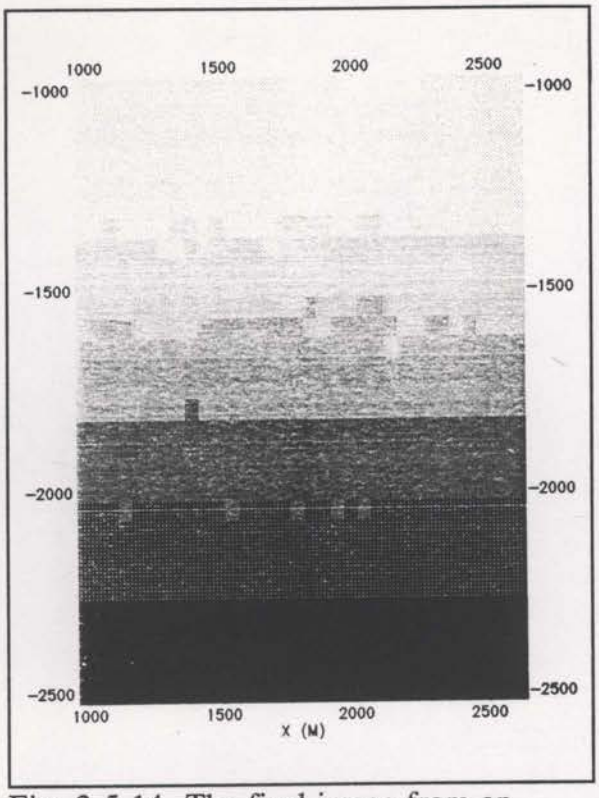


Fig. 2.5-14 The final image from an inversion without smoothing stages.

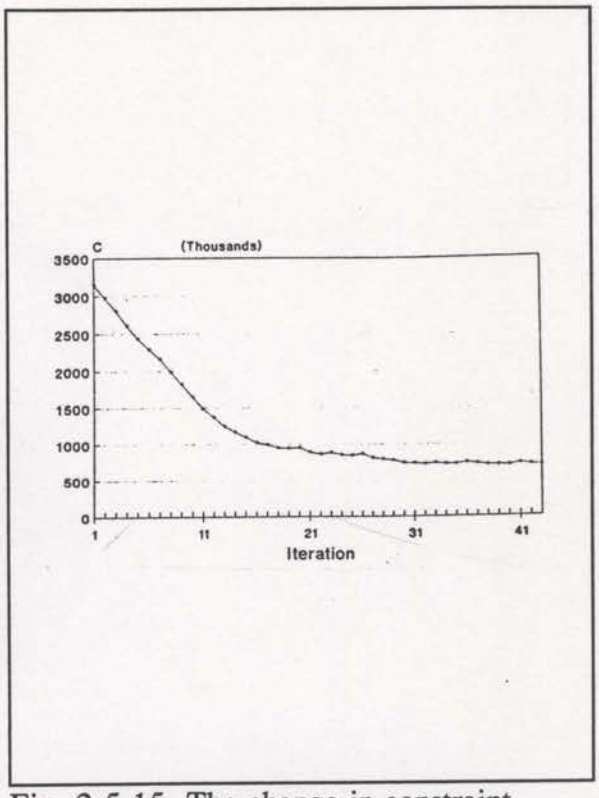


Fig. 2.5-15 The change in constraint statistic values for the inversion without stages of smoothing.

that solution.

The inversion of the traveltime data consists of finding the velocity model that reduces C to its lowest possible value. This is complicated because C is an n-dimensional[4] non-linear function. The most efficient way to search for the minimum is to make local linear (or quadratic) approximations to this non-linear function, and iteratively make small steps of decreasing C values. The problem with following a path of descent is that local minima of C can trap the inversion and prevent it from finding the global minimum (schematically shown in Figure 2.5-16).

When stages of smoothing are used in the inversion, the constraint statistic, C-surface is smoothed during the early stages (recall equation 2.5-1). This has the

[4] 'n' is the number of velocity nodes in the grid.

potential to smooth out local minima temporarily (Figure 2.5-17) and then, as the smoothing decreases, focus on the global minimum. This schematical representation can also be used to demonstrate circumstances where the effective smoothing of C may not be successful. Consider Figures 2.5-18 and 2.5-19. In the first case a residual local minimum remains after smoothing and the inversion is still trapped. Heavier smoothing would have been more appropriate. In the second case (Figure 2.5-19) the inversion actually moves away from the global minimum - less smoothing may reverse this trend. These examples demonstrate that ideal smoothing is data dependent and difficult to define. With the assumption that the constraint surface is reasonably stable near the global minimum and that the starting model is not too far from the actual model, then the technique of stages of decreasing smoothing, as described by Williamson (1990) and discussed above, will be generally successful.

Jupp and Vozoff's (1975) paper provides the theoretical justification of the technique of stages of decreasing smoothing. A short review of their work is useful here.

In terms of the general inversion problem, let $\mathbf{x}$ represent the model parameters, $\mathbf{d}$ the data values and $\mathbf{g(x)}$ the modelled data values. The inversion procedure attempts to minimise,

$$(\mathbf{d} - \mathbf{g(x)})^2 \quad .$$

This is the general constraint statistic. The truncated Taylor series expansion gives,

$$\mathbf{g(x+\delta x) = g(x) + J\delta x}$$

where, $J_{ij} = \dfrac{\partial g_i}{\partial x_j}$ are the Frechet derivatives.

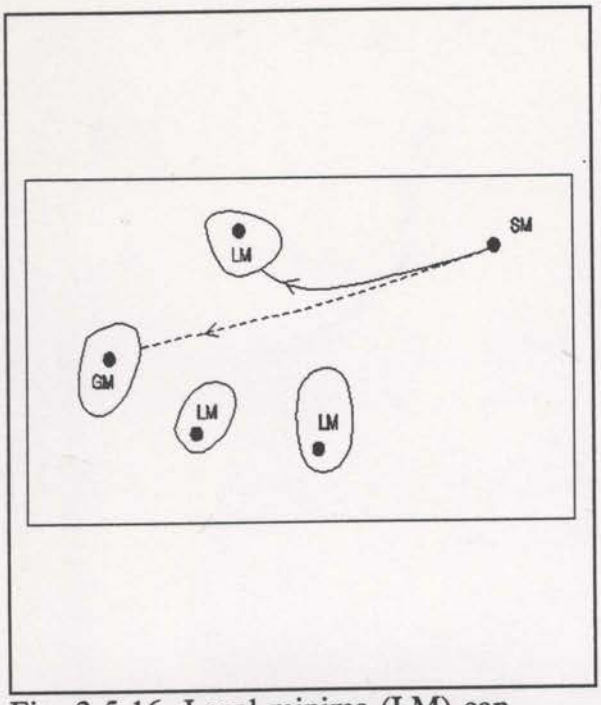


Fig. 2.5-16 Local minima (LM) can prevent the inversion path from reaching the global minimum (GM). SM is the location of the starting model in this space.

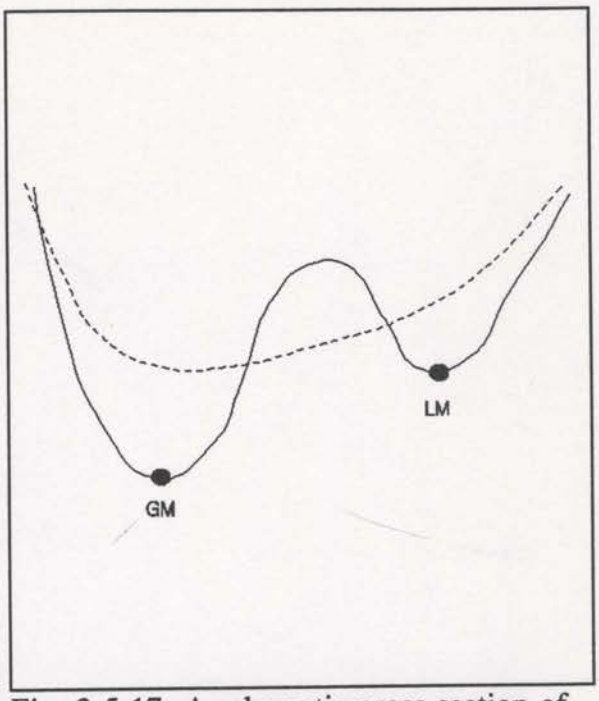


Fig. 2.5-17 A schematic cross-section of the constraint statistic surface. Heavy smoothing in early stages can smooth out local minima (LM) while later stages allow focussing on the global minimum (GM).

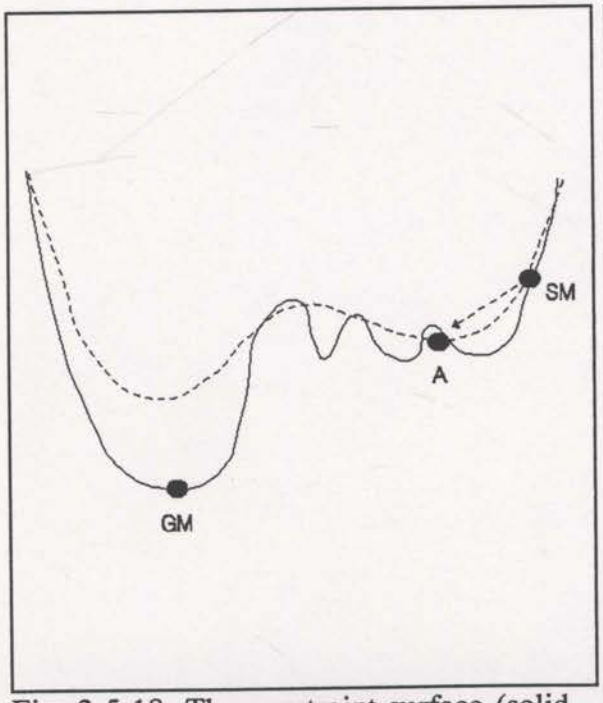


Fig. 2.5-18 The constraint surface (solid line) is smoothed (dashed line) but the starting model (SM) is still trapped be a residual local minimum at point A. Heavier smoothing is required.

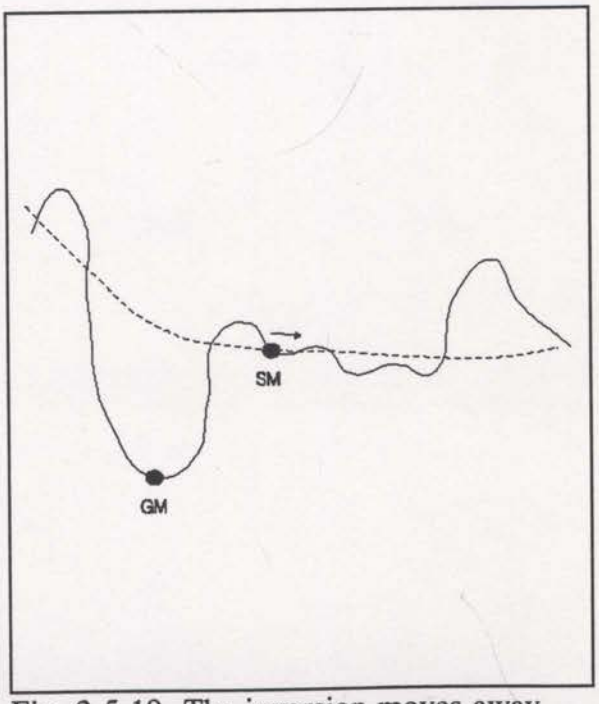


Fig. 2.5-19 The inversion moves away from the global minimum (GM) even after smoothing (dashed line). Less smoothing may be more appropriate.

This equation can also be expressed as,

$$\delta \mathbf{g} \sim \mathbf{J} \delta \mathbf{x} \quad .$$

Using the singular value decomposition (SVD) of $\mathbf{J}$, Jupp and Vozoff (1975) show that the components of the model that have small singular values have very little effect on $\delta \mathbf{g}$. Changes of these components will have little effect on the constraint statistic. Such components of the model are unstable.

The simplest way to avoid problems with small singular value components of the model is to specify a singular value threshold. Changes of components with singular values below the threshold are not considered. Jupp and Vozoff define this as the truncation strategy. Rather than employ such an abrupt cutoff, Jupp and Vozoff also considered a more gentle 'damping' of the small singular value components. They claim that the Marquardt method (1963) is one such damping technique but generalise this to a parametric damping function that encompasses both the Truncation and Marquardt approaches, along with others.

Near the end of their paper, Jupp and Vozoff actually suggest a practical strategy of raising the threshold for the beginning of the inversion "so that only the basic features of the model will be resolved". Reducing this threshold as the inversion proceeds seems to be closely related to Williamson's stages of decreasing smoothing approach. As Jupp and Vozoff suggested, if the components with large singular values are the smoother, or more basic, components, then the SVD and stages of smoothing techniques are the same in principle. It can be concluded that the stages of smoothing strategy is effectively damping the unstable components of the model.

Marquardt's (1963) paper contains an excellent description of the ideas that led

him to develop his damped inversion. He did not use singular values, but Jupp and Vozoff (1975) proved that his method is equivalent to a particular type of damping applied to the singular values. Marquardt noticed that the method of steepest descent made very good progress at the beginning of an inversion but converged slowly at the end, whereas the Gauss-Newton method may be slow at the start but converges rapidly at the end. In an effort to try and get the benefits of both procedures, Marquardt added a constant to the diagonal of the Frechet derivative matrix that resulted from the truncated Taylor series expansion (see above),

$$\delta \mathbf{g} = (\mathbf{J} + \lambda \mathbf{I})\, \delta \mathbf{x}$$

and showed that with $\lambda = \infty$ one get the steepest descent algorithm and with $\lambda = 0$, the Gauss-Newton algorithm. By varying λ as the inversion proceeds, one can change gradually from the steepest descent algorithm to the Gauss-Newton algorithm. Jupp and Vozoff (1975) showed that this is equivalent to their damped singular value approach.

2.6 Weighted least-squares.

When the surface angles are picked from real seismic data (see section 2.3) some mispicks are unavoidable. These mispicks can result in some very large X_{err} values even with an accurate velocity model. Even without mispicking, some rays can be very sensitive and have large X_{err}'s for quite accurate models. Since the constraint statistic squares the X_{err} values, a few such large X_{err}'s can stifle the inversion process. In general, rays with smaller X_{err}'s are more likely to be reliable. It is reasonable to apply a weighting to the data to bias the inversion toward rays with smaller X_{err}'s.

Recall the expression for the constraint statistic (equation 2.5-1),

$$C = \mathbf{X}_{\mathbf{err}}^{\mathbf{T}} \mathbf{C}_{\mathbf{d}}^{-1} \mathbf{X}_{\mathbf{err}} \quad , \tag{2.6-1}$$

but here ignoring the additive smoothing term. If the data covariance matrix is diagonal, this function can be expressed in suffix notation as,

$$C = \sum_i \frac{X_{err(i)}^2}{\sigma_i^2} \quad , \tag{2.6-2}$$

where σ_i^2 is the variance associated with the i th data value. From the above discussion it is reasonable to assume that there is a relationship between σ_i^2 and the magnitude of $X_{err(i)}$. For the purposes of the current study, it is simpler to assign weights, w_i, instead of variances σ_i^2, such that,

$$C = \sum_i X_{err(i)}^2 w_i \qquad (2.6\text{-}3)$$

with $w_i = 1/\sigma_i^2$. These weights are to be chosen such that the inversion process is biased toward rays with smaller X_{err}'s.

After experimentation, the following modified Hanning function was used as a satisfactory specification for the weights,

$$\begin{aligned} w_i &= \left[\frac{3}{4} + \frac{1}{4}\cos\left(\frac{\pi X_{err(i)}}{X_{ref}}\right)\right]^2 \qquad \text{for } X_{err(i)} < X_{ref} \\ &= 0.25 \qquad\qquad\qquad\qquad \text{for } X_{err(i)} \geq X_{ref} \,. \end{aligned} \qquad (2.6\text{-}4)$$

The parameter X_{ref} is used to control the extent of the weighting. The value of X_{ref} was originally set to somewhat arbitrary 300 metres.

The incorporation of weights into the inversion process initially seemed straightforward. However, with the weights as defined in equation (2.6-4), there will be changes in the weights between iterations. Changing the velocity model changes the $X_{err(i)}$'s and hence also changes the w_i's. This means that ∇C and $\nabla\nabla C$ will have extra terms due to the weights.

Consider the gradient of the constraint statistic,

$$\frac{\partial C}{\partial v_j} = \sum_i \left[2 X_{err(i)} \frac{\partial X_{err(i)}}{\partial v_j} w_i + X_{err(i)}^2 \frac{\partial w_i}{\partial v_j} \right] \,. \qquad (2.6\text{-}5)$$

The second term of this expression would not normally be present since $\frac{\partial w_i}{\partial v_j}$ is usually zero. The second derivative is,

$$\frac{\partial C}{\partial v_k \partial v_j} = \sum_i \left[2 \frac{\partial X_{err(i)}}{\partial v_k} \frac{\partial X_{err(i)}}{\partial v_j} w_i + 2 X_{err(i)} \frac{\partial X_{err(i)}}{\partial v_j} \frac{\partial w_i}{\partial v_k} + 2 X_{err(i)} \frac{\partial X_{err(i)}}{\partial v_k} \frac{\partial w_i}{\partial v_j} + X_{err(i)}^2 \frac{\partial^2 w_i}{\partial v_k \partial v_j} \right], \tag{2.6-6}$$

where the term containing $\frac{\partial^2 X_{err(i)}}{\partial v_k \partial v_j}$ has once again been ignored (see section 2.5). The complexity of these equations for the gradients of C suggest that applying weights in the inversion process may be difficult. The second derivative has three extra terms.

The derivative of the weight vector must be evaluated. The first derivative is,

$$\frac{\partial w_i}{\partial v_j} = 2 \left[\frac{3}{4} + \frac{1}{4} \cos\left(\frac{\pi X_{err(i)}}{X_{ref}} \right) \right] \left\{ -\frac{1}{4} \sin\left(\frac{\pi X_{err(i)}}{X_{ref}} \right) \right\} \frac{\pi}{X_{ref}} \frac{\partial X_{err(i)}}{\partial v_j} = a_i \frac{\partial X_{err(i)}}{\partial v_j} , \tag{2.6-7}$$

where

$$a_i = \frac{-\pi}{2 X_{ref}} \sin\left(\frac{\pi X_{err(i)}}{X_{ref}} \right) \left(\frac{3}{4} + \frac{1}{4} \cos\left(\frac{\pi X_{err(i)}}{X_{ref}} \right) \right) = \frac{-\pi}{2 X_{ref}} \sin\gamma \left(\frac{3}{4} + \frac{1}{4} \cos\gamma \right) , \tag{2.6-8}$$

and $\gamma = \frac{\pi X_{err(i)}}{X_{ref}}$. The second derivative is,

$$\frac{\partial^2 w_i}{\partial v_k \partial v_j} = \frac{\partial}{\partial X_{err(i)}}\left\{\frac{\partial w_i}{\partial v_j}\right\}\frac{\partial X_{err(i)}}{\partial v_k}$$

$$= \left\{\frac{-\pi}{2X_{ref}}\cos\gamma\,\frac{\pi}{X_{ref}}\left(\frac{3}{4}+\frac{1}{4}\cos\gamma\right)\frac{\partial X_{err(i)}}{\partial v_j}\right.$$

$$-\frac{\pi}{2X_{ref}}\sin\gamma\left[-\frac{1}{4}\sin\gamma\,\frac{\pi}{X_{ref}}\,\frac{\partial X_{err(i)}}{\partial v_j}\right.$$

$$\left.\left.+\left(\frac{3}{4}+\frac{1}{4}\cos\gamma\right)\frac{\partial^2 X_{err(i)}}{\partial v_k \partial v_j}\,\frac{\partial v_k}{\partial X_{err(i)}}\right]\right\}\frac{\partial X_{err(i)}}{\partial v_k}$$

or, on simplification,

$$\frac{\partial^2 w_i}{\partial v_k \partial v_j} = \left\{\frac{-\pi^2}{2X_{ref}^2}\left(\frac{3}{4}\cos\gamma+\frac{1}{4}\cos^2\gamma-\frac{1}{4}\sin^2\gamma\right)\right\}\frac{\partial X_{err(i)}}{\partial v_k}\,\frac{\partial X_{err(i)}}{\partial v_j}$$

$$- b_i\,\frac{\partial X_{err(i)}}{\partial v_k}\,\frac{\partial X_{err(i)}}{\partial v_j}\quad , \tag{2.6-9}$$

where the term containing $\frac{\partial^2 X_{err(i)}}{\partial v_k \partial v_j}$ has been ignored. Using equations (2.6-7) and (2.6-9) the derivatives of the constraint statistic become,

$$\frac{\partial C}{\partial v_j} = \sum_i \frac{\partial X_{err(i)}}{\partial v_j}\left[2w_i X_{err(i)} + a_i X_{err(i)}^2\right] \tag{2.6-10}$$

and,

$$\frac{\partial^2 C}{\partial v_k \partial v_j} = \sum_i \frac{\partial X_{err(i)}}{\partial v_k}\,\frac{\partial X_{err(i)}}{\partial v_j}\left[2w_i + 4a_i X_{err(i)} + b_i X_{err(i)}^2\right] \tag{2.6-11}$$

which shows that applying the weights may not be as difficult as first suggested by equations (2.6-5) and (2.6-6). The variables a_i and b_i are both simple functions of

$X_{err(i)}$, as is w_i. A subroutine is called before the execution of the inversion process to compute the values of w_i, a_i and b_i for the current iteration. This subroutine is called VARIAN and the code is listed in the appendix.

It is crucial that the weights do not decrease too rapidly with increasing $X_{err(i)}$. If this happens it is possible that increasing $X_{err(i)}$ could result in a reduction of the constraint statistic, C (see equation (2.6-3)). Such a condition would prevent the successful inversion of the data. It is necessary that,

$$\frac{\partial C}{\partial X_{err(i)}} = 2X_{err(i)}w_i + X_{err(i)}^2 \frac{\partial w_i}{\partial X_{err(i)}} \qquad (2.6\text{-}12)$$

be positive for all $X_{err(i)}$. Therefore,

$$\frac{\partial w_i}{\partial X_{err(i)}} > \frac{-2w_i}{X_{err(i)}} \qquad (2.6\text{-}13)$$

must be satisfied to prevent the weights from decreasing too rapidly with increasing $X_{err(i)}$.

A direct comparison of inversion results without and with weights can be seen in Figures 2.6-1 and 2.6-2. The synthetic data used were that same as described in section 2.5 (see Figures 2.5-5 and 2.5-6). It is clear that the inversion with $X_{err(i)}$ weighting has achieved superior results.

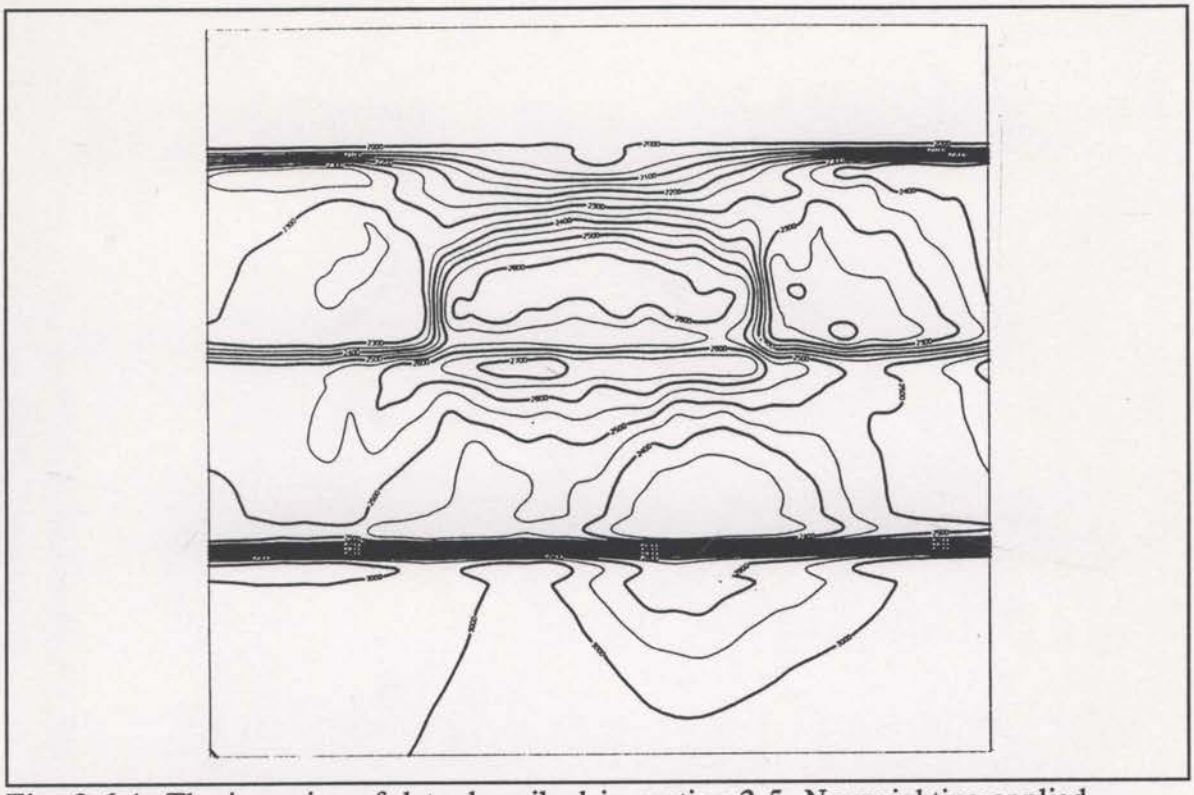


Fig. 2.6-1 The inversion of data described in section 2.5. No weighting applied.

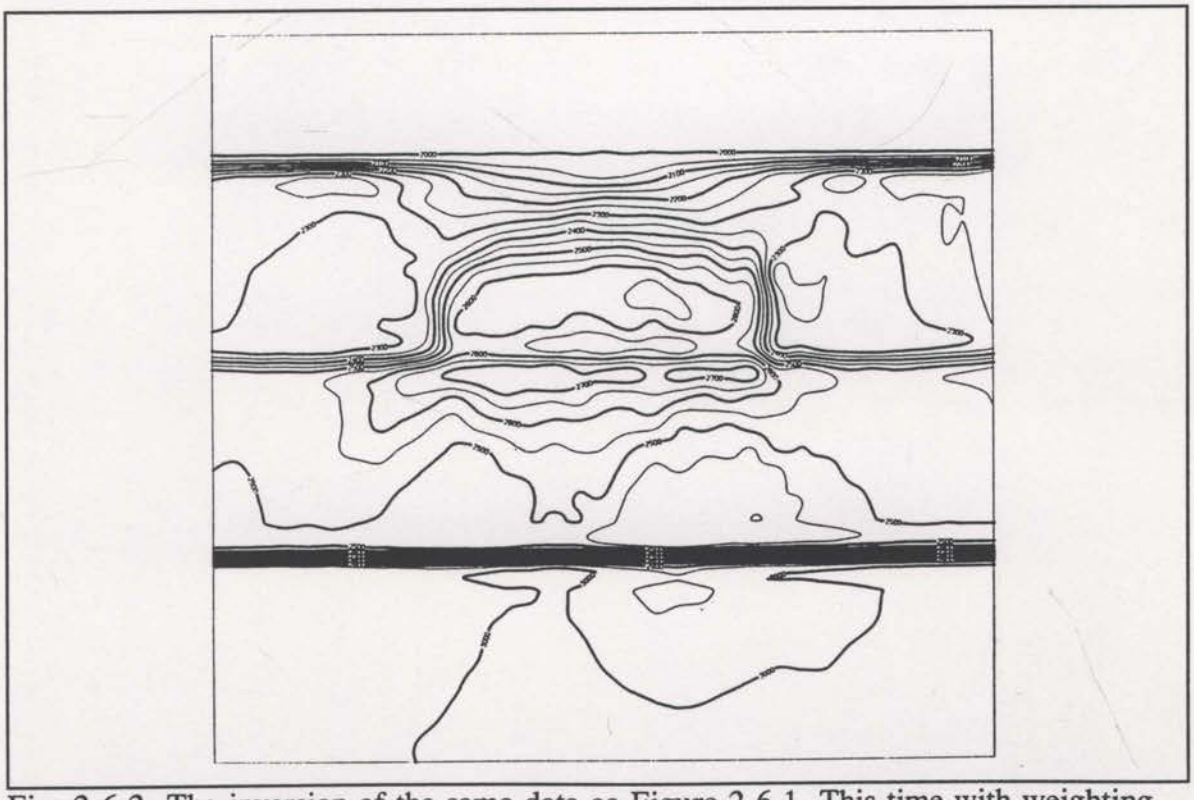


Fig. 2.6-2 The inversion of the same data as Figure 2.6-1. This time with weighting applied (as in equation 2.6-4).

§3 THE ENTROPY PRINCIPLE - THEORY

3.1 Introduction

Tomographic image reconstruction in geophysical applications is generally an underdetermined and somewhat ill-posed problem. The data are almost always incomplete and noisy, and it is found that many different images can represent satisfactory solutions. Since typical images consist of tens of thousands of elements, it is too difficult to display an analysis of error bounds, and the geophysicist is compelled to select a single image as the solution. This is not a very unusual situation as scientific data is often given as the line of best fit through the given data values, even though there are many such lines which would satisfy the data just as well. In geophysical image reconstruction, the solution is generally chosen by imposing some desirable property like smoothness (Sword, 1986), or geological feasibility (van Trier, 1988). The desirable property is generally included by adding a regularisation term to the error function being minimised, creating a new constraint function,

$$\mathbf{F(d,m)} = \mathbf{C(d)} + \lambda \mathbf{B(m)}$$

where the vector **d** represents the data and **m** represents the model parameters. Function C is typically a measure of traveltime residual and function B could be, for example, a smoothing function. In this thesis, C is the sum of squared $X_{err(i)}$ and a smoothing term (see equation 2.5-1) and B is the entropy function. The parameter λ enables trade-off between data fit and model regularisation. This construction results in

a single solution image, but the non-uniqueness of the solution has not really been removed; it is now represented by the arbitrariness of B.

The entropy principle provides a mechanism for choosing, from the range of suitable solution images, the image that has no unjustifiable bias relative to the data and any prior information. Understanding this principle can be confusing as it is possible to derive and explain it from many different viewpoints. Basically, the entropy principle states that the solution image that should be selected is the one with the maximum entropy value (the entropy function will be defined later). The maximum entropy solution can be thought of as the one with a configuration that has the most possible ways of actually occurring. It can also be thought of as the solution which expresses the least amount of information necessary to satisfy the data. Most choices of the regularisation function, B, will actually result in solutions that express more information than is required to satisfy the data values and prior information. When this happens, there is an implicit assumption that more information is used than is at hand, and the unwarranted structure of the resulting image cannot be defended. The entropy function guards against this possibility and provides the "common sense" solution.

The history of the entropy principle was developed in two disciplines; thermodynamics and information theory. In thermodynamics, entropy is the basic parameter of the Second Law of Thermodynamics. The history of entropy in thermodynamics has been well documented by Jaynes, 1988a. The entropy function appeared in Kelvin and Clausius' work on heat engines as a measure of the thermodynamic state of the system. Gibbs then showed that any given system, not in equilibrium, will move towards a state of equilibrium that has the maximum entropy allowed by the constraints imposed on the system. Boltzmann explained why

thermodynamic systems approach a state of maximum allowed entropy when he found that the entropy value was actually a measure of the phase volume of a gas with given energy. Since the phase volume corresponds to the number of global quantum states, it is concluded that the maximum entropy state is the one that can occur in the most ways (see Jaynes, 1988a). In thermodynamics, maximum entropy solutions are thought of as states which have the most ways of actually occurring. The concept of least information did not emerge in this context.

As a very simple example, consider the release of some gas molecules into an empty box. What will be the equilibrium distribution of the gas molecules? Common sense suggests that a uniform distribution would result, even though no information exists to allow this conclusion to be deduced (it depends on the initial trajectories and velocities of all the molecules). It appears that there would be many more ways that the gas molecules could physically scatter into a uniform distribution rather than, for example, a distribution where the molecules tend to concentrate in the left half of the box. This is why a uniform distribution is the common sense answer, not because it must be correct but because it has many more chances to be correct. The number of ways any particular distribution can be realised is quantifiable if the box is divided into a number of discrete locations. Given N molecules and M locations, then,

$$W = \frac{N!}{n_1!n_2!...n_M!} \tag{3.1-1}$$

is the number of ways the molecules can be rearranged to give the same distribution (n_i is the number of molecules at location i). The value of W is greatest when all n_i equal N/M, that is, a uniform distribution. If the box is divided into just two halves

and only fifty molecules are considered, then the number of ways a uniform distribution could be realised is,

$$\frac{50!}{25!25!} = 1.26\times10^{14} \quad ,$$

whereas a distribution with eighty percent of the molecules in one half of the box can only be realised in

$$\frac{50!}{10!40!} = 1.03\times10^{10}$$

ways. Therefore, there are 12233 possible uniform distributions to every 80%-20% distribution. It would be inconsistent (lacking any further information) to choose an 80%-20% distribution as the molecular state when there are so many more ways that the uniform distribution could occur. Furthermore, as the number of molecules increases the uniform distribution becomes even more likely (for 60 molecules the ratio jumps to 84285 to 1). When the number of molecules reaches the many millions contained in actual gases, distributions other than uniform become so unlikely that the maximum entropy solution appears to be a law of nature rather than just the most statistically likely solution. It actually turns out that all the classical gas laws can be derived from maximum entropy distributions.

The mathematical link between maximum entropy and the number of ways of realisation is given by the Stirling approximation. If N is large, the Stirling approximation gives (see Jaynes, 1982),

$$\frac{1}{N}\log W \approx -\sum_i \frac{n_i}{N}\ln\frac{n_i}{N} \quad , \qquad (3.1\text{-}2)$$

which relates the classical entropy function ($S = -\sum_i \frac{n_i}{N} \ln \frac{n_i}{N}$, see next paragraph) to

the measure of the number of ways that a proposed solution can be realised, W. The maximum entropy solution is not necessarily the correct answer, but it has more ways in which it can be correct.

The other discipline to which the principle of entropy can be traced is the field of information theory and the work of Shannon (1947/48 : for an explanation of information theory and proofs of Shannon's theorems, see Khinchin, 1957). The entropy function of information theory has the same form as the right-hand side of equation (3.1-2) and is used as a measure of either information content or the uncertainty of a stochastic distribution. Given a random process with i possible outcomes, each of probability p_i, then Shannon defined the amount of information gained from the realisation of outcome i to be $-\ln p_i$. The mathematical expectation of the information gained is then,

$$S = -\sum_i p_i \ln p_i \qquad (3.1\text{-}3)$$

which is once again the classical form of the entropy function.

It would appear that the two historical sources of the entropy function are unrelated, but an information theory analogy can be made to the gas molecule example. Consider a random process that is unknown except for the fact that it has i possible outcomes. What will be the probability distribution of this random process? Common sense suggests that, without any further information, a uniform or flat probability distribution is the best answer. In a very similar way to the gas molecule

example, there are many more ways that the probabilities can be arranged to give a uniform distribution. From the viewpoint of information, any probability distribution which is non-uniform will favour some outcomes over others, but in the absence of prior information the given data does not allow for such conclusions. The uniform distribution represents the least amount of information that needs to be presented in order to solve the problem. Only when further information is available can structure be added to the probability distribution. It is interesting to note that if the mean and variance of a continuous process are known, then the maximum entropy solution is the Gaussian or normal distribution (see Skilling and Gull, 1984).

In a series of important papers, Jaynes (for example 1957, 1982, 1984, 1985, 1988a, 1988b) has explained the rationale and essence of the entropy principle with reference to many disciplines. The entropy principle can be thought of as a means of imposing common sense into difficult problems. As a further simple example, assume that a fair coin has been tossed a million times. What percentage of heads has been tossed? Common sense suggests that fifty percent will be close to correct. This is the maximum entropy answer. However, there is a very slight chance that a million heads had been tossed in a row. Since there are so many more ways that the coin could be tossed to result in fifty percent heads, it makes sense to choose this as the solution (assuming no further information is known). These ideas are related to Laplace's "Principle of Insufficient Reason" (Jaynes (1988b)). In the problems of image reconstruction there are tens of thousands of variables and using the entropy principle ensures that such common sense decisions are made at every step.

During the last few decades, scientists have been applying the principle of entropy to a number of problems outside statistical mechanics and information theory.

Some examples include deconvolution (Skilling and Gull, 1985), spectral estimation (Burg, 1967; Ables, 1974) and various image reconstruction problems in astronomy (again see Skilling and Gull, 1985). Recently the entropy principle has been applied to geophysical traveltime tomography (Bassrei, 1990; Whiting, 1991b).

Image reconstruction using the entropy principle has been used successfully in X-ray, γ-ray and radio astronomy for a number of years (Gull and Daniell (1978); Skilling and Bryan (1984); Skilling and Gull (1985)). The aim is to reconstruct an image of a portion of the sky from measured radiation. The sky can be divided into discrete cells and the intensity of radiation originating from each cell can be treated as the outcome of a random process (the distribution of stars across the sky is effectively random). Information theory arguments can be used to rationalise the use of entropy in such astronomical problems (even so, there can still be conceptual problems with this, see Cornwell, 1983).

Consider now the problem of traveltime tomographic image reconstruction in geophysics. Here, the objective is to construct an estimate of the subsurface seismic velocity field on a grid of discrete points. It is not obvious how to associate such a problem with any random feature such as the movements of gas molecules or the outcomes of a random process. This introduces a difficulty in the use of entropy constraints in such inverse problems.

One popular way to introduce the entropy principle in image reconstruction problems has been through contingency tables (Gull and Skilling, 1983). The entries of the table are effectively the outcomes of a random process and the application of the entropy principle is straightforward (Good, 1963). Apart from the similar appearance of contingency tables and images divided into regular cells, there is a conceptual

difficulty in making a valid association between the two problems. However, knowledge gained from contingency table analysis is directly applicable to one particularly simple traveltime tomography problem that will be considered in the following sections. Along with a few straightforward rules for image reconstruction, it can be shown that the entropy principle is applicable to general traveltime problems. This results in less spurious structure in the solutions and more stable inversions.

3.2 Contingency tables

A two-dimensional contingency table records the number of occurrences of two variables in a crosstabulated format. A typical example is the crosstabulation of the type of weather against the season, as seen in Figure 3.2-1. The entries in the table are the observed number of days of each of the types of weather in each of the four seasons of the year. Analysing a table like this often involves estimating the degree of association between the two variables being observed (see Press et al., 1989, section 13.6), but the primary interest here is in trying to reconstruct the table given only partial information. This problem is ideally suited to the use of the principle of entropy (Good, 1963) since the partial information allows for many possible solutions.

A 'marginal' value of a contingency table is defined as the sum of a single row or column of the table. That is, for example, it represents the total number of fine days or the total number of days in winter. Consider solving for the values of the table given only the eight marginal values. The solution to this problem is highly non-unique and the best solution is not immediately obvious. The marginal distribution for the contingency table is indicated by the arrows of Figure 3.2-1. To demonstrate the non-uniqueness in the problem, the table of Figure 3.2-2 is presented. This reconstructed table is obviously very different to the actual table of Figure 3.2-1. It clearly lacks common sense as well since it suggests that every day in winter was fine and the only days of snow were in spring. Even so, this table does satisfy the marginal distribution. From the marginal values, it rained for 178 days and there were 90 days of winter, however, the reconstruction of Figure 3.2-2 assumes that there were no rainy days

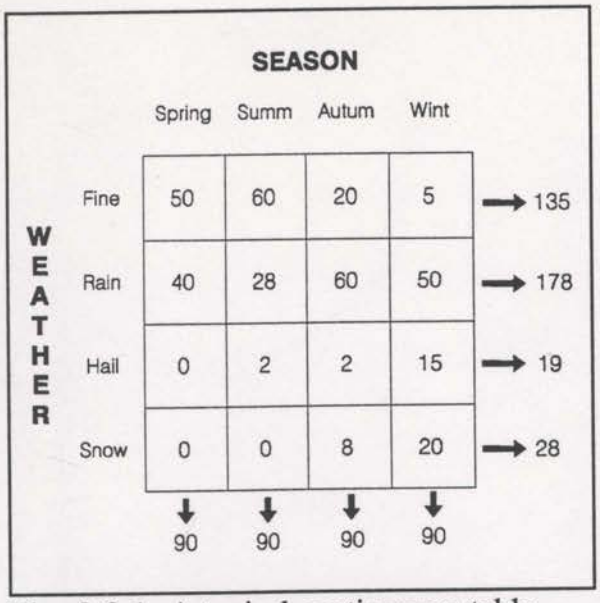


Fig. 3.2-1 A typical contingency table recording the type of weather and the season, sampled over a complete year. The marginal values are posted next to the arrows.

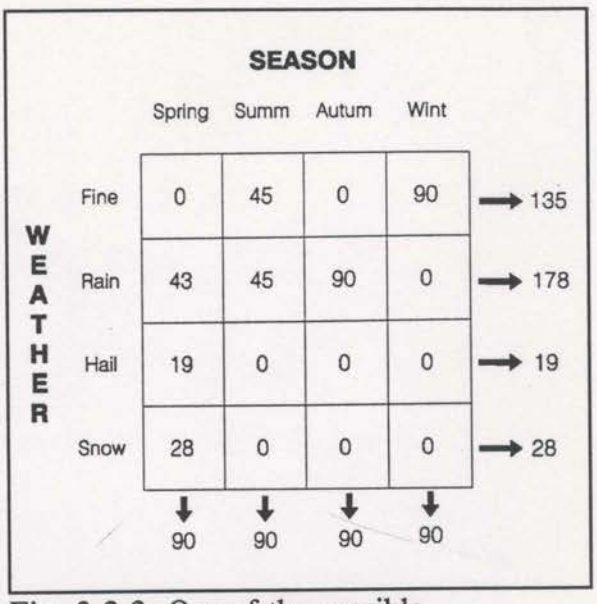


Fig. 3.2-2 One of the possible reconstructed contingency tables that satisfy the marginal values.

actually in winter. Given just the marginal data there is no basis for making this claim. More information is being assumed than is at hand.

The marginal data contains no information about the correlations that exist between any particular row or column. Without any further information, the best that can be done is to assume that the rows and columns have no association and are statistically independent. If the row marginal shows that $p\%$ of the days are rainy, then all that can be said without introducing unjustifiable bias, is that $p\%$of the days of each season are rainy. If the data are defined as,

r_i = row marginal for row i, i = 1,4

and c_j = column marginal for column j, j = 1,4

and with V_{ij} = the reconstructed value crossed by row i

and column j,

then the unbiased or uncorrelated reconstruction is given by,

$$V_{ij} = \frac{r_i c_j}{N}, \tag{3.2-1}$$

where N is the total number of days (see Press et al., 1989, p478). Reconstructing the table in this way leads to the result of Figure 3.2-3. This result is also very different from the actual table shown in Figure 3.2-1, but it represents all the information we can extract from the marginal data alone without introducing further assumptions. This solution shall be called the "common sense" solution.

It is possible (and usually desirable) to introduce a deliberate bias through the use of prior information. The contingency table in question here is an example where prior information certainly exists; for example, it is known that it is more likely to snow in winter than in summer. To avoid complicating the current analysis the effects of prior information will be considered later.

The usefulness of this contingency table analysis becomes clear when it is observed that the uncorrelated reconstruction (Figure 3.2-3) is actually the maximum entropy reconstruction when given only the marginal data. Frieden (1972) was one of the first to relate entropy to image problems, and Skilling (1988) derived the entropy function particularly for image

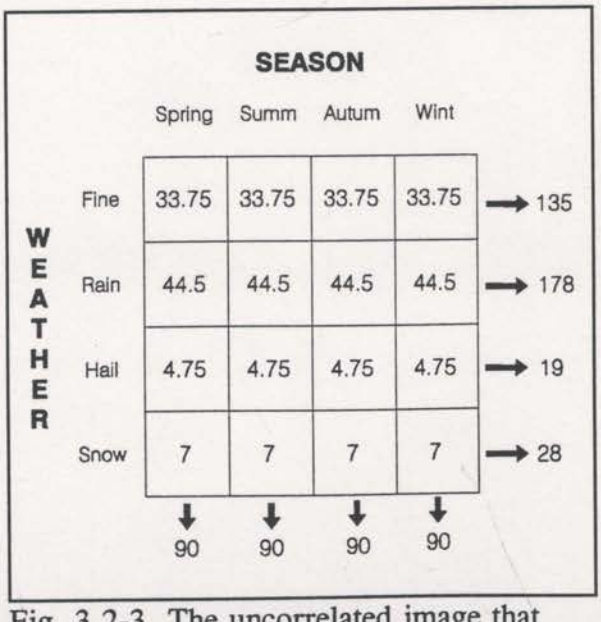


Fig. 3.2-3 The uncorrelated image that satisfies the marginal values. Created using equation (3.2-1). This is the "common sense" image.

reconstruction. The entropy function for the current contingency table problem can be written as,

$$S = \sum_i \sum_j (V_{ij} - V_{ij} \ln V_{ij}). \tag{3.2-2}$$

The entropy function of equation (3.1-3) is only for values between 0 and 1 (probabilities), this new form does not require normalisation of the image values and is to be maximised directly while satisfying the constraints in the form of the marginal values. The usual method for doing this is by introducing Lagrange multipliers (α_i and β_j) and the variational equation,

$$\frac{\partial}{\partial V_{ij}} \left[\sum_i \sum_j (V_{ij} - V_{ij} \ln V_{ij}) + \sum_i \alpha_i \left(\sum_k V_{ik} \right) + \sum_j \beta_j \left(\sum_k V_{kj} \right) \right] = 0. \tag{3.2-3}$$

Solving this for one of the locations of the table we get,

$$-\ln V_{ij} + \alpha_i + \beta_j = 0 \tag{3.2-4}$$

or,

$$V_{ij} = e^{\alpha_i + \beta_j} = e^{\alpha_i} . e^{\beta_j} \tag{3.2-5}$$

where α_i and β_j must be chosen such that the relevant marginal values are satisfied. It is a simple matter to show that equation (3.2-5) is equivalent to equation (3.2-1). Thus, maximising the entropy function (equation (3.2-2)) has reconstructed the unbiased, "common sense" contingency table. The critical feature of the entropy function is the logarithm. The logarithmic form of S directly results in the exponentials in equation (3.2-5), thus converting the added constraints into a multiplicative term that actually allows the multiplication of the marginal values, as required by equation (3.2-1). No other form would be able to convert an addition to a multiplication as required.

Statistical theory allows us to measure the correlation in the table by the use of the covariance,

$$\mathrm{cov}(i,j) = \mathrm{E}[ij] - \mathrm{E}[i]\,\mathrm{E}[j]. \tag{3.2-6}$$

Here, the contingency table is viewed as a two-dimensional random variable. The expectation values are defined as,

$$\left.\begin{aligned} \mathrm{E}[i] &= \sum_i i\frac{R_i}{N}, \\ \mathrm{E}[j] &= \sum_j j\frac{C_j}{N}, \\ \mathrm{E}[ij] &= \sum_i \sum_j ij\frac{V_{ij}}{N}. \end{aligned}\right\} \tag{3.2-7}$$

Note that the row and column numbers have been used as the values of the constructed random variables. If the reconstructed image is to be uncorrelated then,

$$\begin{aligned} &\mathrm{E}[ij] = \mathrm{E}[i].\mathrm{E}[j] \\ &\sum_i \sum_j ij\frac{V_{ij}}{N} = \sum_i i\frac{R_i}{N}.\sum_j j\frac{C_j}{N} \\ &= \sum_i \sum_j ij\frac{\left(\frac{R_i.C_j}{N}\right)}{N} \end{aligned} \tag{3.2-8}$$

therefore, $V_{ij} = \frac{R_i.C_j}{N}$, as before. The table created by common sense (Figure 3.2-3) and by maximising the entropy function is the only uncorrelated reconstruction given the marginal values. Thus the "common sense", uncorrelated and maximum entropy tables are all equivalent.

Maximising entropy (equation (3.2-2)) ensures that no extra information is assumed while the contingency table is being reconstructed. It is desirable to duplicate

this property in general tomographic reconstructions. The next section looks at a simple traveltime tomography problem that is very similar to the contingency table problem. The simple model proves to be very useful later, when more general tomography problems are considered.

3.3 A simple tomographic problem.

Skilling (1988) made use of a simple tomographic problem to demonstrate the necessity of the logarithmic structure of the entropy function. Here, this simple problem provides the link between the above analysis of contingency tables and more general tomographic image reconstruction problems.

The model for this simple problem is a square with unit side length. This square can be evenly divided into any number, M by M, of discrete square cells. The aim is to reconstruct the image V_{ij} given measurements of the kind,

$$\frac{1}{M}\sum_{j=1}^{M} V_{ij} = A_i , \qquad i=1,...,M \tag{3.3-1}$$

where $1/M$ is the side length of each cell, and,

$$\frac{1}{M}\sum_{i=1}^{M} V_{ij} = B_j , \qquad j=1,...M. \tag{3.3-2}$$

That is, there are tomographic projections along every row and down every column. In analogy with the contingency table problem, these data are called marginal data values. Furthermore, assume that the marginal values are normalised,

$$\sum_{i=1}^{M} A_i = \sum_{j=1}^{M} B_j = 1. \tag{3.3-3}$$

Despite its simplicity, this is a very important problem since the desired reconstructed image is known. From the analysis of contingency tables, the only image that does not introduce unjustified bias is that which multiplies the marginal data values that intersect at each cell. This image is uncorrelated and is the only one which does not assume any

extra information.

The image can now be reconstructed by satisfying the constraints (equations (3.3-1) and (3.3-2)) subject to the maximisation of the entropy function (equation (3.2-2)). This is done by introducing Lagrange multipliers (α_i and β_j) and constructing a variational equation,

$$\frac{\partial}{\partial V_{ij}}\left[\sum_i\sum_j (V_{ij} - V_{ij}\ln V_{ij}) + \sum_i \alpha_i \sum_j \frac{V_{ij}}{M} + \sum_j \beta_j \sum_i \frac{V_{ij}}{M}\right] = 0 \qquad (3.3\text{-}4)$$

which is essentially the same as the variational equation for the contingency table problem (equation (3.2-3)). The reconstructed image for this problem is therefore,

$$V_{ij} = A_i B_j, \qquad (3.3\text{-}5)$$

the product of the marginal data values. Just as with the contingency tables, maximising entropy results in the desired uncorrelated image.

Therefore, given tomographic data of complete simple horizontal and vertical projections through a square medium, it has been shown that maximising entropy during image reconstruction will result in the unique uncorrelated image (as required). Using any function other than the entropy function (equation (3.2-2)) for regularisation may introduce unjustified structure into the image.

The model used in this section serves as the link between contingency tables and tomographic problems. However, tomographic projections in geophysics are curved and not evenly distributed and there are usually far too many of them to permit each to be treated with its own Lagrange multiplier. The next section considers these problems.

3.4 General Traveltime Tomography Problems

The general traveltime tomography problem is represented schematically in Figures 3.4-1 and 3.4-2. Figure 3.4-1 displays some typical raypaths that may be encountered in crosswell problems and Figure 3.4-2 shows typical raypaths for reflection data. Note that reflector depths are to be ignored here (see section 2.2). It has been shown that reflection tomography can be performed without reflector depth parameterisation (Sword, 1986; Whiting, 1991b), but combined velocity and depth entropy functions can be constructed if necessary (see Skilling and Gull, 1984). In both the crosswell and reflection cases, the raypaths (or projections) do not form the simple, regular marginal data values of the example in the previous section and the desired solution cannot be obtained by simply multiplying certain data values. The covariance cannot even be measured in these problems as the data are not suitable.

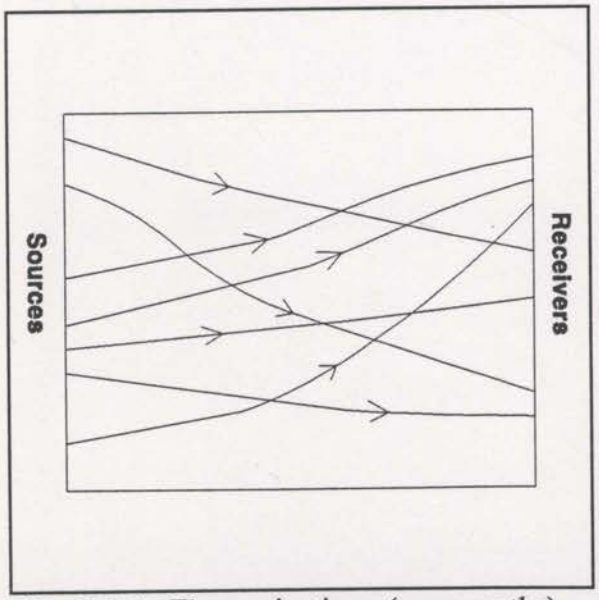


Fig. 3.4-1 The projections (or raypaths) of the general crosswell tomography problem.

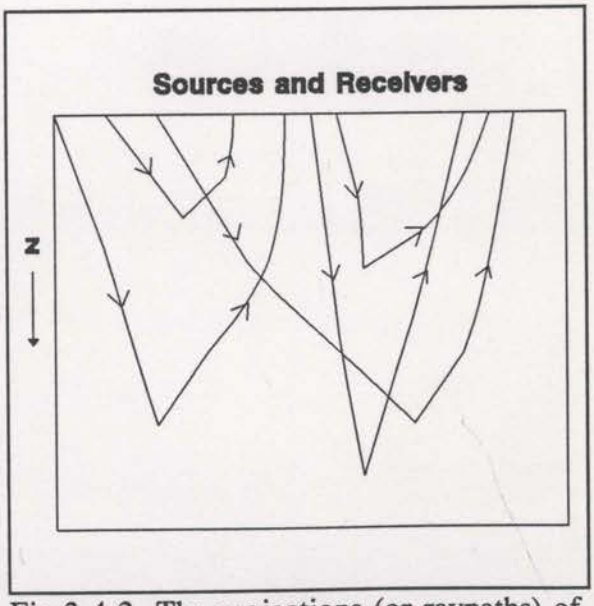


Fig 3.4-2 The projections (or raypaths) of the general reflection tomography problem.

Also, as mentioned earlier, there are generally so many raypaths that it is impractical to assign them individual Lagrange multipliers.[2] Normally all of the rays are collected together with their errors summed into a single constraint statistic,

$$C = \sum_{k} (t_d - t_m)^2 \qquad (3.4\text{-}1)$$

where t_d is the actual data traveltime and t_m is the traveltime computed by tracing the kth raypath through the model[1]. Some information is actually lost by forming this single C statistic since this statistic does not comprehend where the raypaths are. However, with a good coverage of rays and the assumption that the derivatives, $\frac{\partial C}{\partial V_{ij}}$, can be accurately computed, a useful tomographic reconstruction is still possible. Now that the tomographic problem is specifically defined as a traveltime problem, the image elements, V_{ij}, will be referred to as velocity elements (or cell velocities).

The information at hand for the solution of this general tomographic image reconstruction problem is simply the value of the constraint statistic, C, and its Frechet derivatives, $\frac{\partial C}{\partial V_{ij}}$. The aim of the tomographic inversion is to minimise C, or at least reduce it to a predefined level. Since the function C is highly non-linear, this can only be achieved in practice by making a series of small changes through successive iterations. However, once the predefined level is reached, it is found that a

[2] Carrion (1991) has proposed a tomography scheme based on the assignment of a Lagrange multiplier for each ray.

[1] This is the general constraint statistic - the traveltime residual. In this study the $X_{err(i)}$ statistic is used (see sections 2.2 and 2.4).

large number of images can have the same C value. It is not easy to single out one of these images as any image that reduces C sufficiently is clearly acceptable, given only the constraint statistic and its derivatives. To help choose an image, prior information is introduced. The prior information is supplied from prior knowledge of the medium and is represented by a prior information model, m_{ij}, which is essentially the "best guess" starting model. In practice there is always some prior information available, even if it is just a simple velocity increase with depth.

Once the prior model is defined, all that has to be done is to choose the solution which maximises entropy relative to this prior model. With a prior model, the entropy function is modified to (see Skilling, 1988),

$$S = \sum_i \sum_j \left(V_{ij} - V_{ij} \ln \frac{V_{ij}}{m_{ij}} \right). \qquad (3.4\text{-}2)$$

The form of entropy given in equation (3.2-2) assumed that there was no prior information. Note that equation (3.4-2) reduces to equation (3.2-2) when all $m_{ij} = 1$.

However, it is not obvious that maximum entropy solutions will be especially desirable in general tomographic problems. Before a serious analysis of this can proceed, the desirable properties of the solution must be defined. The next section establishes some common sense rules for reconstructing satisfactory solution images.

3.5 Least Commitment

The general tomographic image reconstruction problem requires a constraint statistic, C, along with its derivatives, $\frac{\partial C}{\partial V_{ij}}$, and a prior information model, m_{ij}. The aim of the inversion is to achieve a predefined reduction in C by varying the velocities, V_{ij}, from the prior model, m_{ij}. The simple example shown in Figure 3.5-1 will be used to help visualise some common sense restrictions to be imposed on the reconstruction of the image. In Figure 3.5-1 the starting position portrayed assumes that $V_{ij} = m_{ij}$ in every cell, thus resulting in $V_{ij}/m_{ij} = 1$ everywhere. Also, let the derivative of the constraint statistic be the same for every cell.

There are a large number of ways in which the velocity values can be changed so that the required reduction of the constraint statistic is achieved. At one extreme, a single cell can be changed enough to effect the total necessary reduction (see Figure 3.5-2). However the choice of the modified cell is totally arbitrary. Such sources of arbitrariness are contrary to the common sense approach of maximum entropy methods.

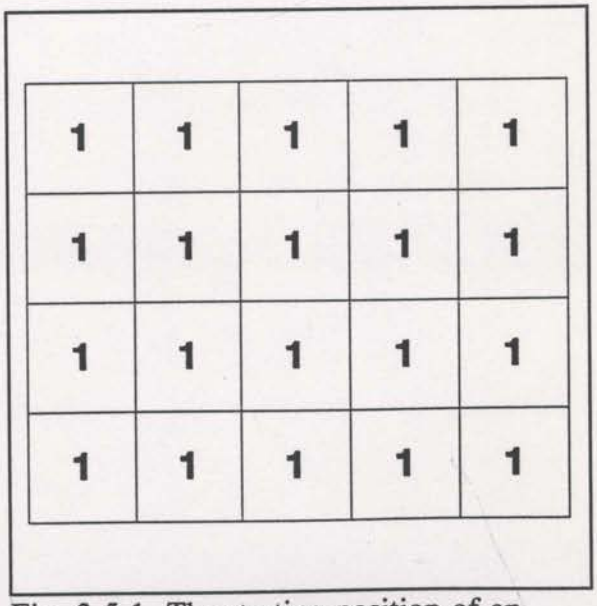


Fig. 3.5-1 The starting position of an instructive image reconstruction problem. V_{ij}/m_{ij} is plotted where, at the start, $V_{ij}=m_{ij}$ everywhere.

Since all of the constraint statistic derivatives are equal, common sense suggests that no cell can be singled out in preference to any other cell. In this case, the only sensible reconstruction is the one which changes all of the cell velocities equally. Picking a single cell to carry the full burden of the C reduction causes the reconstruction to make a very strong commitment to that cell. However, it is preferable to impose as little commitment to any one cell as possible while reducing C by the specified amount. Intuitively, the reconstruction with equal changes that is desired here, would also be the reconstruction with the least overall commitment.

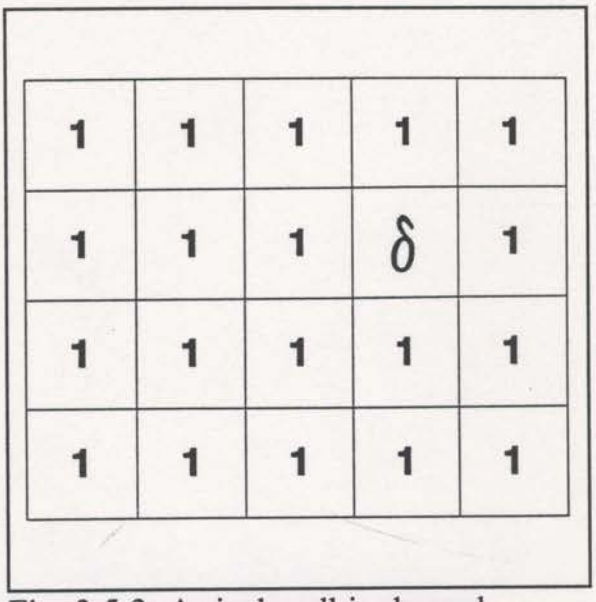


Fig. 3.5-2 A single cell is changed sufficiently (to $V_{ij}/m_{ij}=\delta$) to reduce C to the required level.

On the other hand, if one cell had a larger constraint statistic derivative than the rest, it would be desirable for this cell to bear more of the required velocity change. At first it appears that this would increase the overall commitment level, but because of the larger derivative, the velocity change in this cell would be more than offset by a decrease of the change required in all the other cells. The final result would actually be of lower overall commitment than for a uniformly changed model. Clearly, for the concept of commitment to be evaluated further, a method of measuring the overall level of commitment must be defined.

Assume that it is possible to measure the amount of commitment using a non-negative function, ψ, of V_{ij}/m_{ij} (let $x_{ij} = V_{ij}/m_{ij}$ for brevity). It is proposed that the

following four properties suffice to define such a function (with $m_{ij} > 0$, and ψ defined for $x_{ij} > 0$);

(a) $\psi(x_{ij}) \geq 0$,

(b) $\psi(1) = \psi'(1) = 0$,

(c) $\psi''(x_{ij}) > 0$ for all x_{ij} ,

and (d) $\psi'(x_{ij}) \to -\infty$ *as* $x_{ij} \to 0^+$.

Property (a) implies that any variation of V_{ij} from m_{ij} introduces positive commitment. Property (b) states that $V_{ij} = m_{ij}$ is the state of zero commitment and that this is a local minimum. Properties (c) and (d) will require further explanation.

To help understand property (c), imagine that Figure 3.5-2 is an intermediate reconstruction (even though not a very acceptable one) and all of the gradients, $\frac{\partial C}{\partial V_{ij}}$, are equal. To reduce C to the required level assume that a further increment ϵ (>0) in velocity is required. Figures 3.5-3 and 3.5-4 suggest two of the possible ways of doing this, either by adding the extra change to the cell already changed or adding it to any other cell. The image of Figure 3.5-3 has greater overall commitment than that of Figure 3.5-4, and for the function ψ to reflect this,

$$[\psi(\delta+\epsilon) - \psi(\delta)] > [\psi(1+\epsilon) - \psi(1)]$$

or more generally,

$$[\psi(a+\epsilon) - \psi(a)] > [\psi(b+\epsilon) - \psi(b)] \quad , \quad \text{if } a > b \geq 1.$$

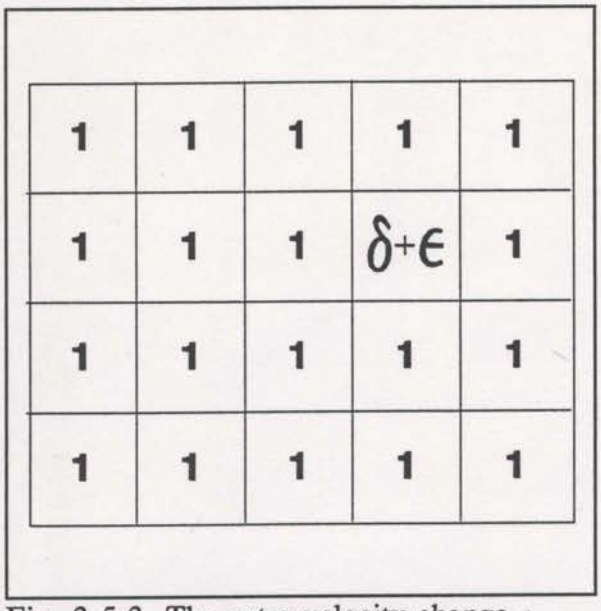


Fig. 3.5-3 The extra velocity change ϵ could be added to the cell already changed.

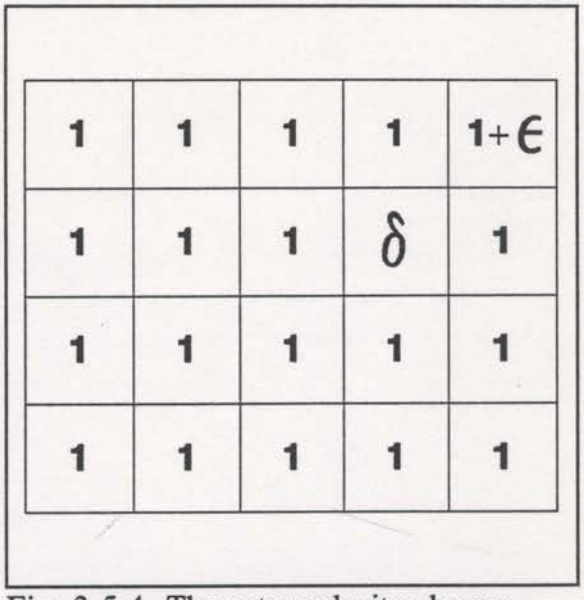


Fig. 3.5-4 The extra velocity change ϵ could be added to any other cell.

This inequality is fulfilled if ψ is convex, or $\psi'' > 0$ for all x_{ij}.

The above inequality is actually more powerful than it first appears. If ϵ is to be added to one of the cells in Figure 3.5-4 other than the cell of value δ, then the above inequality gives,

$$\left[\psi([1+\frac{\epsilon}{2}]+\frac{\epsilon}{2}) - \psi(1+\frac{\epsilon}{2})\right] > \left[\psi(1+\frac{\epsilon}{2}) - \psi(1)\right]$$

or,

$$\psi(1+\epsilon) > 2\psi(1+\frac{\epsilon}{2}) \quad .$$

This means that splitting the additional ϵ increment into two halves and adding these to two cells, is a state of less commitment than adding all of ϵ to a single cell. This argument can obviously be carried on until the ϵ velocity increment is evenly divided across all available cells. It can be shown that when the original δ increment was

made (Figure 3.5-2), the minimum commitment reconstruction would have been one with equal velocity changes in every cell. Therefore, the simple condition of convexity imposed on the commitment function, ψ, is sufficient to ensure that least commitment reconstructions will be sensible, as intuitively defined earlier.

The convexity property defined above, also has relevance for the case where the constraint statistic derivatives are not equal. In this case, common sense requires that more of the velocity change be carried by cells with larger derivatives. For the moment, consider a two velocity cell example where the constraint statistic derivatives are not equal (also assume that both derivatives are negative - similar arguments follow for other values). The reconstruction with equal changes in velocity must not be the least commitment reconstruction in this case. Due to the convexity of the commitment function, an increase in the velocity of one cell will only produce lower commitment if the decrease in the other cell is greater (how much greater depends on the degree of convexity). The only way this can happen, and still produce the same constraint statistic reduction, is if the cell with the larger derivative is the one increased. Therefore, the convexity of the commitment function also forces more of the velocity change to be taken up in cells with larger constraint statistic derivatives.

The fourth property, where the derivative of the commitment function is asymptotic to $-\infty$ at the $x=0$ axis, is specified in order to make it effectively impossible for a zero velocity to be reconstructed. Negative and zero velocities will be naturally avoided.

The four properties listed above define a commitment function that will enforce apparent common sense (as intuitively described above) into the reconstruction of the image. However, the definition is not so constrained as to disallow quite a degree of

flexibility. The form of the required regularisation function has been defined, but not its details. Skilling (1988) showed that if the reconstruction problem happened to be of the form described in the simple tomographic problem section, then the regularisation function needed to obtain the required solution is the entropy function of equation (3.4-2). This special case implies a commitment function given by,

$$\psi(x_{ij}) = A\left[1 - x_{ij} + x_{ij}\ln x_{ij}\right] \qquad ;A>0 \,, \tag{3.5-1}$$

which satisfies all four properties described previously. The fact that the entropy function is maximised in practice rather than ψ minimised, means that entropy can be thought of as a measure of non-commitment.

The entropy function (equation (3.4-2)) is schematically represented in Figure 3.5-5. It is maximum at $x_{ij} = V_{ij}/m_{ij} = 1$ and its domain is $x_{ij} \geq 0$. Also, the logarithmic form ensures that the magnitude of its derivative monotonically increases as V_{ij}/m_{ij} moves away from 1 (hence satisfying property (c)). Therefore, based on the preceding discussion, maximising the entropy function will ensure that commitment to individual cells is kept to a minimum. This in turn implies that the entropy function will ensure sensible (or common sense) image reconstructions in both the "simple" and "general" tomography problems.

The actual form of the maximum entropy reconstruction is most easily observed from the variational equation. The general problem is to minimise the constraint statistic C while maximising entropy. In practice however it is more convenient to maximise entropy subject to an acceptable level of C. This reduces to the variational equation,

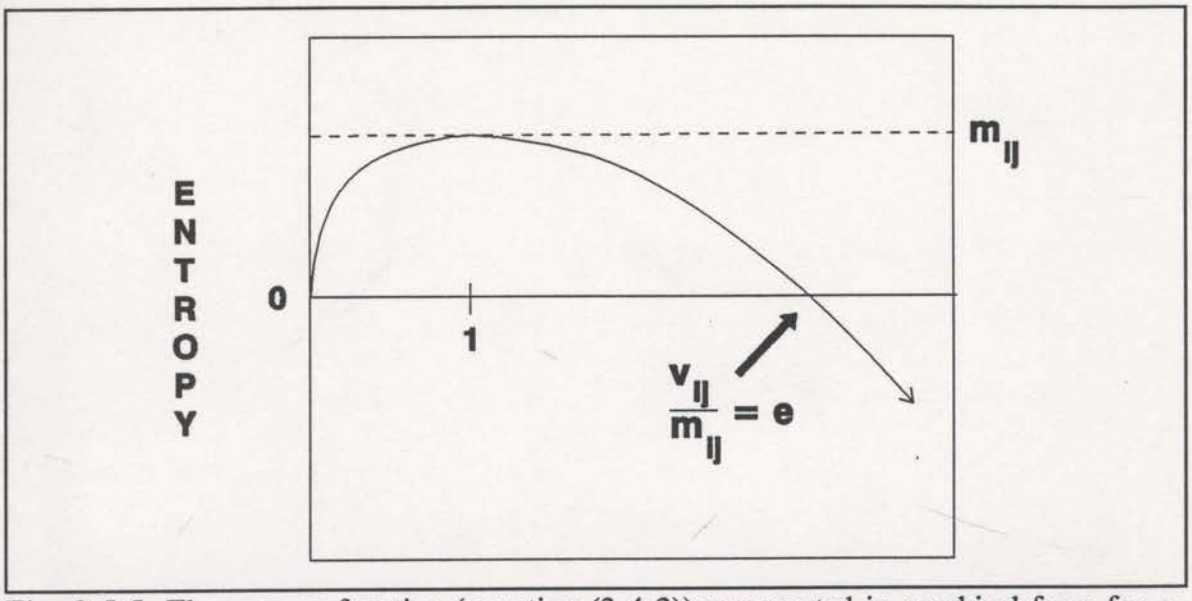


Fig. 3.5-5 The entropy function (equation (3.4-2)) represented in graphical form for a single cell.

$$\frac{\partial}{\partial V_{ij}}\left[\sum_i \sum_j \left(V_{ij} - V_{ij}\ln\frac{V_{ij}}{m_{ij}}\right) + \lambda C\right] = 0\,, \tag{3.5-2}$$

where λ is a (single) Lagrange multiplier. This leads to,

$$-\ln\frac{V_{ij}}{m_{ij}} + \lambda\frac{\partial c}{\partial V_{ij}} = 0$$

or, equivalently,

$$\frac{V_{ij}}{m_{ij}} = e^{\lambda\frac{\partial c}{\partial V_{ij}}}\,, \tag{3.5-3}$$

where λ is chosen so that the required reduction of C is achieved. Equation (3.5-3) implies that all the cells with the same gradient, $\partial C/\partial V_{ij}$, will have exactly the same velocity change relative to their prior values, m_{ij}. Also, the larger the value of $\partial C/\partial V_{ij}$, the larger will be the relative velocity change. Some simple examples will now show how maximising entropy influences actual inversions.

3.6 Simple Examples

In this section, the common sense properties (defined in the previous section) of desirable tomographic image reconstructions will be demonstrated with the help of some very simple examples. Maximum entropy reconstructions will be compared with three other possible solution images. The first reconstructed image that will be compared to the maximum entropy image is just the smoothest image that solves the constraints. The second is the "least change" solution[1] (that is, the one that minimises $\sum (v_{ij} - v_{ij}^o)^2$ where v_{ij}^o is the starting velocity), and the third is the "least change from prior" solution (the one that minimises $\sum (v_{ij} - m_{ij})^2$ where m_{ij} is the prior information velocity model).

The first example demonstrates the case where the reconstructed image may not be representative of the information given by the constraint statistic derivatives, or more simply, the sensitivities. Consider an image that is to be tomographically reconstructed and consists of only two cells. This model could be traversed by any number of rays but the only information that matters is the sensitivity of the constraint statistic to velocity changes in each cell. For this example, assume that the sensitivities are $\frac{\partial C}{\partial v_1} = -1$ and $\frac{\partial C}{\partial v_2} = -2$, and that the required C reduction for an acceptable solution is 1 unit. With a starting model equal to the prior information model of $v_1 = v_2 = 1$, the

[1] Also known as the "minimum norm" solution.

$v_1 = v_2 = 1$, the reconstructions are (v_1, v_2)=(1.1861,1.4069), (1.3333,1.3333), (1.2,1.4) and (1.2,1.4), being respectively the maximum entropy, smoothest, least change, and least change from prior solutions. If the sensitivities are reliable then the smoothest solution does not honour their relative magnitudes. For this example, the least change images and the maximum entropy image are quite similar and, considering the sensitivities, appear to have made sensible reconstructions. If prior knowledge is available that suggests that the image should be smooth to some degree, this can be built into the constraint statistic and the maximum entropy principle can still be applied.

The next example is slightly more complex and demonstrates two important features of sensible reconstructions. Consider the nine cell, 2-dimensional problem of Figure 3.6-1. For the required C reduction of 9 units, the 4 chosen solutions have been generated (see Figure 3.6-2). The maximum entropy image (Figure 3.6-2(d)) is perfectly smooth, as it should be since each cell has exactly the same sensitivity and the prior information model is smooth. The least change solution (Figure 3.6-2(b)) actually ~~increases~~ maintains the difference between the top right cell and the other cells. This method is too greatly affected by earlier changes of the velocities. This is a serious shortfall since it is common for cells to be erroneously changed by early iterations. The top right cell may have been updated by an earlier iteration which had its rays incorrectly placed, and the least-squares minimum change criterion (which is the criterion used by the popular SIRT algorithm (see Stewart, 1989)) is incapable of correcting this as the ray locations improve. Such incorrect updates will appear as spurious additional structure on the final image. For this example, both the least change from prior and smoothest solutions (Figures 3.6-2(c) and 3.6-2(a)) are identical

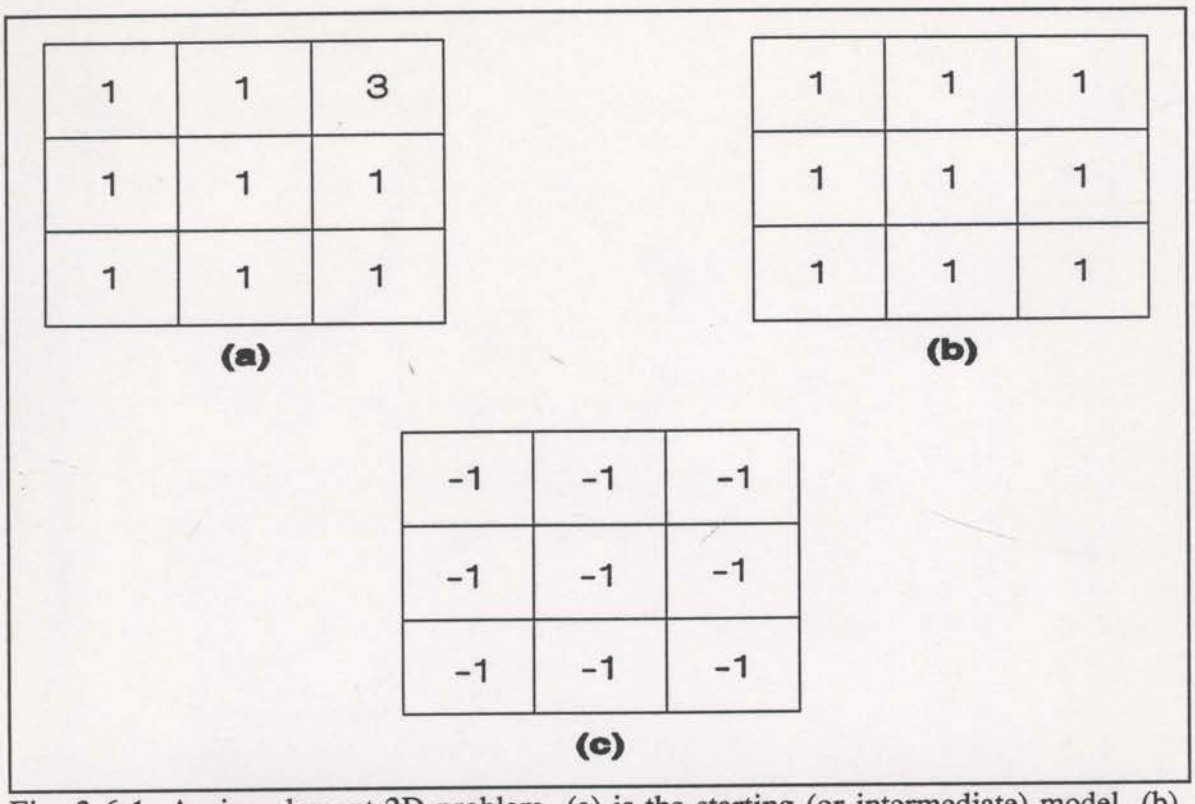


Fig. 3.6-1 A nine element 2D problem. (a) is the starting (or intermediate) model, (b) is the prior information model, and (c) contains the sensitivities of the constraint statistic. The required reduction of C is 9 units.

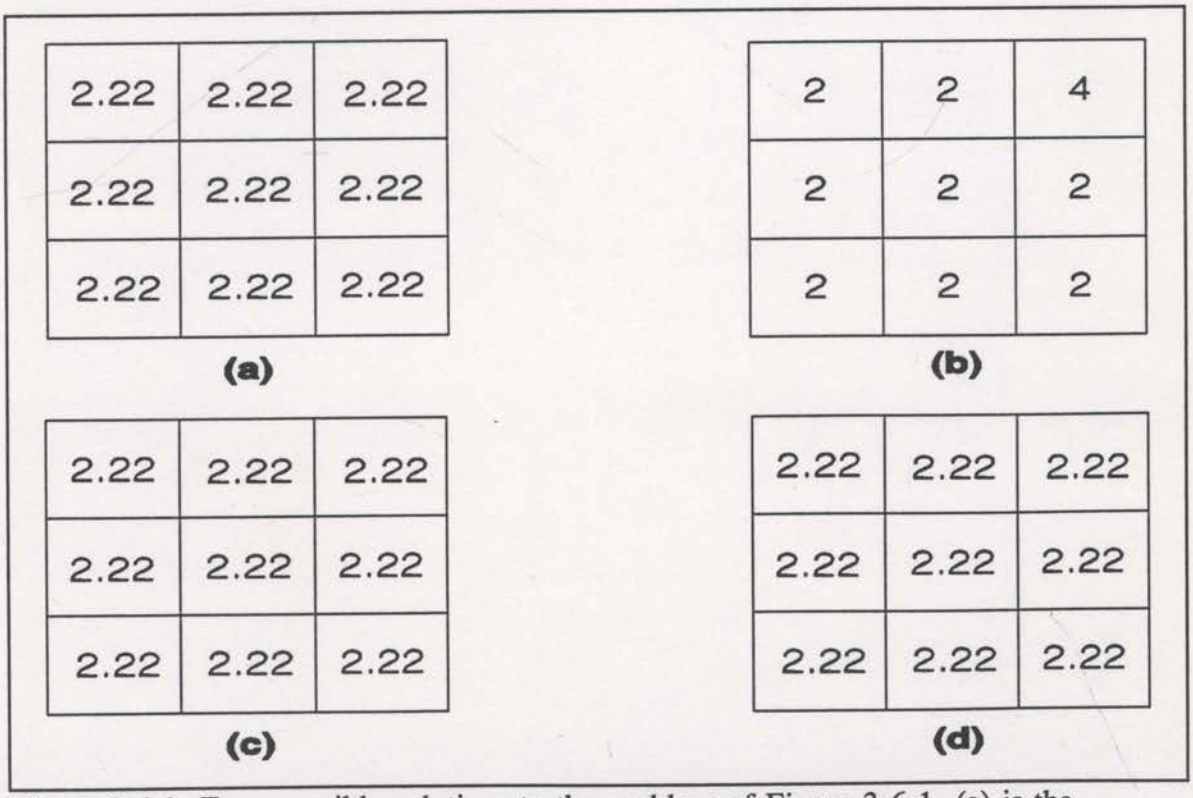


Figure 3.6-2 Four possible solutions to the problem of Figure 3.6-1. (a) is the smoothest solution, (b) is the least squares minimum change solution, (c) has the minimum change from the prior, and (d) is the maximum entropy image.

A further two-dimensional example is depicted in Figure 3.6-3. On this occasion the sign of the sensitivities create negative velocity values for all reconstructed images except the maximum entropy image (see Figure 3.6-4). The negative values can be avoided by imposing a minimum velocity (eg. Carrion et al., 1993), but this would be just another source of arbitrariness. The maximum entropy solution guarantees positivity.

In all of the above examples the prior information model was a uniform one and so did not supply any structural information as the prior velocity of each cell was set to unity. To see how prior information can help with the reconstruction, consider the problem of Figure 3.6-5. The starting model here is set equal to the prior model and suggests a velocity increase with depth (see Figure 3.6-5(a)). Given the sensitivities of Figure 3.6-5(b), the maximum entropy image is shown in Figure 3.6-6. Even though the cell at the bottom of the model had a larger positive sensitivity (positive sensitivities will reduce V_{ij}), its final velocity is larger than that of the cell at the top of the model because of the different prior values. One way to view the use of prior information is to notice that the maximum entropy reconstruction is not concerned with the actual velocity values, but rather with the ratio of velocity and prior model velocity, V_{ij}/m_{ij} (see equation (3.5-3)). Alternatively, rearranging equation (3.5-3),

$$V_{ij} = m_{ij}(e^{\lambda})^{\frac{\partial C}{\partial V_{ij}}} = \text{prior}(\text{const})^{\text{sensitivity}} . \tag{3.6-1}$$

Once λ is chosen to satisfy the constraints, e^{λ} becomes a constant for the whole image. The magnitude and sign of the cell's sensitivity then exponentiates this

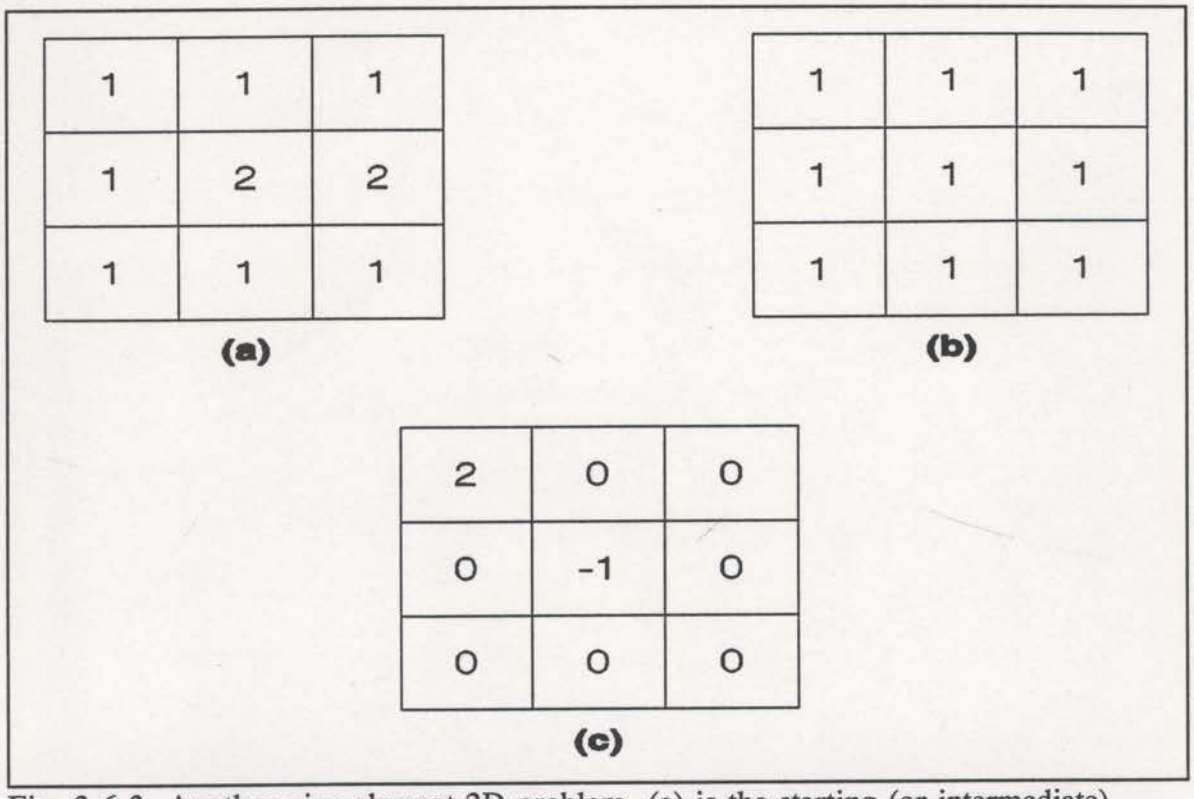


Fig. 3.6-3 Another nine element 2D problem. (a) is the starting (or intermediate) model, (b) is the prior information model, and (c) contains the sensitivities of the constraint statistic, C. The required C reduction is 8 units.

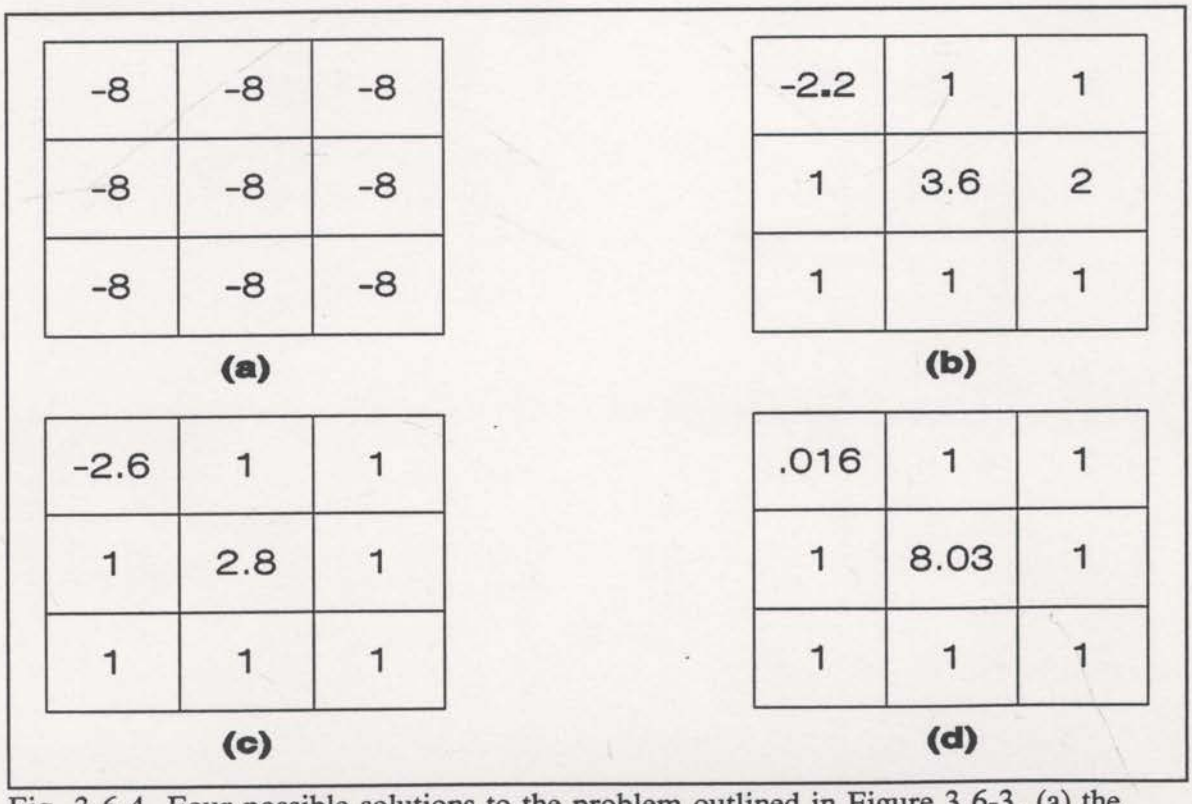


Fig. 3.6-4 Four possible solutions to the problem outlined in Figure 3.6-3. (a) the smoothest, (b) least-squares minimum change, (c) nearest to prior, and (d) maximum entropy.

constant, and the prior value is a simple multiplier. As a cell sensitivity approaches zero (little or no information) then V_{ij} approaches the prior value. The prior acts as both a default value and as a relative scaler for the image values. Relevant prior information model values can help keep the inversion in good shape through the iterations, especially when some poor sensitivity estimates appear.

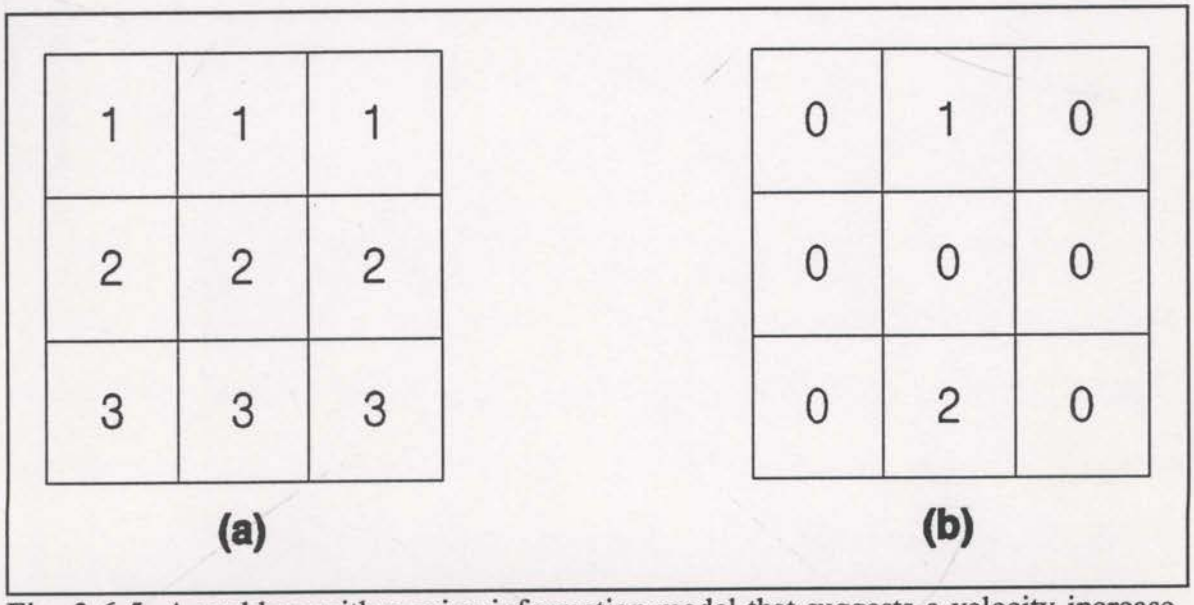


Fig. 3.6-5 A problem with a prior information model that suggests a velocity increase with depth. (a) is the starting and prior model, (b) contains the sensitivities of the constraint statistic. The required C reduction is 4 units.

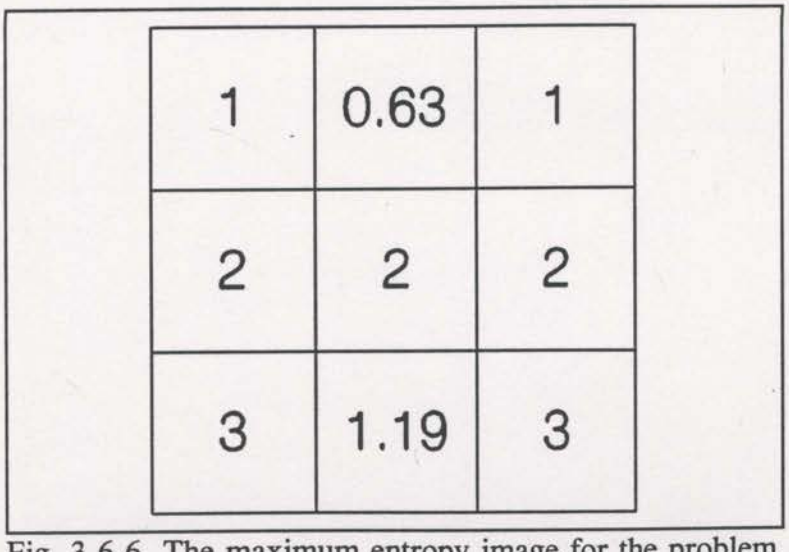


Fig. 3.6-6 The maximum entropy image for the problem of Figure 3.6-5

3.7 Synthetic Data Results

Figure 3.7-1 is the velocity/depth model used in a synthetic crosswell tomography problem. Sources were located at a 20 metre spacing down the left side of the model and receivers were similarly spaced down the right side, resulting in 676 raypaths in total. The data were initially inverted with the conventional SIRT algorithm (see Stewart, 1989) which finds the solution image with least-squares minimum change. The SIRT reconstruction is shown in Figure 3.7-2. The corresponding maximum entropy reconstruction, using a constant 2000 m/s prior model, is shown in Figure 3.7-3. Notice how all 3 anomalies ~~are~~ may be more apparent in the maximum entropy reconstruction (the maximum entropy inversion algorithm used is based on that of Skilling and Bryan, 1984; see section 4). Also, the velocities are nearer to the correct values. Away from the anomalies, the maximum entropy image also exhibits less

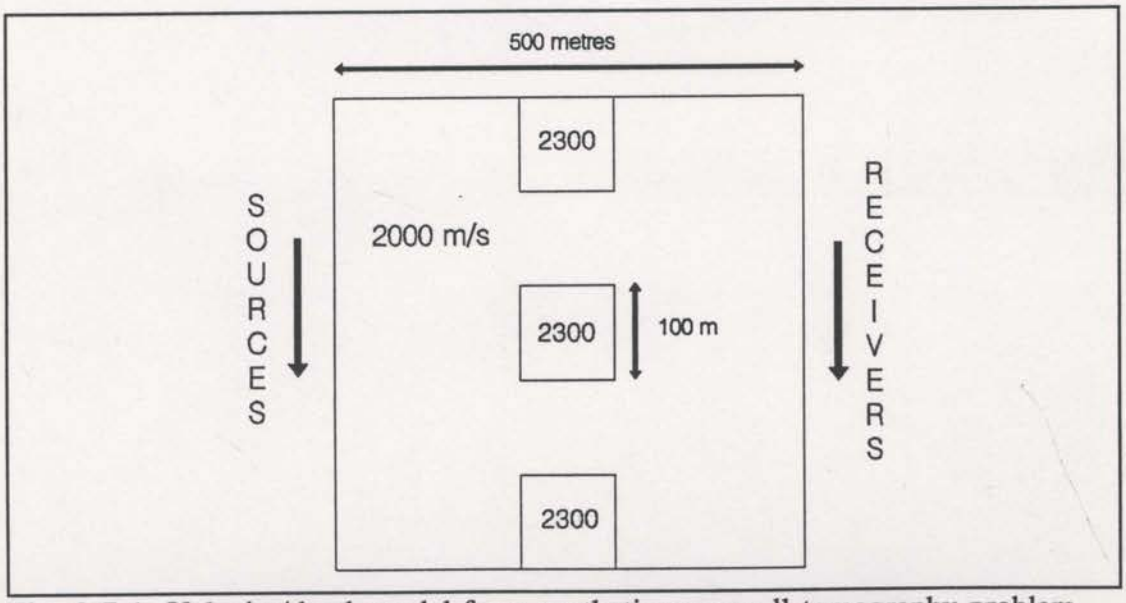


Fig. 3.7-1 Velocity/depth model for a synthetic crosswell tomography problem. Sources and receivers separated by 20 metres; total of 676 raypaths.

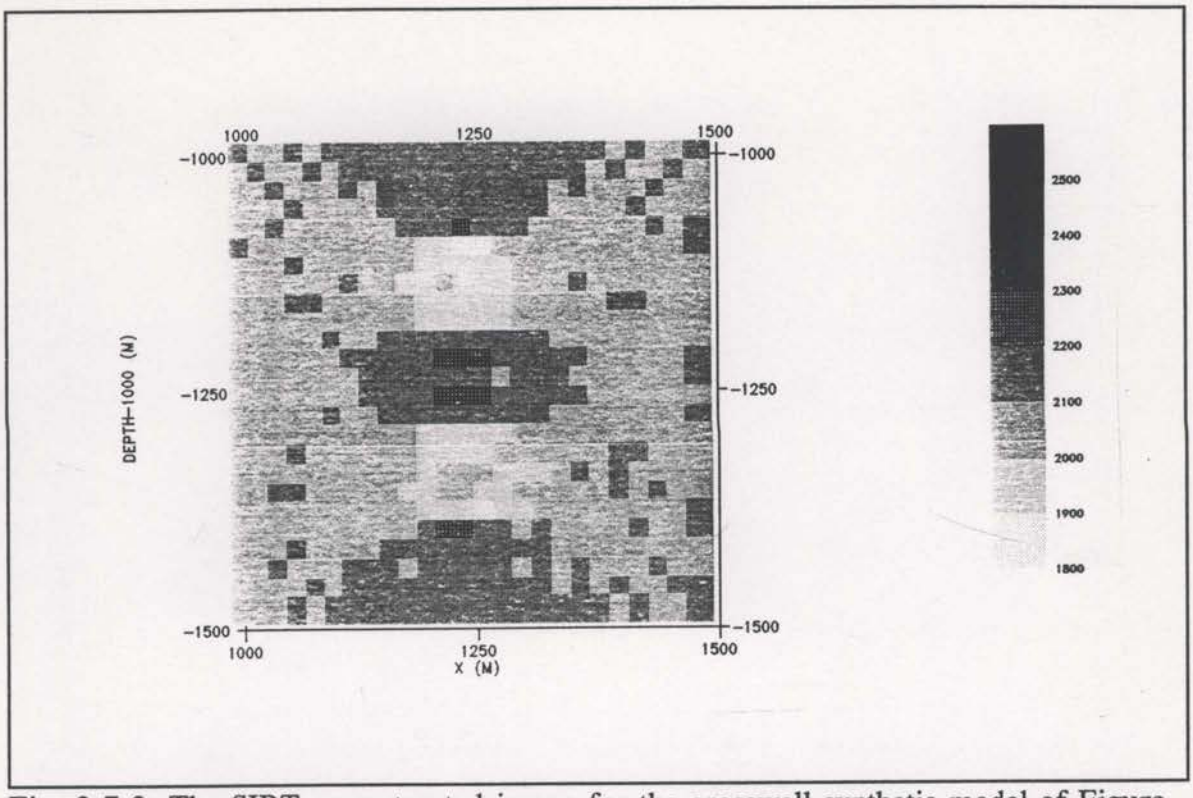


Fig. 3.7-2 The SIRT reconstructed image for the crosswell synthetic model of Figure 3.7-1

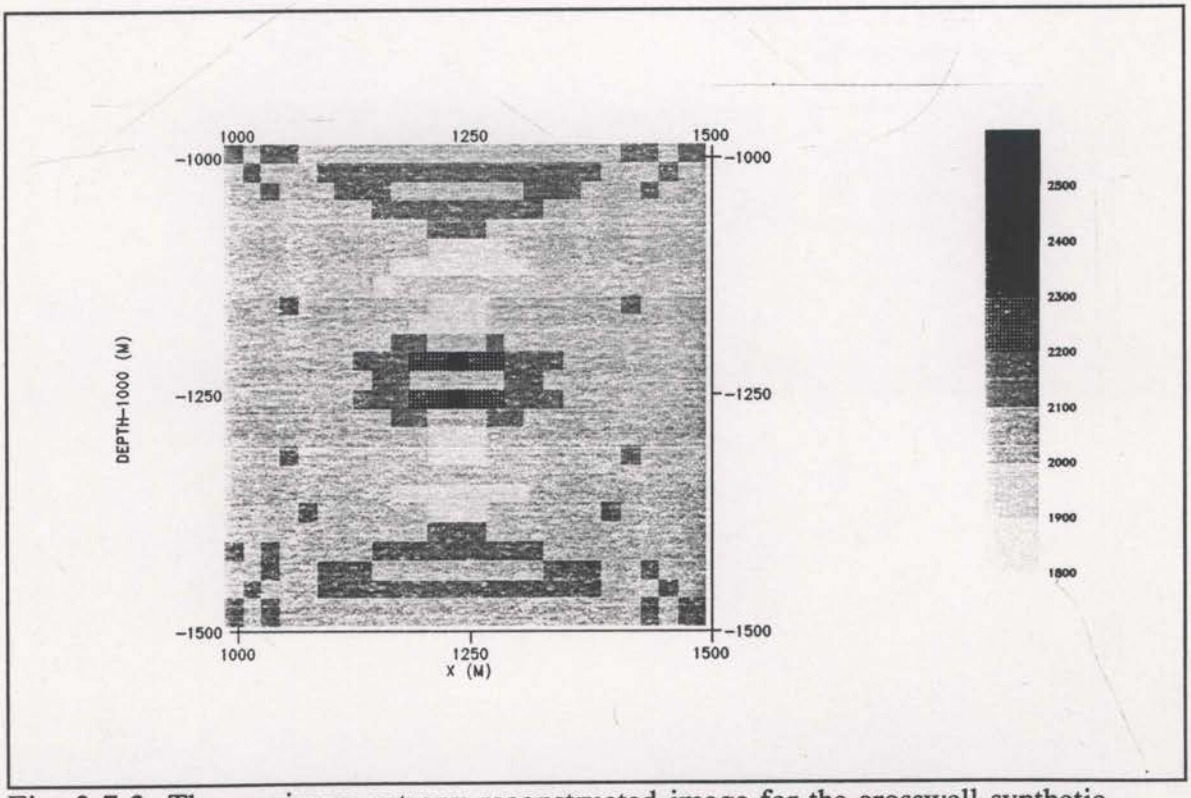


Fig. 3.7-3 The maximum entropy reconstructed image for the crosswell synthetic model of Figure 3.7-1

spurious structure since each maximum entropy iteration effectively questions the changes made in earlier iterations. If there is no longer justification for the change, it tends to reset the velocity of the cell back to that of the prior model.

An example of reflection tomography (Whiting, 1991b), both with and without maximum entropy constraints, will be demonstrated using the model of Figure 3.7-4. This model consists of three constant velocity layers with a high angle fault roughly in the centre of the model. The starting model and the prior information model consisted of the three layers without the fault and with a smooth velocity transition from one layer to the next. The results of two inversions can be seen in Figures 3.7-5 and 3.7-6, the former being the reconstruction without any entropy consideration and the latter being the maximum entropy image. Notice that the maximum entropy image has less unwarranted structure. The only "bumps" occurring in the layer boundaries are those associated with the faults.

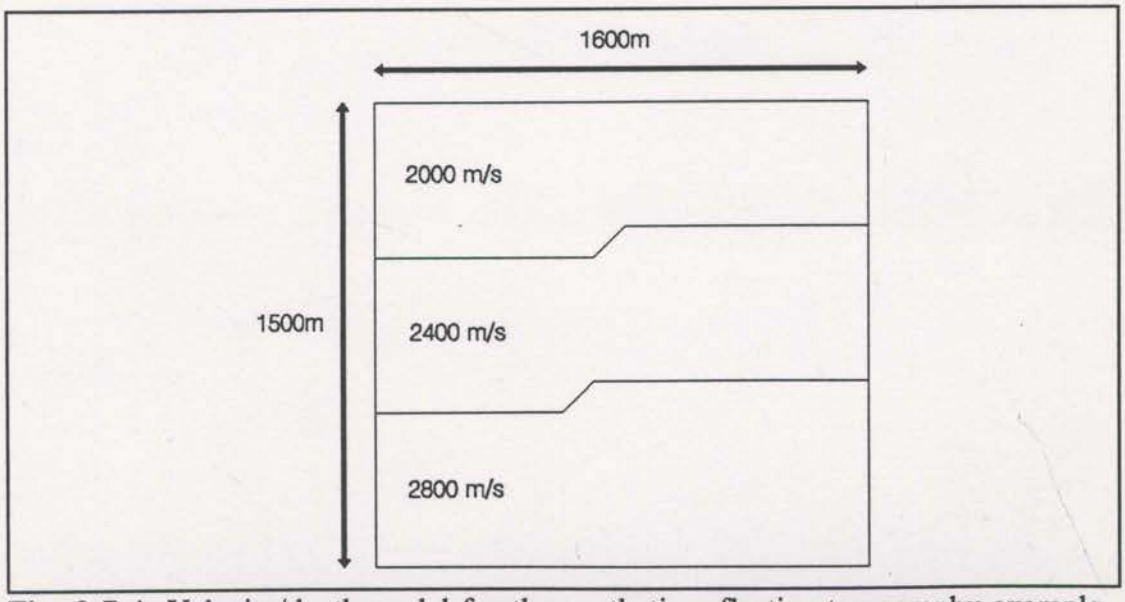


Fig. 3.7-4 Velocity/depth model for the synthetic reflection tomography example. 800 raypaths were used with random source/receiver locations and random reflection points.

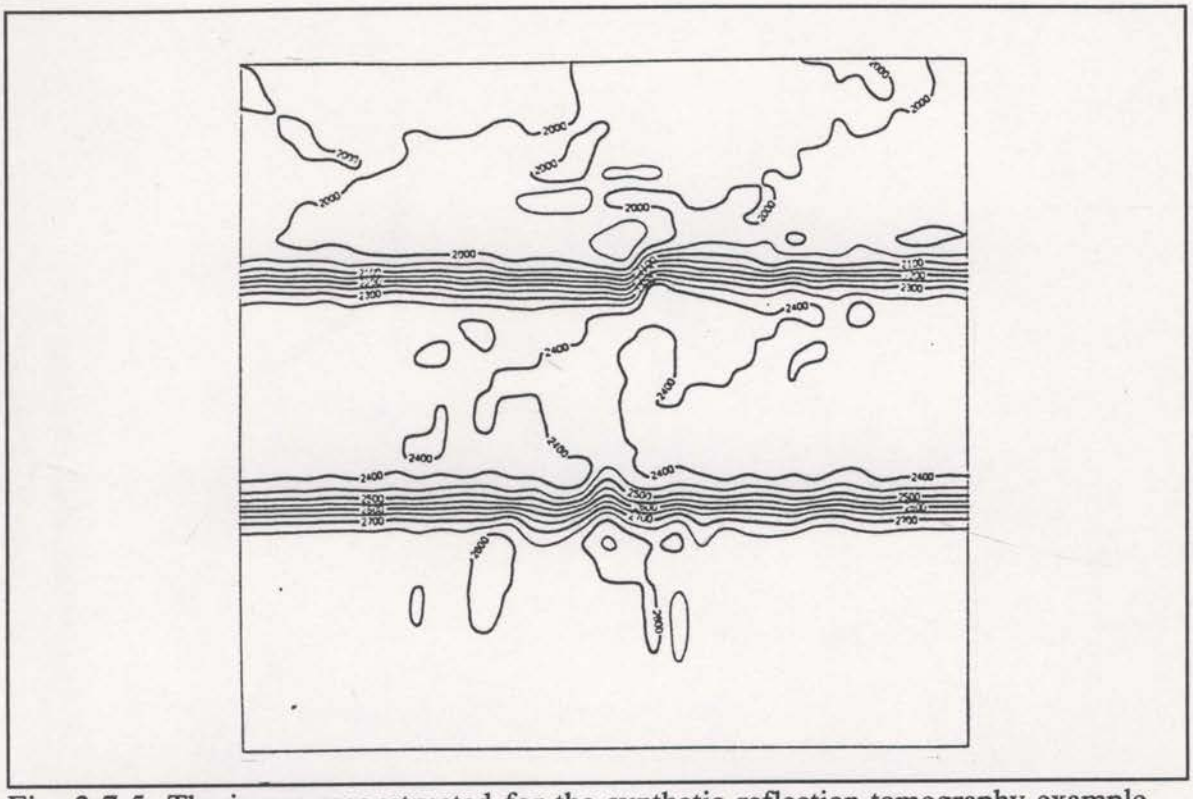


Fig. 3.7-5 The image reconstructed for the synthetic reflection tomography example without entropy constraints.

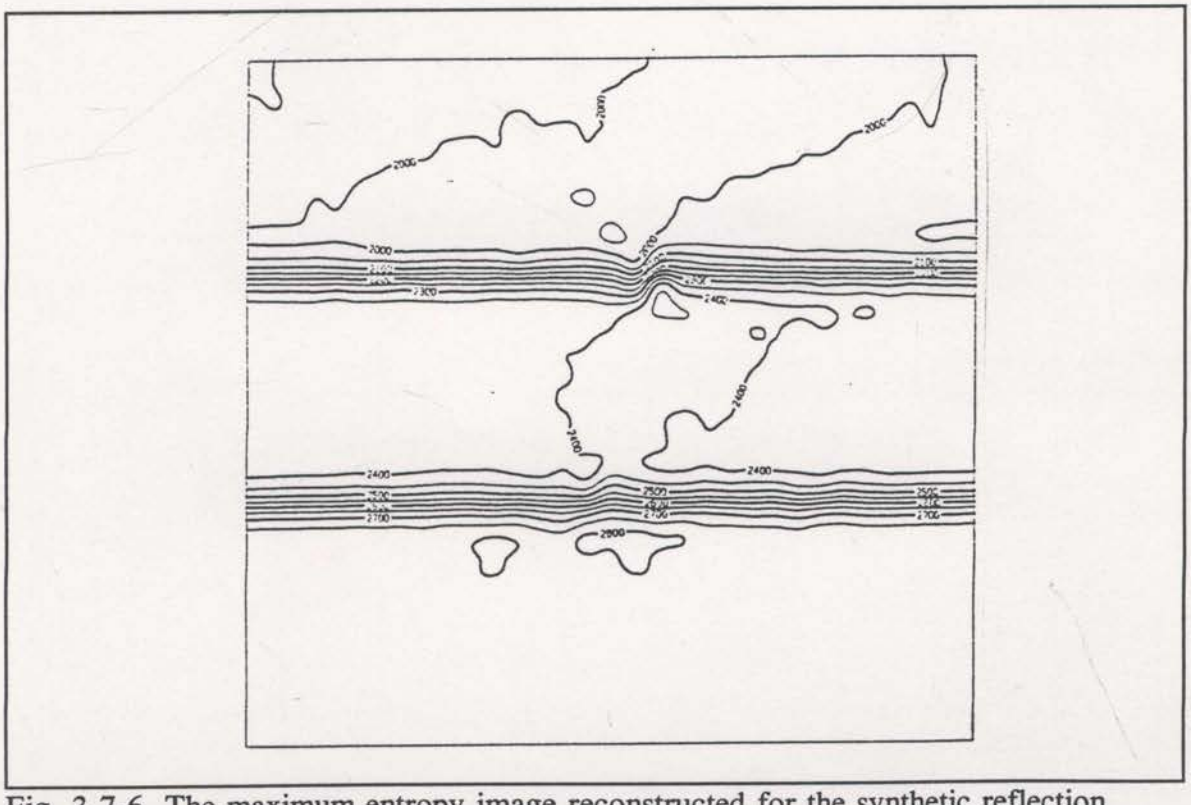


Fig. 3.7-6 The maximum entropy image reconstructed for the synthetic reflection tomography example.

3.8 Discussion

The entropy principle has a long and successful history. The same theory as developed in statistical mechanics and information theory can be used, with suitable adjustments, in tomographic image reconstruction. The reason for the universality is that the entropy principle is basically an expression of "common sense". When faced with a range of possible solutions, entropy considerations suggest that the solution that has the most ways of actually occurring should be chosen. If a solution is chosen that has fewer ways of actually occurring, then it is implicit that more information is being assumed. This notion is consistent with the concept of "least commitment" in tomographic image reconstruction. Choosing an image with more commitment to individual cells, or structure, implicitly assumes that more information is known. Maximising entropy ensures that only as much information as is possible (given the current prior information) is extracted from the data, and no more.

The entropy function (introduced in tomographic image reconstruction with the assistance of contingency table analysis) ensures that the velocities are always positive and that the reconstructed values reflect both the size of the constraint statistic sensitivities and the structure of the prior information model. In this way, as long as the sensitivities are mostly noise free and the inversion is not trapped by a local minimum, a useful and relevant image can be reconstructed. Examples of reconstructions of a crosswell and a reflection traveltime tomographic inversion have demonstrated these features and the usefulness of the entropy principle in tomographic image reconstruction.

§4 THE ENTROPY PRINCIPLE - APPLICATION

4.1 The basic maximum entropy inversion algorithm.

Inversion of traveltime data by the entropy principle attempts to minimise a residual traveltime error statistic while maintaining maximum entropy. This is a simultaneous optimisation problem. Such a problem is of considerable complexity because the model space is an n-dimensional space commonly of much more than 10,000 dimensions. Computational efficiency is paramount under these circumstances and matrix techniques are not feasible. The algorithm used in this study is based on that described by Skilling and Bryan (1984). They developed a general algorithm but their primary objective was astronomical imaging. This section outlines the basic strategy of this algorithm. The following sections will describe the additions and modifications that have been made for the current study.

The general problem to be solved by the maximum entropy inversion algorithm is to find model parameters, $\mathbf{q}$, that best predict the given data, $\mathbf{d}$, given a theoretical relationship (generally non-linear) of the form,

$$\mathbf{d} = \mathbb{R}(\mathbf{q}) \quad . \tag{4.1-1}$$

Here, $\mathbf{d}$ represents an m-vector of data values, $\mathbf{q}$ represents an n-vector of model parameters, and $\mathbb{R}$ represents a mapping from model space to the data space. Generally, this non-linear operator $\mathbb{R}$ is linearised and the inversion process is

performed in an iterative fashion. In this case, equation (4.1-1) can be written as,

$$\delta d_i = \sum_j R_{ij} \delta q_j \quad , \tag{4.1-2}$$

a standard linear matrix equation, where the matrix R_{ij} represents the linear approximation of $\mathbb{R}$, and the δd_i, δq_j notation implies that this linear approximation is generally only valid for small changes of the parameters. As the problem is solved in a series of small steps, a constraint statistic, C, is introduced to ensure that the model parameters are continually being changed toward those that most accurately predict the data. The simplest form of this constraint statistic is just the data residual,

$$C = (\mathbf{d} - \mathbb{R}(\mathbf{q}))^T C_d^{-1} (\mathbf{d} - \mathbb{R}(\mathbf{q})) \quad , \tag{4.1-4}$$

(which assumes Gaussian statistics) where $\mathbf{C}_d$ is the data covariance matrix. The iterative technique then attempts to reduce C at every iteration until the minimum is located. If the number of data and model parameters are small, the matrix R_{ij} will have reasonable dimension and it is feasible to invert it using standard linear algebra. The inverted matrix can then be used to give the updated model parameters for each iteration directly,

$$R^{-1} \delta \mathbf{d} = \delta \mathbf{q} \quad . \tag{4.1-3}$$

However, the problem is rarely well conditioned and the matrix is often not invertible. Also, the size of the R matrix quickly makes matrix operations impractical.

The role of the inversion is to find the global minimum of a non-linear function (ie. the constraint statistic, C) within an n-dimensional space, where 'n' is very large ($>10{,}000$). An iterative approach to finding this global minimum begins with a set of

initial model parameters which defines a single point in this n-space. At this location, it is necessary to estimate ∇C and construct a local linear approximation to the C-surface that can be used to move to a nearby point with a (hopefully) smaller value of C. A new local linear approximation is made and the next small step is taken. The procedure iterates in this fashion until a minimum of the constraint statistic is located. Optimisation algorithms that work in this way are called 'descent methods'. Commonly used variants are the 'steepest descent' and 'conjugate gradient' techniques.

If the second-order gradient of C, $\nabla\nabla C$, can be estimated, the linear approximation can be upgraded to a locally quadratic approximation. Quadratic approximations help provide reasonable limits on the iterative step distance. With just ∇C it is difficult to decide on the size of model parameter change that can be accommodated by each iteration. It is generally difficult to compute $\nabla\nabla C$ but approximations to it also prove to be useful. The conjugate gradient technique attempts to build up information about $\nabla\nabla C$ as the iterations proceed.

Most descent methods have a criterion for choosing a single search direction in n-space based on C, ∇C and possibly $\nabla\nabla C$ at the current location. In between each iteration, new data estimates are computed using equation (4.1-1) and ∇C and $\nabla\nabla C$ must be re-evaluated. This inter-iteration computing is significant and it is more efficient and sensible to use more than just one search direction for each iteration. This suggests searching in a subspace rather than along a single line. Subspace inversion methods are not new in geophysics (see Kennett et al., 1988). The results of a subspace search can be no worse than a linear search as the direction of linear search

is included within the subspace. Skilling and Bryan (1984) suggest a 4 or 6 dimensional subspace.

Optimisation with the maximum entropy principle requires the minimisation of C while maximising the entropy function, S (see section 3). Skilling and Bryan (1984) suggest simultaneous optimisation of these functions[1]. Successful simultaneous optimisation relies on the subspace to provide reasonable approximations of both S and C. Inspired by the conjugate gradient method, Skilling and Bryan (1984) choose the following six vectors for the definition of the subspace,

$$\begin{aligned} &\nabla S,\ \nabla\nabla C.\nabla S,\ \nabla\nabla C.\nabla\nabla C.\nabla S \\ &\nabla C,\ \nabla\nabla C.\nabla C,\ \nabla\nabla C.\nabla\nabla C.\nabla C\ . \end{aligned} \tag{4.1-5}$$

Even though $\nabla\nabla C$ is potentially a very large matrix, each of these vectors can be derived by vector operations only.

Note that $\nabla\nabla S$ does not appear in these definitions (4.1-5). This is because Skilling and Bryan (1984) introduce an "entropy metric". In their astronomical application the model parameters can have very small values tending toward zero. To help avoid some of these parameters assuming negative values Skilling and Bryan measure distances as,

$$\sum_i \frac{\delta q_i^2}{q_i} \tag{4.1-6}$$

where δq_i represents a change of the i th model parameter. This is equivalent to a

---

[1] It is possible to construct a single function, H, using a Lagrange multiplier, λ (ie. $H = C - \lambda S$). Skilling and Bryan (1984) claim that simultaneous optimisation is easier because of complications in finding the required value of λ.

metric of $\frac{1}{q_i}$ which can be shown to be equal to $-\nabla\nabla S$. Skilling and Bryan call this the entropy metric. This metric is of little importance in subsurface interval velocity imaging since the dynamic range of velocity variations is quite small. The effect of using this metric is to modify the subspace vectors (4.1-5) to,

$$\begin{aligned} &q(\nabla S),\ q(\nabla\nabla C.\nabla S),\ q(\nabla\nabla C.\nabla\nabla C.\nabla S) \\ &q(\nabla C),\ q(\nabla\nabla C.\nabla C),\ q(\nabla\nabla C.\nabla\nabla C.\nabla C) \end{aligned} \tag{4.1-7}$$

where the $q()$ notation implies component by component multiplication. Now it is clear that $q(\nabla\nabla S) = I$ and would add nothing if included in the subspace vector definition. These six vectors (4.1-7) define the subspace to be used, however, it is common to use only a 4-dimensional subspace with the first two vectors of each type.

The subspace vectors can be organised into an (n x r)-matrix, E, that maps vectors from the original n-dimensional space to the r-dimensional subspace,

$$\mathbf{E}^{\mathrm{T}}\mathbf{q} = \mathbf{y} \ . \tag{4.1-8}$$

In the n-dimensional model space, the original image $\mathbf{q}_0$ is to be updated to a new image $\mathbf{q}$,

$$\mathbf{q} = \mathbf{q}_0 + \boldsymbol{\delta}\mathbf{q} \ . \tag{4.1-9}$$

The constraint statistic, C, and the entropy, S, surfaces can be expressed as,

$$S = S_0 + \boldsymbol{\delta}\mathbf{q}^{\mathrm{T}}\nabla S_0 + \frac{1}{2}\boldsymbol{\delta}\mathbf{q}^{\mathrm{T}}\nabla\nabla S_0\boldsymbol{\delta}\mathbf{q} + \textit{higher order terms} \tag{4.1-10}$$

$$C = C_0 + \boldsymbol{\delta}\mathbf{q}^{\mathrm{T}}\nabla C_0 + \frac{1}{2}\boldsymbol{\delta}\mathbf{q}^{\mathrm{T}}\nabla\nabla C_0\boldsymbol{\delta}\mathbf{q} + \textit{higher order terms} \tag{4.1-11}$$

where C_0 and S_0 are the values of C and S at $\mathbf{q}_0$. Equation (4.1-8) provides the

[5] $D_\beta = \mathrm{diag}[\beta_1, \beta_2, \ldots]$ where $\beta_1, \beta_2, \ldots$ are the eigenvalues of B.

means for mapping these surfaces to the r-dimensional subspace. If the higher order terms of equations (4.1-10) and (4.1-11) are ignored, quadratic approximations result,

$$\bar{S}(\mathbf{y}) = S_0 + \mathbf{y}^T\mathbf{a} - \frac{1}{2}\mathbf{y}^T\mathbf{A}\mathbf{y} \tag{4.1-12}$$

$$\bar{C}(\mathbf{y}) = C_0 + \mathbf{y}^T\mathbf{b} + \frac{1}{2}\mathbf{y}^T\mathbf{B}\mathbf{y} \quad , \tag{4.1-13}$$

where,

$$\mathbf{a} = \mathbf{E}^T\nabla S_0 \quad , \quad \mathbf{A} = \mathbf{E}^T(-\nabla\nabla S_0)\mathbf{E} \quad ,$$

$$\mathbf{b} = \mathbf{E}^T\nabla C_0 \quad , \quad \mathbf{B} = \mathbf{E}^T(\nabla\nabla C_0)\mathbf{E} \quad .$$

These approximations are the basis of the simultaneous minimisation of the constraint statistic and maximisation of entropy.

The mathematical manipulation of the quadratic approximations within the subspace (equations (4.1-12) and (4.1-13)) is simplified if the matrices $\mathbf{A}$ and $\mathbf{B}$ are simultaneously diagonalised. A detailed examination of this process is left for section 4.2. As will be seen in section 4.2, a transformation is applied to the subspace such that equations (4.1-12) and (4.1-13) become,

$$\bar{S}(\mathbf{x}) = S_0 + \mathbf{x}^T\mathbf{a}' - \frac{1}{2}\mathbf{x}^T\mathbf{x} \tag{4.1-14}$$

$$\bar{C}(\mathbf{x}) = C_0 + \mathbf{x}^T\mathbf{b}' + \frac{1}{2}\mathbf{x}^T\mathbf{D}_\beta\mathbf{x} \quad , \tag{4.1-15}$$

where $\mathbf{x}$, $\mathbf{a}'$ and $\mathbf{b}'$ are the transformed versions of $\mathbf{y}$, $\mathbf{a}$ and $\mathbf{b}$ respectively, and $\mathbf{D}_\beta$ is a diagonal matrix.[5] Notice that $\mathbf{A}$ has been transformed to the identity matrix and $\mathbf{B}$ has been diagonalised. These equations simplify the mathematics considerably.

It is now possible to locate the minimum value of the quadratic approximation, $\bar{C}(\mathbf{x})$, with respect to the r-dimensional subspace. Set the derivative with respect to $\mathbf{x}$

means for mapping these surfaces to the r-dimensional subspace. If the higher order terms of equations (4.1-10) and (4.1-11) are ignored, quadratic approximations result,

$$\bar{S}(\mathbf{y}) = S_0 + \mathbf{y}^T\mathbf{a} - \frac{1}{2}\mathbf{y}^T\mathbf{A}\mathbf{y} \tag{4.1-12}$$

$$\bar{C}(\mathbf{y}) = C_0 + \mathbf{y}^T\mathbf{b} + \frac{1}{2}\mathbf{y}^T\mathbf{B}\mathbf{y} \quad , \tag{4.1-13}$$

where,

$$\mathbf{a} = \mathbf{E}^T\nabla S_0 \quad , \quad \mathbf{A} = \mathbf{E}^T(-\nabla\nabla S_0)\mathbf{E} \quad ,$$

$$\mathbf{b} = \mathbf{E}^T\nabla C_0 \quad , \quad \mathbf{B} = \mathbf{E}^T(\nabla\nabla C_0)\mathbf{E} \quad .$$

These approximations are the basis of the simultaneous minimisation of the constraint statistic and maximisation of entropy.

The mathematical manipulation of the quadratic approximations within the subspace (equations (4.1-12) and (4.1-13)) is simplified if the matrices A and B are simultaneously diagonalised. A detailed examination of this process is left for section 4.2. As will be seen in section 4.2, a transformation is applied to the subspace such that equations (4.1-12) and (4.1-13) become,

$$\bar{S}(\mathbf{x}) = S_0 + \mathbf{x}^T\mathbf{a}' - \frac{1}{2}\mathbf{x}^T\mathbf{x} \tag{4.1-14}$$

$$\bar{C}(\mathbf{x}) = C_0 + \mathbf{x}^T\mathbf{b}' + \frac{1}{2}\mathbf{x}^T\mathbf{D}_\beta\mathbf{x} \quad , \tag{4.1-15}$$

where $\mathbf{x}$, $\mathbf{a}'$ and $\mathbf{b}'$ are the transformed versions of $\mathbf{y}$, $\mathbf{a}$ and $\mathbf{b}$ respectively, and $\mathbf{D}_\beta$ is a diagonal matrix.[5] Notice that A has been transformed to the identity matrix and B has been diagonalised. These equations simplify the mathematics considerably.

It is now possible to locate the minimum value of the quadratic approximation, with respect to $\mathbf{x}$

[5] $\mathbf{D}_\beta = \mathrm{diag}[\beta_1, \beta_2, \ldots]$ where $\beta_1, \beta_2, \ldots$ are the eigenvalues of B.

to zero,

$$\frac{\partial \bar{C}}{\partial \mathbf{x}} = \mathbf{b}' + \mathbf{D}_\beta \mathbf{x} = 0 \quad .$$

Therefore,

$$\mathbf{X}_{\min} = \frac{-\mathbf{b}'}{\mathbf{D}_\beta} = -\mathbf{D}_{\frac{1}{\beta}} \mathbf{b}' \tag{4.1-16}$$

is the subspace location of the constraint statistic minimum. This can be substituted into equation (4.1-15) to give the actual value of the constraint statistic minimum,

$$\begin{aligned} \bar{C}_{\min} &= C_0 + \left(-\mathbf{b}'\mathbf{D}_{\frac{1}{\beta}}\right)^{\mathbf{T}} \mathbf{b}' + \frac{1}{2}\left(-\mathbf{b}'\mathbf{D}_{\frac{1}{\beta}}\right)^{\mathbf{T}} \mathbf{D}_\beta \left(-\mathbf{b}'\mathbf{D}_{\frac{1}{\beta}}\right) \\ &= C_0 - \mathbf{b}'^{\mathbf{T}} \mathbf{D}_{\frac{1}{\beta}} \mathbf{b}' + \frac{1}{2}\mathbf{b}'^{\mathbf{T}} \mathbf{D}_{\frac{1}{\beta}} \mathbf{b}' \\ &= C_0 - \frac{1}{2}\mathbf{b}'^{\mathbf{T}} \mathbf{D}_{\frac{1}{\beta}} \mathbf{b}' \quad . \end{aligned} \tag{4.1-17}$$

The strategy is to now choose a value, $\bar{C}_{aim}$, as the constraint statistic value to be aimed at by this iteration. Choosing $\bar{C}_{aim}$ less than $\bar{C}_{\min}$ would be foolish. Even choosing $\bar{C}_{aim} = \bar{C}_{\min}$ is not ideal as this would leave no room for maximising entropy. Skilling and Bryan (1984) use the value,

$$\bar{C}_{aim} = \frac{2}{3}\bar{C}_{\min} + \frac{1}{3}C_0 \quad , \tag{4.1-18}$$

that is, two-thirds of the way towards the minimum of $\bar{C}(\mathbf{x})$.[2]

Within the subspace, there will be many points with the constraint statistic

[2] Other proportions were tested in this study but this specification generally gave the best results.

value of $\bar{C}_{aim}$, but the one with maximum entropy must be found[3]. A new function is constructed to help find this point,

$$\bar{H}(\mathbf{x}) = \alpha \bar{S}(\mathbf{x}) - \bar{C}(\mathbf{x}) \quad , \tag{4.1-19}$$

where α is a Lagrange multiplier. This function is to be maximised, so,

$$\frac{\partial \bar{H}}{\partial \mathbf{x}} = \alpha \frac{\partial \bar{S}}{\partial \mathbf{x}} - \frac{\partial \bar{C}}{\partial \mathbf{x}} = \alpha(\mathbf{a}' - \mathbf{x}) - (\mathbf{b}' + \mathbf{D}_\beta \mathbf{x}) = 0$$

which gives,

$$\mathbf{x} = \mathbf{D}_{\frac{1}{\beta+\alpha}} (\alpha \mathbf{a}' - \mathbf{b}') \quad . \tag{4.1-20}$$

As α varies, this equation defines the maximum entropy trajectory through the subspace. Following this trajectory will always result in maximum entropy points. It is possible to search along this trajectory until the point with value $\bar{C}_{aim}$ is found. Figure 4.1-1 schematically demonstrates the significance of the points $\mathbf{x}$ within the subspace. When $\alpha = \infty$ in equation (4.1-20), $\mathbf{x} = \mathbf{a}'$, and when $\alpha = 0$, $\mathbf{x} = -\mathbf{D}_{\frac{1}{\beta}} \mathbf{b}'$. These vectors define the maximum of entropy and the minimum of the constraint statistic and represent the ends of the maximum entropy trajectory.

Skilling and Bryan (1984) implemented a simple "α-chop" algorithm for finding the required point on the maximum entropy trajectory. This algorithm steps through α values until a value is found that results in a constraint statistic very close to $\bar{C}_{aim}$. This is stable because $\bar{C}$ varies monotonically along the maximum entropy

---

[3] It is possible for the maximum entropy point to be multi-valued (see Skilling and Bryan, 1984). This possibility does not create any problems though.

trajectory due to the quadratic approximation. Even so, the quadratic approximation will not be valid too far away from the origin of the subspace coordinates[4]. For this reason, a distance limit, L, is imposed. If $\bar{C}_{aim}$ cannot be achieved within the distance limit, then the intersection of the distance limit and the smallest $\bar{C}$ value on the maximum entropy trajectory is chosen. A more severe problem can occur (rarely) if the current model is poor. There may be no point on the maximum entropy trajectory within the distance limit. This case is handled (see Skilling and Bryan, 1984) by including an extra term to the function $\bar{H}(\mathbf{x})$ which includes the length of $\mathbf{x}$ and an extra Lagrange multiplier. The Lagrange multiplier is normally kept at a zero value and is only increased in the rare case of having no valid points within the distance limit.

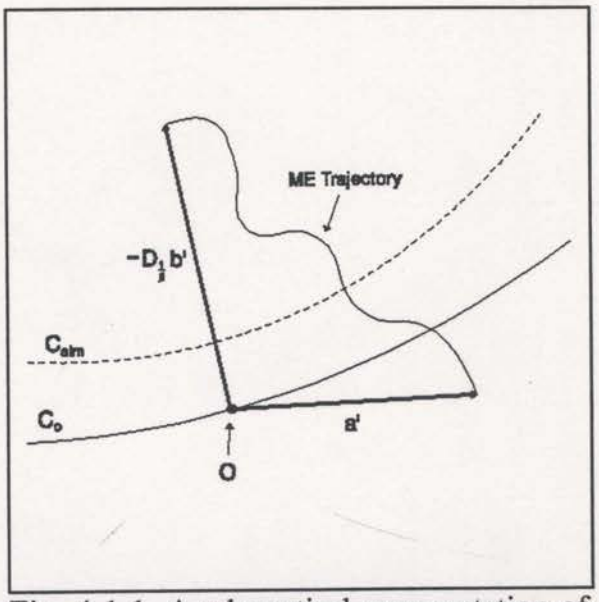


Fig. 4.1-1 A schematical representation of the subspace showing the maximum entropy trajectory.

Once the required point $\mathbf{x}$ has been chosen in the subspace, it can be mapped back to the full n-dimensional space. This is done using the mapping of equation (4.1-8) after suitable transformation to account for the simultaneous diagonalisation (see section 4.2). The projection of $\mathbf{x}$ into the full space gives the updated model for this iteration.

A very important parameter used for checking the performance of the maximum

[4] This is the location of the current model. Recall equations (4.1-10) and (4.1-11).

NB. In this figure, the gradient of S is pointing in the direction of steepest ascent whilst the gradient of C is pointing in the direction of steepest descent.

entropy inversion algorithm is the TEST parameter (see Skilling and Bryan, 1984). Figure 4.1-2 shows that the gradients of the entropy surface and the constraint statistic surface must be parallel at the true maximum entropy point. The TEST parameter measures the degree of non-parallelism and is defined as,

$$\text{TEST} = \frac{1}{2}\left|\frac{\nabla S}{|\nabla S|} - \frac{\nabla C}{|\nabla C|}\right|^2 \quad . \tag{4.1-21}$$

It is important that this parameter remains small throughout the inversion. As will be seen in section 4.4, this parameter is useful for monitoring the stability of the inversion as well as the suitability of the quadratic approximations and distance limits.

FORTRAN code for applying the above procedure was supplied by Steve Brown (The University of Sydney). This code was modified as described in the following three sections. The final code consists of a group of subroutines driven by MEMNL$8 (see the appendix).

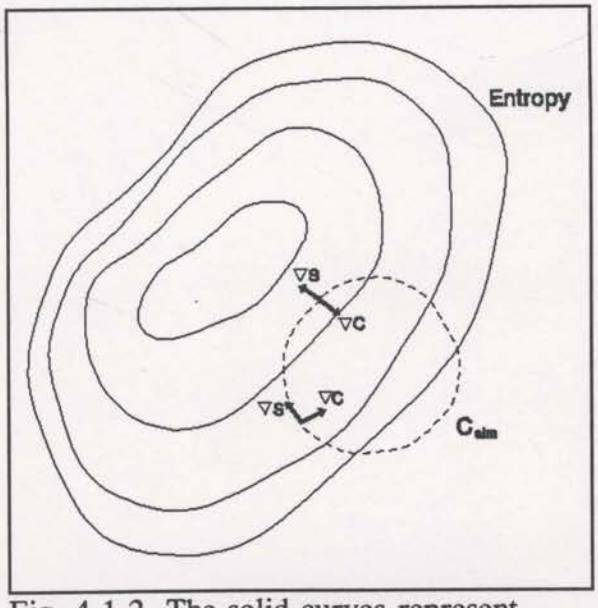


Fig. 4.1-2 The solid curves represent contours of S. The dashed curve is the contour of C with a value of C_{aim}. Maximum entropy points require parallel gradients.

entropy inversion algorithm is the TEST parameter (see Skilling and Bryan, 1984). Figure 4.1-2 shows that the gradients of the entropy surface and the constraint statistic surface must be parallel at the true maximum entropy point. The TEST parameter measures the degree of non-parallelism and is defined as,

$$\text{TEST} = \frac{1}{2}\left|\frac{\nabla S}{|\nabla S|} - \frac{\nabla C}{|\nabla C|}\right|^2 \quad . \tag{4.1-21}$$

It is important that this parameter remains small throughout the inversion. As will be seen in section 4.4, this parameter is useful for monitoring the stability of the inversion as well as the suitability of the quadratic approximations and distance limits.

FORTRAN code for applying the above procedure was supplied by Steve Brown (The University of Sydney). This code was modified as described in the following three sections. The final code consists of a group of subroutines driven by MEMNL$8 (see the appendix).

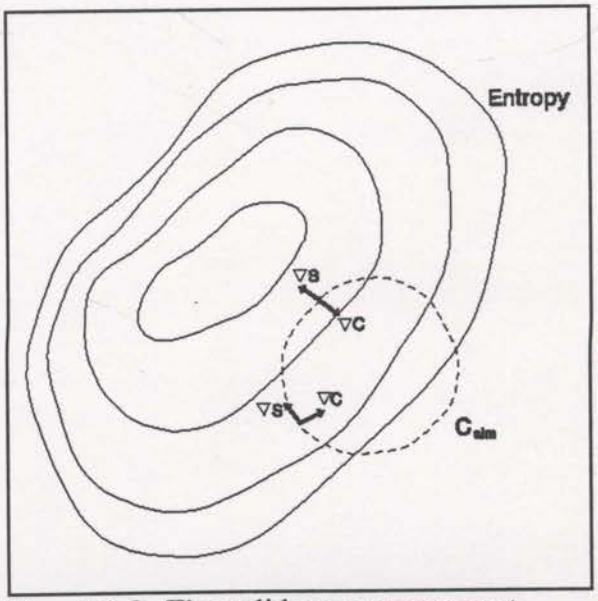


Fig. 4.1-2 The solid curves represent contours of S. The dashed curve is the contour of C with a value of C_{aim}. Maximum entropy points require parallel

NB. In this figure, the gradient of S is pointing in the direction of steepest ascent whilst the gradient of C is pointing in the direction of steepest descent.

4.2 Simultaneous diagonalisation within the subspace.

As mentioned in the previous section, Skilling and Bryan (1984) defined an entropy metric to assist with the large dynamic range problems that accompany inversions of astronomical data. Using this metric affects the theory of the simultaneous diagonalisation of matrices **A** and **B** (see section 4.1). It is not essential to develop this theory in terms of the entropy metric. The purpose of this section is to define precisely how this diagonalisation is to be performed.

Recall the subspace representations of the quadratic approximations (equations (4.1-12) and (4.1-13)),

$$\bar{S}(\mathbf{y}) = S_0 + \mathbf{y}^{\mathrm{T}}\mathbf{a} - \frac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{y} \qquad (4.2\text{-}1)$$

$$\bar{C}(\mathbf{y}) = C_0 + \mathbf{y}^{\mathrm{T}}\mathbf{b} + \frac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{B}\mathbf{y} \quad , \qquad (4.2\text{-}2)$$

where,

$$\mathbf{a} = \mathbf{E}^{\mathrm{T}}\nabla S_0 \quad , \quad \mathbf{A} = \mathbf{E}^{\mathrm{T}}(-\nabla\nabla S_0)\mathbf{E} \quad ,$$

$$\mathbf{b} = \mathbf{E}^{\mathrm{T}}\nabla C_0 \quad , \quad \mathbf{B} = \mathbf{E}^{\mathrm{T}}(\nabla\nabla C_0)\mathbf{E} \quad .$$

Here **a** and **b** are (r x 1)-vectors and **A**, **B** are (r x r)-matrices. Now, $-\nabla\nabla S_0$ is positive definite[1] which makes **A** real, symmetric and positive definite, and **B** real and symmetric.

---

[1] The second derivative of S (equation 3.2-2) is $\frac{\partial^2 S}{\partial q_i \partial q_j} = \frac{-\delta_{ij}}{q_j}$. The entries of $-\nabla\nabla S_0$ are all positive.

The aim is to find a transformation H which simultaneously diagonalises A and B. Define the transformation as,

$$\mathbf{y} = \mathbf{H}\mathbf{x} \tag{4.2-3}$$

so that equations (4.2-1) and (4.2-2) become,

$$\bar{S}(\mathbf{x}) = S_0 + \mathbf{x}^T(\mathbf{H}^T\mathbf{a}) - \frac{1}{2}\mathbf{x}^T(\mathbf{H}^T\mathbf{A}\mathbf{H})\mathbf{x} \tag{4.2-4}$$

$$\bar{C}(\mathbf{x}) = C_0 + \mathbf{x}^T(\mathbf{H}^T\mathbf{b}) + \frac{1}{2}\mathbf{x}^T(\mathbf{H}^T\mathbf{B}\mathbf{H})\mathbf{x} \quad , \tag{4.2-5}$$

It is now possible to invoke a well known theorem which states: if A is real, symmetric and positive definite and B is real and symmetric then there exists a transformation, H, such that $\mathbf{H}^T\mathbf{A}\mathbf{H} = \mathbf{I}$,[2] and $\mathbf{H}^T\mathbf{B}\mathbf{H}$ is diagonal.

In particular H can be constructed from the eigenvectors of A and B. If $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_r]$ is the (r x r)-matrix of normalised eigenvectors of A, then $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and $\mathbf{U}^T\mathbf{A}\mathbf{U} = \mathbf{D}_\alpha$ where $\mathbf{D}_\alpha$ is the diagonal matrix consisting of the eigenvalues of A, $(\alpha_1, \alpha_2, \ldots, \alpha_r)$. All the eigenvalues are positive since A is positive definite. Now define,

$$\mathbf{G} = \mathbf{U}\mathbf{D}_{\frac{1}{\sqrt{\alpha}}} = \left[\frac{\mathbf{u}_1}{\sqrt{\alpha_1}}, \frac{\mathbf{u}_2}{\sqrt{\alpha_2}}, \ldots, \frac{\mathbf{u}_r}{\sqrt{\alpha_r}}\right] \tag{4.2-6}$$

so that,

$$\mathbf{G}^T\mathbf{G} = \mathbf{D}_{\frac{1}{\sqrt{\alpha}}}\mathbf{U}^T\mathbf{U}\mathbf{D}_{\frac{1}{\sqrt{\alpha}}} = \mathbf{D}_{\frac{1}{\sqrt{\alpha}}}\mathbf{D}_{\frac{1}{\sqrt{\alpha}}} = \mathbf{D}_{\frac{1}{\alpha}} \tag{4.2-7}$$

and,

---

[2] I is the (r x r)-identity matrix.

$$G^TAG = D_{\frac{1}{\sqrt{\alpha}}}(U^TAU)D_{\frac{1}{\sqrt{\alpha}}} = D_{\frac{1}{\sqrt{\alpha}}}D_\alpha D_{\frac{1}{\sqrt{\alpha}}} = I \quad . \tag{4.2-8}$$

Also let $B' = G^TBG$ which will remain real and symmetric. This B' can be diagonalised in the same way as above, with the use of the matrix of normalised eigenvectors of B', namely V. That is,

$$V^TB'V = D_\beta \quad . \tag{4.2-9}$$

Notice that the eigenvalues of B' are $(\beta_1, \beta_2, \ldots, \beta_r)$ and $V^TV = I$. Construct,

$$H = GV = UD_{\frac{1}{\sqrt{\alpha}}}V \quad , \tag{4.2-10}$$

and consider,

$$H^TAH = V^T(G^TAG)V = V^TIV = V^TV = I$$

and,

$$H^TBH = V^T(G^TBG)V = V^TB'V = D_\beta \quad .$$

Therefore, the transformation H as defined by equation (4.2-10) is the required transformation for the simultaneous diagonalisation of A and B within the subspace. Application of this operator leads directly to equations (4.1-14) and (4.1-15).

[1] The second term on the right of this equation disappears when ∇C is evaluated at the current model, v_0, at each step of the iterative inversion.

4.3 Including stages of decreasing smoothing and weighted least squares.

The rationale and purpose of using stages of decreasing smoothing within the inversion process was described in section 2.5. Actually including this technique imposes a significant computational burden. Care has to be taken to ensure that the code is as efficient as possible.

Recall the derivatives of the constraint statistic from section 2.5,[1]

$$\nabla C = 2\mathbf{C}_m \nabla \mathbf{X}_{err}^T \mathbf{X}_{err} + 2(\mathbf{v}-\mathbf{v}_0) , \tag{4.3-1}$$

and,

$$\nabla\nabla C = 2\mathbf{C}_m \nabla \mathbf{X}_{err}^T \nabla \mathbf{X}_{err} \mathbf{C}_m + 2\mathbf{C}_m , \tag{4.3-2}$$

where $\mathbf{C}_m$ is the model covariance matrix which is constructed with Gaussian functions for the smoothing (see section 2.5). Without the stages of smoothing technique, these equations would be,

$$\nabla C = 2\nabla \mathbf{X}_{err}^T \mathbf{X}_{err}$$

and,

$$\nabla\nabla C = 2\nabla \mathbf{X}_{err}^T \nabla \mathbf{X}_{err} .$$

The inclusion of the model covariance matrix, $\mathbf{C}_m$, in these expressions is significant as $\mathbf{C}_m$ is a (n x n)-matrix where 'n' is the number of discrete parameters in the model. Whenever ∇C or $\nabla\nabla C$ is required, this large matrix must be accessed and multiplied into the calculation. For the case of $\nabla\nabla C$ this multiplication is two-fold.

Fortunately $\mathbf{C}_m$ is symmetric and many of the entries are not independent. The

4.3 Including stages of decreasing smoothing and weighted least squares.

The rationale and purpose of using stages of decreasing smoothing within the inversion process was described in section 2.5. Actually including this technique imposes a significant computational burden. Care has to be taken to ensure that the code is as efficient as possible.

Recall the derivatives of the constraint statistic from section 2.5,[1]

$$\nabla C = 2\mathbf{C}_m \nabla \mathbf{X}_{err}^T \mathbf{X}_{err} + 2(\mathbf{v} - \mathbf{v}_0) , \tag{4.3-1}$$

and,

$$\nabla\nabla C = 2\mathbf{C}_m \nabla \mathbf{X}_{err}^T \nabla \mathbf{X}_{err} \mathbf{C}_m + 2\mathbf{C}_m , \tag{4.3-2}$$

where $\mathbf{C}_m$ is the model covariance matrix which is constructed with Gaussian functions for the smoothing (see section 2.5). Without the stages of smoothing technique, these equations would be,

$$\nabla C = 2\nabla \mathbf{X}_{err}^T \mathbf{X}_{err}$$

and,

$$\nabla\nabla C = 2\nabla \mathbf{X}_{err}^T \nabla \mathbf{X}_{err} .$$

The inclusion of the model covariance matrix, $\mathbf{C}_m$, in these expressions is significant as $\mathbf{C}_m$ is a (n x n)-matrix where 'n' is the number of discrete parameters in the model. Whenever ∇C or $\nabla\nabla C$ is required, this large matrix must be accessed and multiplied

[1] The second term on the right of this equation disappears when ∇C is evaluated at the current model, $\mathbf{v}_0$, at each step of the iterative inversion.

numerical value of the entry $(\mathbf{C}_m)_{ij}$ depends on the physical distance separating the grid nodes represented by i and j (see equation (2.5-7)). For a discrete two-dimensional grid with a total of n nodes, there are n different physical separations possible. The (n x n)-matrix $\mathbf{C}_m$ can therefore be completely stored as a vector of n numbers.

Multiplying an (n x n)-matrix by a (n x n)-matrix requires n^3 multiplications, however, multiplying an (n x n)-matrix by an (n x 1)-vector requires only n^2 multiplications. As n becomes large, straight matrix calculations are prohibitive. Fortunately the inversion process described in sections 4.1 and 4.2 can be performed without the necessity of such matrix computations. This means that every time $\nabla\nabla C$ is required in a computation, it is recalculated, but since these calculations are all matrix-vector products, the multiplications required total much less than n^3.

The FORTRAN code that performs the maximum entropy inversion is a group of subroutines driven by a program MEMNL$8 (see the appendix). The original code for this inversion was supplied by Steve Brown. Major modifications have been made to the code to tailor it to seismic reflection tomography and to include the model covariance matrix.

4.4 Automatic inversion control.

As described in section 4.1, the inversion process is based on quadratic approximation of both C (the constraint statistic) and S (entropy) within a 4 or 6-dimensional subspace. In general, the C surface will be more non-linear than a quadratic function. It is important that movements within the model space are kept within the region of acceptable quadratic approximation. The distance limit (see section 4.1), L, is used precisely for this purpose. However, it is not a simple matter to determine the limits of acceptable quadratic approximation.

Another problem for the control of the inversion is suitable criteria for stopping a smoothing stage (see section 2.5) and beginning the next stage with less smoothing. A smoothing stage is to be terminated when no further significant C reductions are to be achieved. It is not simple to determine when this occurs. It must be remembered that C is non-linear and that it is possible for the rate of C decrease to vary significantly. The inversion stage may be prematurely terminated at a point where a couple of iteration produce little decrease, even though some significant decreases could be made in later iterations. Another possibility is that the distance limit can be so small that successive iterations result in little C reduction, even though sizeable, stable reductions may be achieved if the distance limit is relaxed. Without more complete knowledge of the non-linear C surface, these problems may be difficult to overcome.

Fortunately, some of the parameters available from the inversion process itself give some insight into the success and applicability of the current inversion iteration.

Parameters such as the recent history of C reductions, the actual length of the iteration update compared to the distance limit, and the changing values of the TEST parameter (see section 4.1) are all very useful for specifying the requirements for suitable control of the inversion process. Using these parameters, the software can be coded to automatically adapt to the changing conditions that confront the iterative inversion.

The most important parameter that needs to be adaptive for a successful inversion is the distance limit, L (see section 4.1). This limit must have the ability to either increase or decrease rapidly, when necessary. The structure of the C surface can change dramatically between iterations, and especially between stages with different smoothing. A successful distance limit in one stage may be inappropriate in the next stage. Generally, larger distances can be traversed safely when the smoothing is heavier.

The algorithm for the automatic, adaptive control of the inversion process was coded into the main driver routine, METOMOS6 (see the appendix). This algorithm is now outlined in point form.

a) The C values of the most recent 5 iterations are recorded in an array. Let these be known as $C_{mem(i)}$ where i ranges from 1 to 5, and $C_{mem(5)}$ being the most recent value.

b) If any of $C_{mem(2)}$ through $C_{mem(5)}$ is more than 2% lower than $C_{mem(1)}$, then set a flag to signify that sufficient C reduction has been achieved in recent iterations. This value of 2% is arbitrary but it seems to work well in practice. If less than 5 iterations have been performed, this flag is always set.

c) Now check for one of the two bad conditions.

(1) TEST > 0.2 and TEST > TESTOLD. Here 0.2 is the (somewhat arbitrary[1]) choice of the target TEST value and TESTOLD is the TEST value from the previous iteration. This condition implies that the current image is too far from the maximum entropy image and that it is moving further away.

(2) Two or more of the last 5 iterations have resulted in increases of C. This condition implies that the distance limit maybe too large and the quadratic approximation is only valid over a smaller range.

Given either of these conditions, the distance limit is halved, the counter UCOUNT is incremented, all $C_{mem(i)}$ are set to $C_{mem(5)}$ and the execution is sent back for the next iteration. The UCOUNT parameter records the number of times the inversion has been detected to be unstable. Setting all $C_{mem(i)}$ to the value of $C_{mem(5)}$ gives the next iteration, with a smaller distance limit, a clean start. Previous problems are therefore not allowed to plague the following iterations.

If either condition (1) or (2) is satisfied and the distance limit is already very small, and TEST is less than 0.2, then the stage is ended as further distance limit reduction will be pointless.

d) Next, if the 2% flag from b) has not been set and TEST is less than 0.2, then assume that as much C reduction as possible has been achieved and terminate the current stage. Termination will proceed unless the current update distance is more than 90% of the distance limit and UCOUNT (the instability count) is less than 10. In this case, the inversion may well be in a stable situation but simply stifled by a small distance limit. The action here is to increase the distance limit by 3/2 and go onto the

---

[1] Skilling and Bryan, 1984, advocated TEST < 0.1 in their applications.

next iteration.

The UCOUNT parameter is used to stop generally unstable inversions from getting any more distance limit increases. If the inversion has appeared temporarily unstable more than 9 times already (ie. UCOUNT $>$ 10) then it is probably not worth giving it any more chances with a larger distance limit.

e) The last check is for a stable inversion that may benefit from a slightly larger distance limit, hopefully improving the convergence rate. Here, if there were no C increases in the last 5 iterations, and the current update distance is more than 90% of the distance limit, and UCOUNT is less than 10, then increase the distance limit by 5/4. The increases suggested here and in d) have been chosen with the help of practical experimentation.

f) If no action has been enforced by any of the above conditions, simply begin the next iteration without any changes.

The objective of this automatic adaptive inversion control is two-fold: (i) to try and avoid early terminations when considerable reductions were still possible, and (ii) to avoid late terminations where large computational expense is wasted for very little gain. It is difficult to get the balance right but practical results show that the above algorithm works well in most cases.

In simple terms, the procedure described above treats C increases and TEST increases (above 0.2) as indicators of possible instability. These are indications that the extent of valid quadratic approximation may be less than the distance limit. If these conditions persist, the distance limit is reduced. On the other hand, the distance limit can be increased if everything appears stable and the current iteration update is being stopped at the distance limit. In this case, the quadratic approximation may be valid

beyond the distance limit and faster convergence may be achieved with a larger distance limit. The range of validity of the quadratic approximation to C will vary as the local topology of C varies. The automatically controlled inversion, as described above, is designed to adapt the distance limit to the changing conditions encountered as the iterations proceed.

5.1 Tests with square-anomaly model

It is important to have a simple, yet significant, problem for the evaluation of the algorithm described in the earlier sections. Simplicity is needed to ensure that the algorithms success can be clearly measured. For this purpose, the model shown in Figure 5.1-1 has been developed (this model was also used in section 2.4). This model consists of a 2 km by 2 km subsurface area with a background velocity consisting of a simple linear increase with depth of 0.75 m/s per second. The velocity at the surface is 1500 m/s. Within this background velocity field, a 400 x 400 metre square anomaly with a constant velocity of 2250 m/s is centred. This anomaly velocity has a velocity increase for the top half and a decrease for the bottom half.

The velocity of the background and the anomaly are identical at a depth of 1000 metres. Figure 5.1-2 shows the model with the velocities displayed as contours.

Traveltime data were computed for this model by tracing rays from regular source and receiver locations and a range of surface angles. A schematical demonstration of the types of rays that

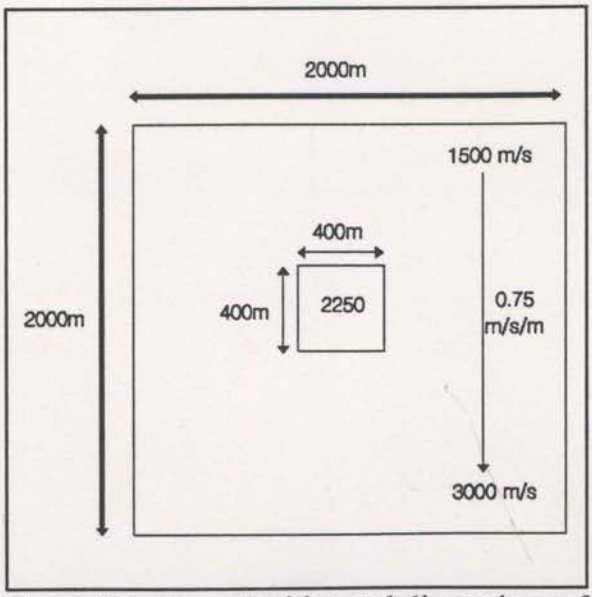


Fig. 5.1-1 The velocities and dimensions of the model to be studied in section 5.1.

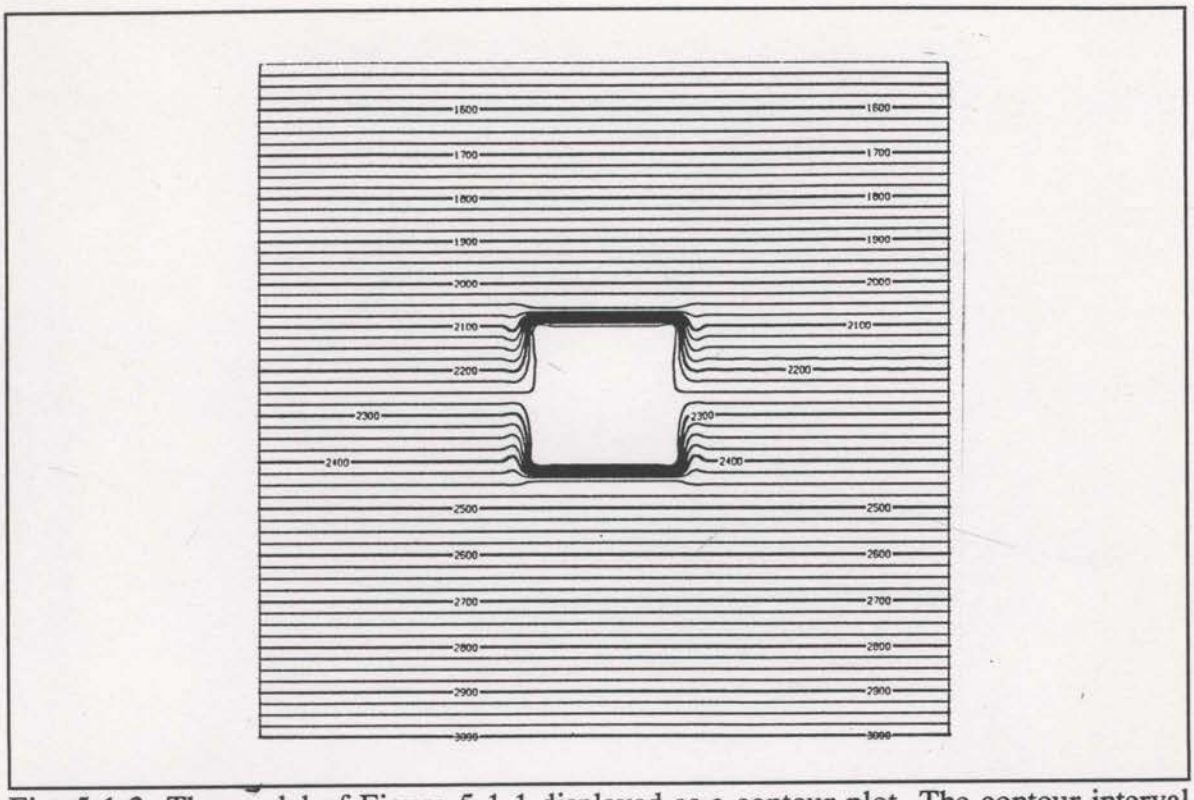


Fig. 5.1-2 The model of Figure 5.1-1 displayed as a contour plot. The contour interval is 25 m/s.

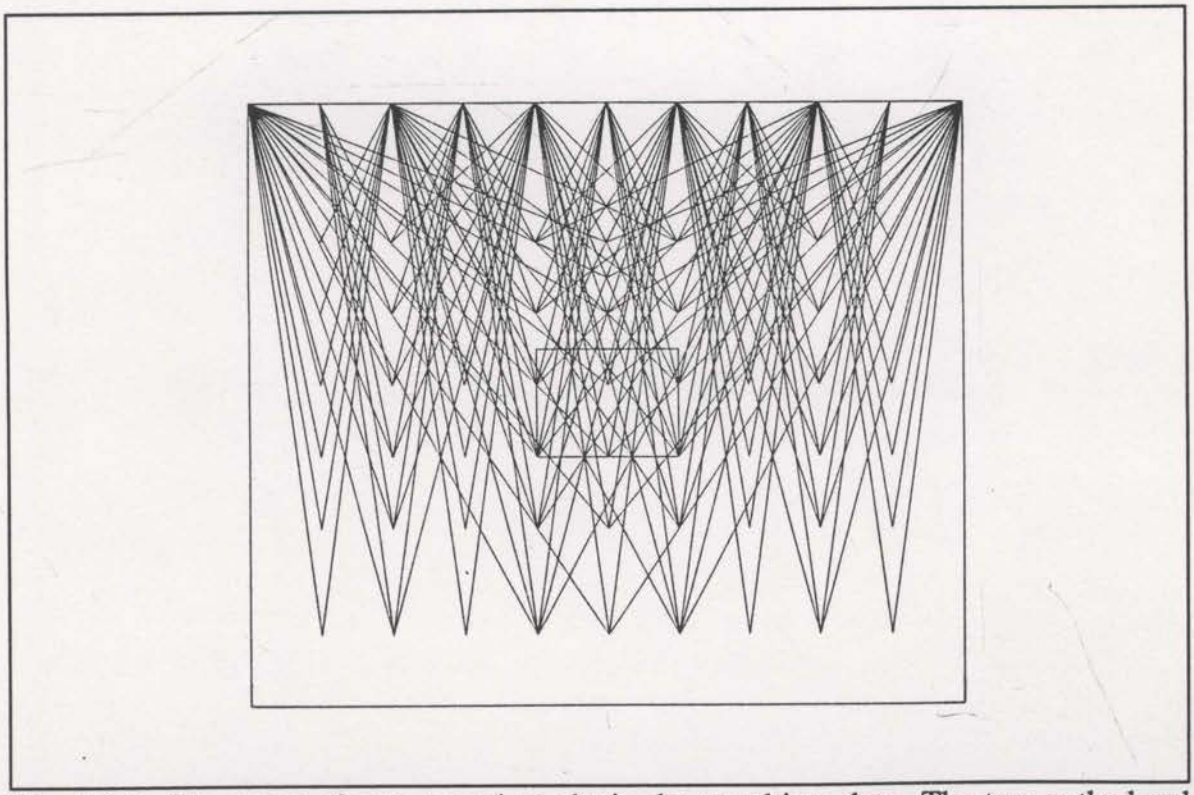


Fig. 5.1-3 The types of rays traced to obtain the traveltime data. The true paths bend where the velocities vary.

[1] The RMS X_{err} value for the reconstruction in Figure 5.1-4 is 3.46 metres whereas for the reconstruction of Figure 5.1-5 it is 9.95 metres. There is an apparent paradox here, the better solution has a larger error. This comes about since the 184 traveltime data set was unable to identify the model but was still able to vary the velocities to match the traveltimes quite well. Consider a data set consisting of a single ray. Clearly the velocities can be varied such the residual error will be zero, yet the solution will not be satisfactory.

were traced is shown in Figure 5.1-3 (Figure 2.4-4 shows some of the actual raypaths). The objective of these ray locations was to provide a uniform range of source/receiver offsets and penetration depths.

Initially only 184 rays were traced for the traveltime data (the ones actually displayed in Figure 2.4-4). An inversion of these data can be seen in Figure 5.1-4. The starting model for the inversion consists of the linear background field only. This inversion used finite-difference derivative estimates and no constraint statistic smoothing. The value of the constraint statistic was reduced from about 140,000 to just 2197, but the anomaly has not been resolved. Traveltime data was then obtained for 1312 raypaths (with surface locations spaced uniformly across the extent of the model) and an identical inversion performed. The final image of Figure 5.1-5 shows that the extra raypaths, implying more complete angular coverage, have resulted in a reasonable image of the anomaly. The 184 traveltime data set did not contain enough information for the inversion to recognise the anomaly.[1]

Section 2.4 discussed the theory behind the semi-analytic derivative estimates. This was developed because finite-difference derivative estimates were becoming very time consuming when 1,312 rays were used. Inversion results with semi-analytic derivative estimates were shown in section 2.4 and are reproduced in Figure 5.1-6. Surprisingly, these results were an improvement over the equivalent results with finite-difference derivative estimates. Possible reasons for this improvement were discussed in section 2.4. Further improvements of the final image were obtained when stages of decreasing smoothing were included in the inversion (see Figure 5.1-7). The stages applied here had Gaussian half widths (see section 2.5) of 125, 67, 33 and 0 metres. This inversion has done a good job of imaging the anomaly, however, a constant velocity of 2250 m/s within

were traced is shown in Figure 5.1-3 (Figure 2.4-4 shows some of the actual raypaths). The objective of these ray locations was to provide a uniform range of source/receiver offsets and penetration depths.

Initially only 184 rays were traced for the traveltime data (the ones actually displayed in Figure 2.4-4). An inversion of these data can be seen in Figure 5.1-4. The starting model for the inversion consists of the linear background field only. This inversion used finite-difference derivative estimates and no constraint statistic smoothing. The value of the constraint statistic was reduced from about 140,000 to just 2197, but the anomaly has not been resolved. Traveltime data was then obtained for 1312 raypaths (with surface locations spaced uniformly across the extent of the model) and an identical inversion performed. The final image of Figure 5.1-5 shows that the extra raypaths, implying more complete angular coverage, have resulted in a reasonable image of the anomaly. The 184 traveltime data set did not contain enough information for the inversion to recognise the anomaly.[1]

Section 2.4 discussed the theory behind the semi-analytic derivative estimates. This was developed because finite-difference derivative estimates were becoming very time consuming when 1,312 rays were used. Inversion results with semi-analytic derivative estimates were shown in section 2.4 and are reproduced in Figure 5.1-6. Surprisingly, these results were an improvement over the equivalent results with finite-difference derivative estimates. Possible reasons for this improvement were discussed in section 2.4.

---

[1] The RMS X_{err} value for the reconstruction in Figure 5.1-4 is 3.46 metres whereas for the reconstruction of Figure 5.1-5 it is 9.95 metres. There is an apparent paradox here, the better solution has a larger error. This comes about since the 184 traveltime data set was unable to identify the model but was still able to vary the velocities to match the traveltimes quite well. Consider a data set consisting of a single ray. Clearly the velocities can be varied such the residual error will be zero, yet the solution will not be satisfactory.

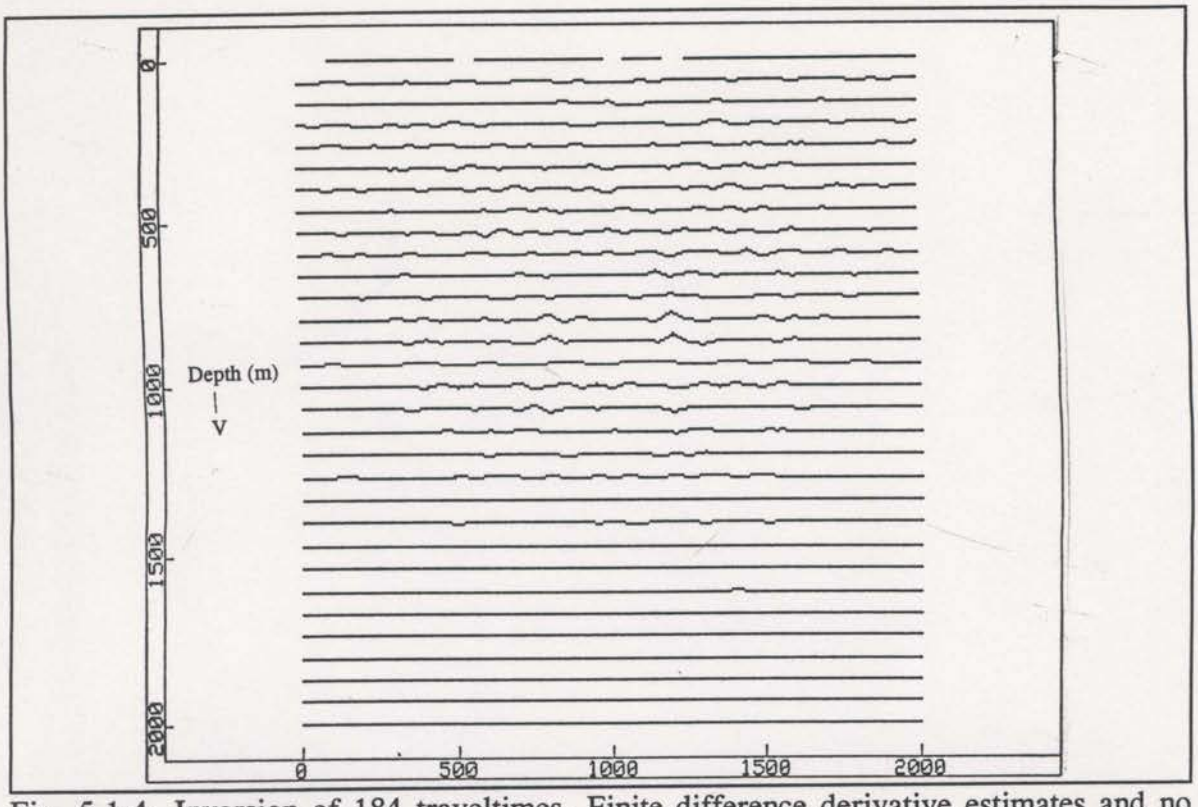


Fig. 5.1-4 Inversion of 184 traveltimes. Finite difference derivative estimates and no smoothing. Final C value of 2197.

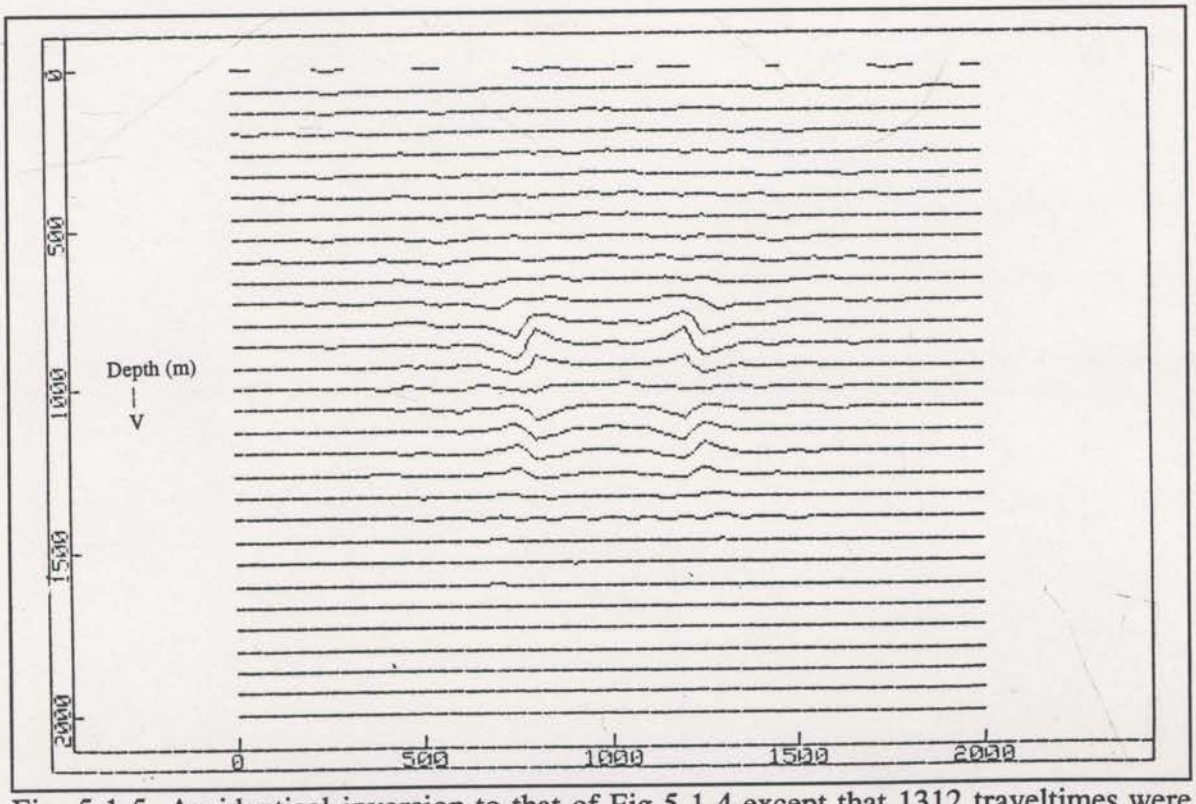


Fig. 5.1-5 An identical inversion to that of Fig 5.1-4 except that 1312 traveltimes were used. Final C value is approximately 130,000.

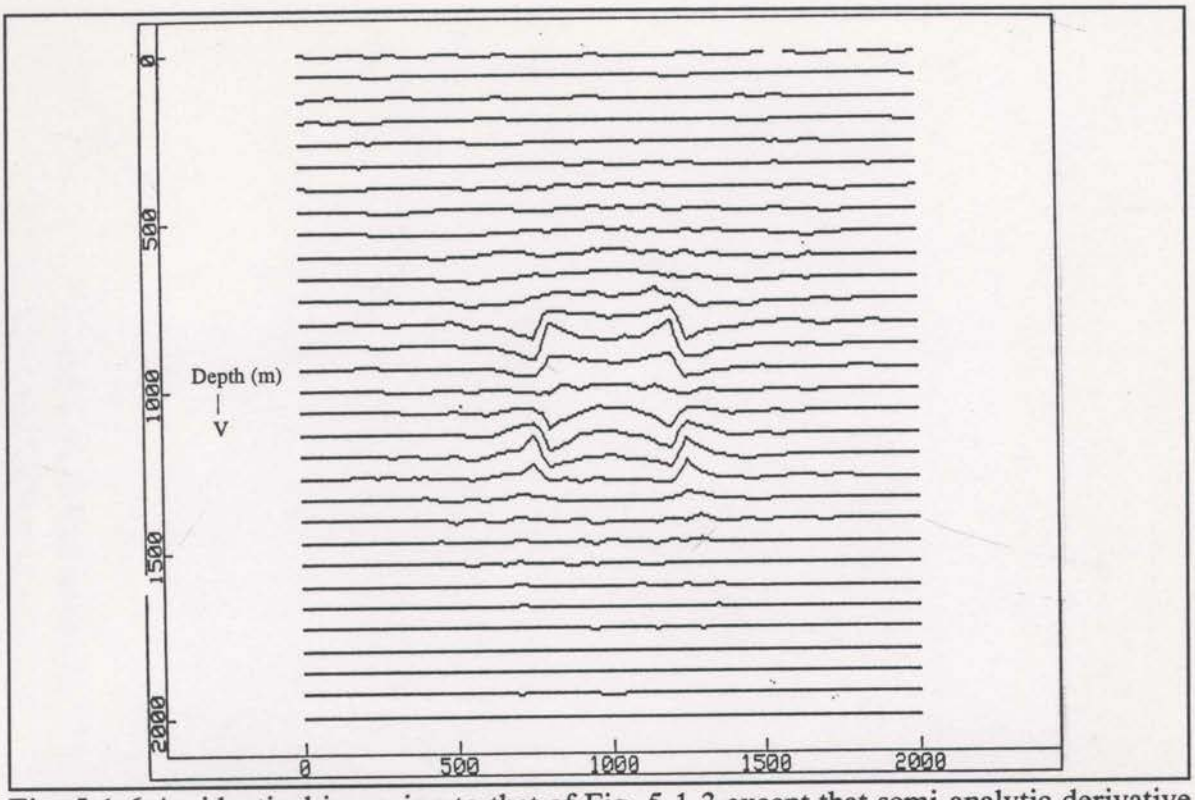


Fig. 5.1-6 An identical inversion to that of Fig. 5.1-3 except that semi-analytic derivative estimates were used. The final C value is approximately 50,000.

the anomaly has not resulted. The synthetic traveltime data is noise free, so the less than perfect image reconstruction signifies imperfections in the specifications of the problem. Inherent ambiguities exist that are not associated with noise.

All of the above inversions were initialised with the correct background velocity field as the starting model. In general, this background velocity field would not be known accurately. The image of Figure 5.1-8 is the result of an inversion that used an incorrect background field as the starting velocity model. The linear velocity increase of the background was set to 0.825 m/s per metre instead of 0.75 m/s per metre. This inversion did not use stages of decreasing smoothing. At first glance the anomaly appears to be imaged just as well as when the correct background starting model was used (Figure 5.1-6). However, closer inspection reveals that the anomaly has been imaged with velocities that are too fast and at a depth that is too deep. The inversion has been trapped by a local

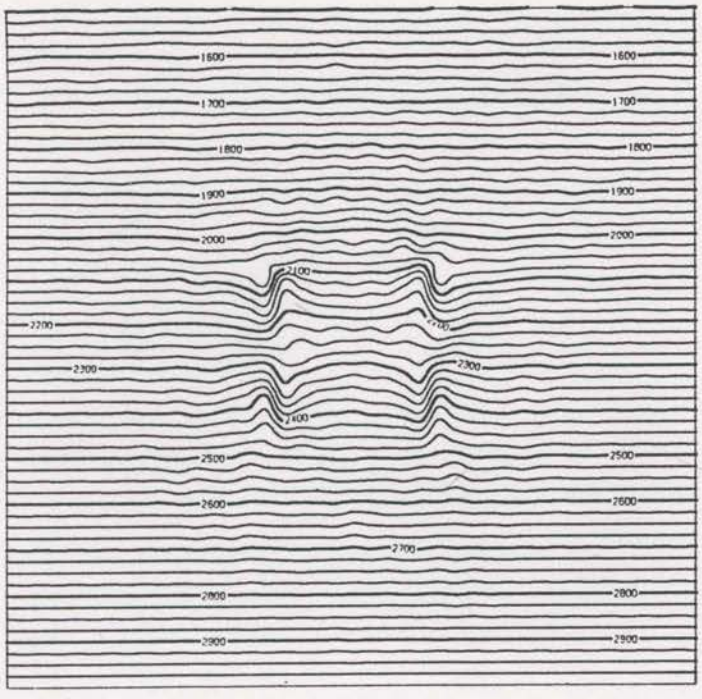


Fig. 5.1-7 Improved inversion results when stages of decreasing smoothing are added to the inversion used if Fig. 5.1-6. The final C value is approximately 43,000.

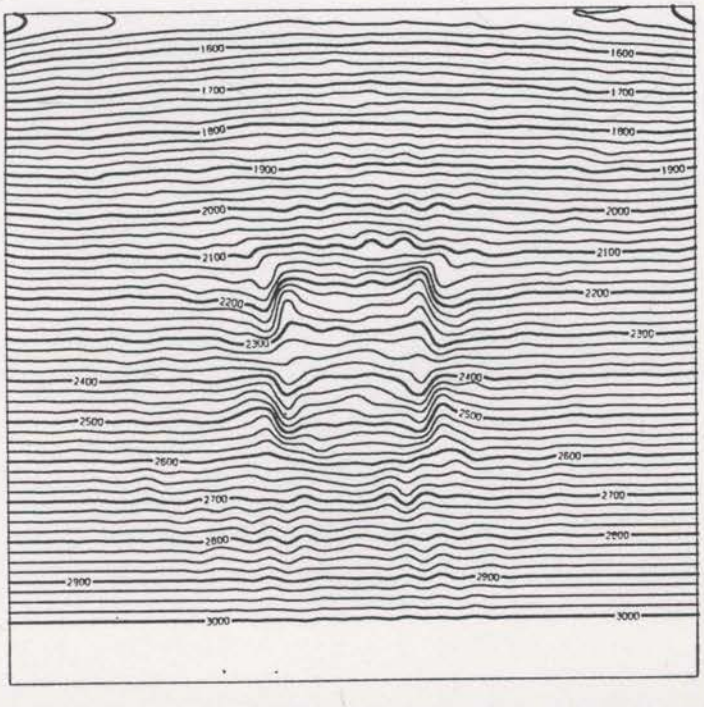


Fig. 5.1-8 The final image of an identical inversion to that of Fig. 5.1-6, ecept that the background linear gradient was 0.825 m/s per metre. The final C value is approximately 59,000.

minimum.

Incorporation of stages of decreasing smoothing into the inversion (see section 2.5) should help to avoid local minima. Figures 5.1-9 through 5.1-12 record the progress of an inversion with stages of decreasing smoothing. Figure 5.1-10 shows that the velocity field has been slowed by the first stage where the smoothers had half-widths of 2000 metres; this is the type of adjustment needed to correct the erroneous starting velocity model. The second stage, with smoothing half-widths of 1000 metres, continues the correction (Figure 5.1-11). These first two stages have approximately corrected the errors in the starting velocity model. The final stage of the inversion, without smoothing, has imaged the anomaly with correct velocities and depths (Figure 5.1-12). The local minimum that trapped the inversion of Figure 5.1-8 has been avoided.

Even though the synthetic data used in the above inversion are noise free, and computed using the same ray tracing as used in the inversion, the best image is still not exact and the final C value is significantly greater than zero. Either the inversion is still being trapped by a local minimum or it has encountered a broad plateau that has slowed convergence enough to bring the algorithm to a stop (see section 4.4). The usefulness of reflection tomography is seriously questioned by this observation.

Consider the circumstances of Figure 5.1-13. A ray is traced across the boundary between two layers. With the currently assigned velocities, a significant X_{err} value results. The actual change required is a simple increase of v_2 (see Figure 5.1-13). However, it is clear that any increase in the velocity contrast at the boundary will bend the two halves of the ray towards each other and reduce X_{err}. The velocities near the ray's intersection with the boundary become very sensitive. Decreases of v_1 just above the boundary are

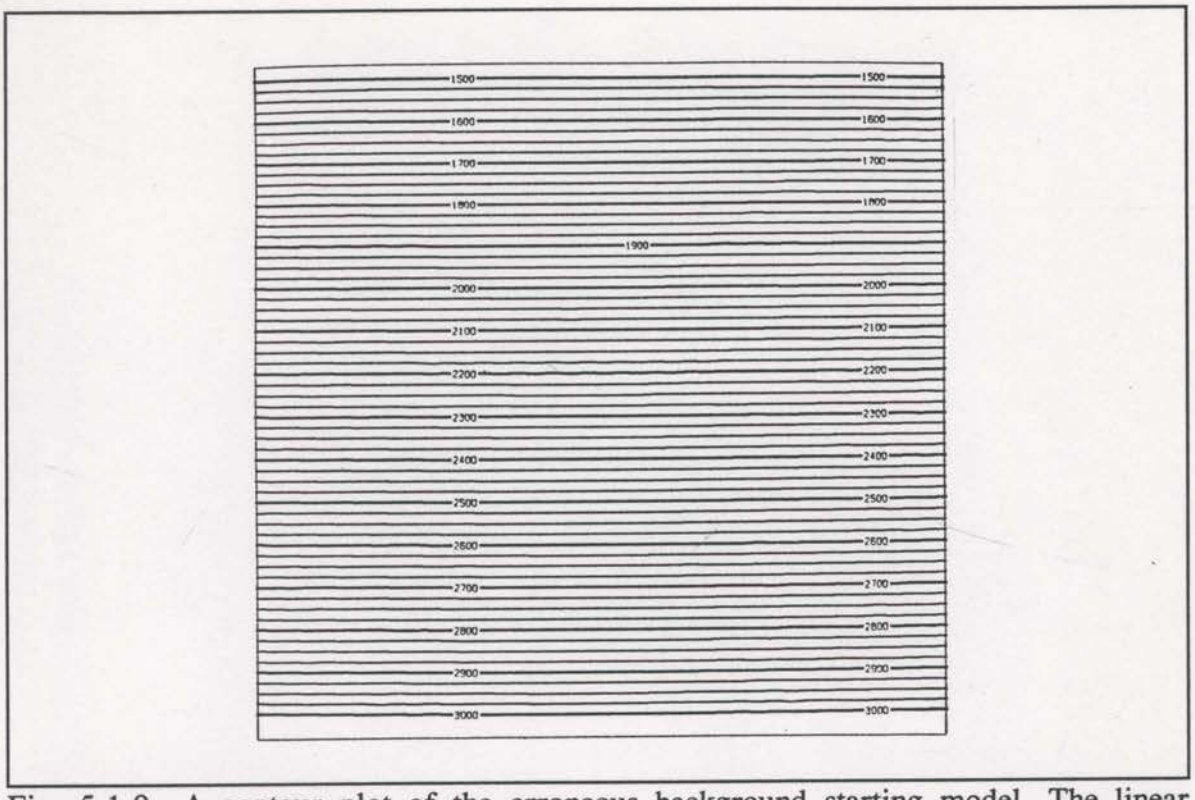


Fig. 5.1-9 A contour plot of the erroneous background starting model. The linear increase is 0.825 m/s per metre.

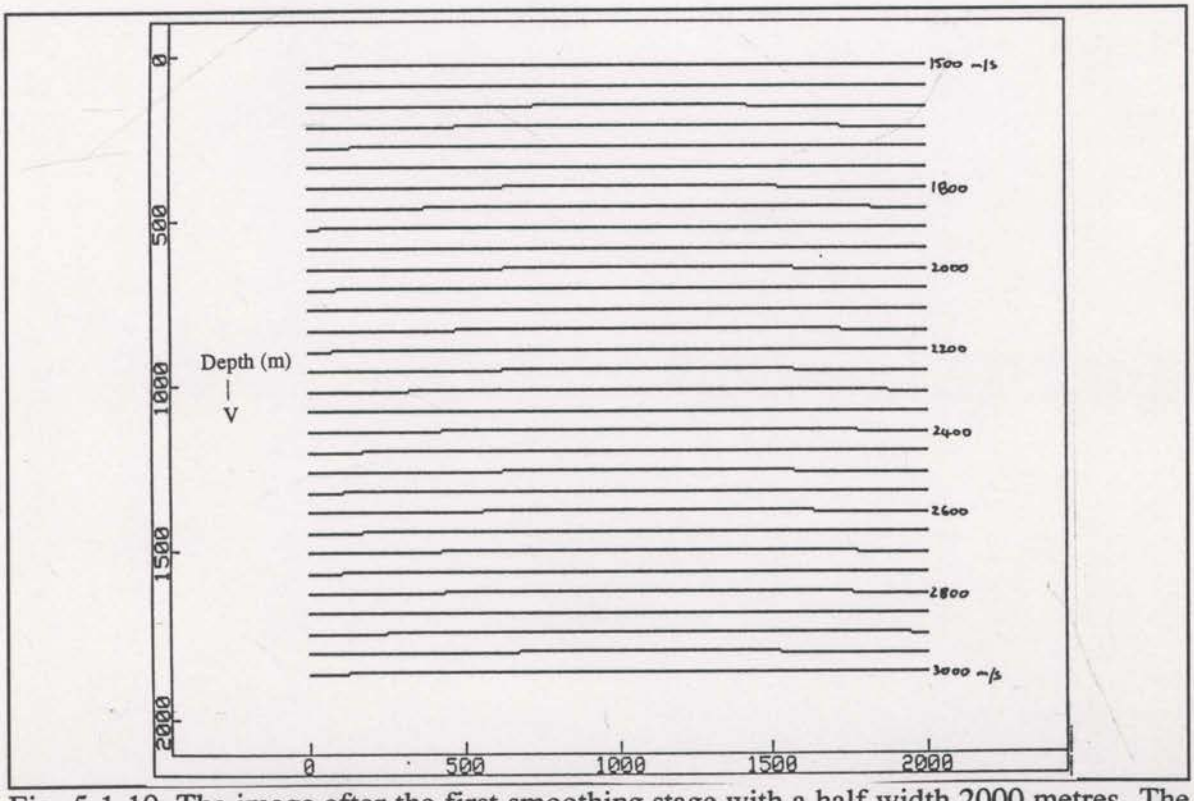


Fig. 5.1-10 The image after the first smoothing stage with a half width 2000 metres. The final C calue is approximately 1,700,000.

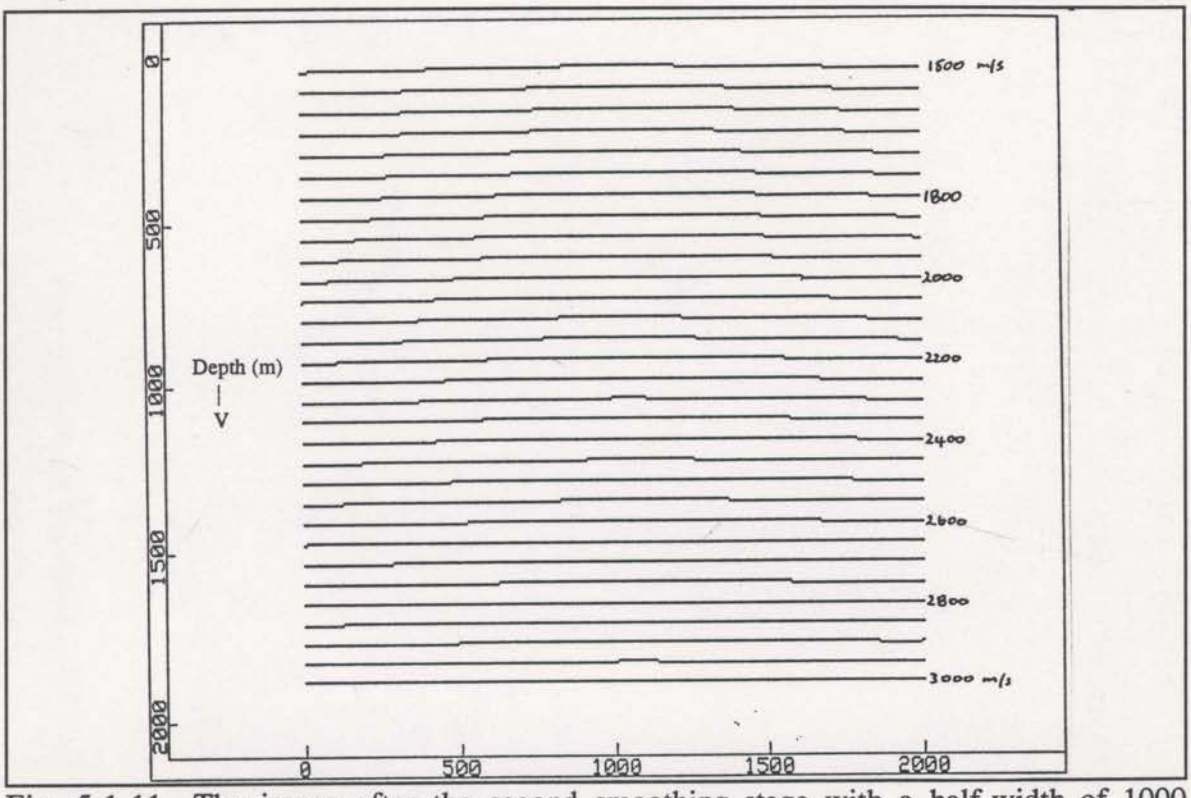


Fig. 5.1-11 The image after the second smoothing stage with a half-width of 1000 metres. The final C value is approximately 1,000,000.

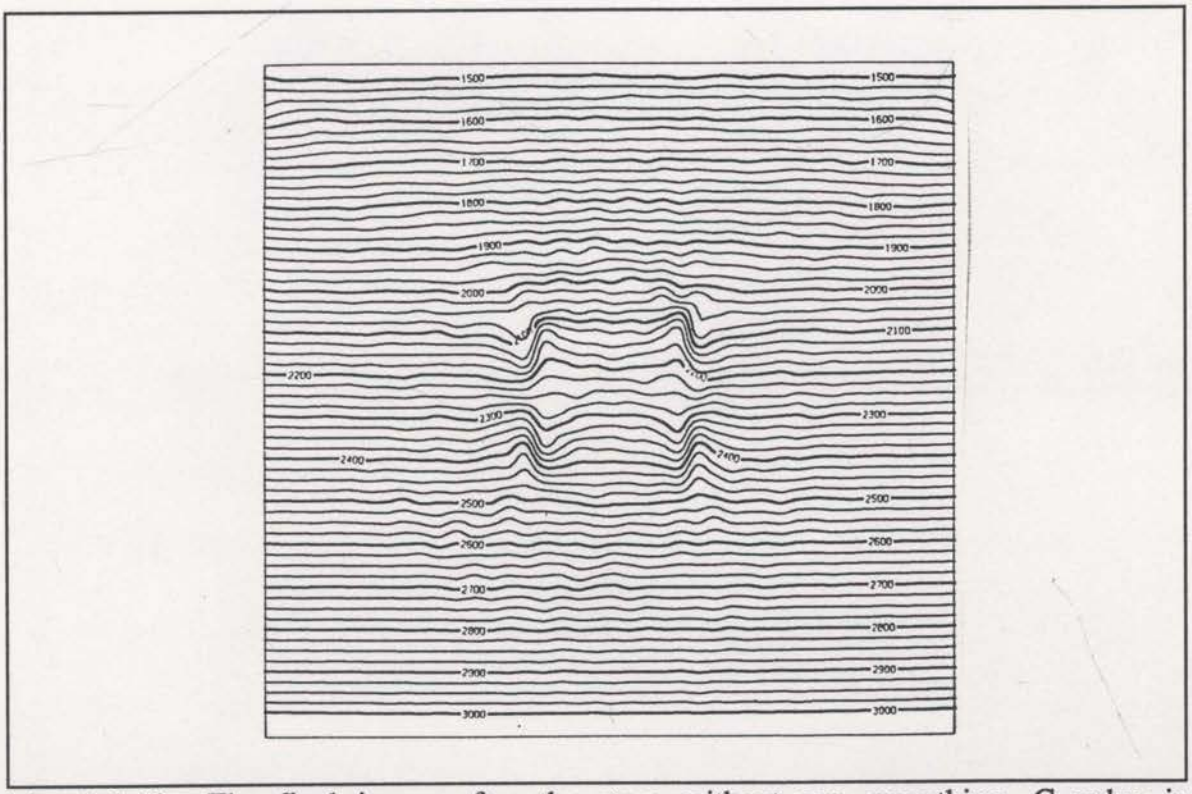


Fig. 5.1-12 The final image after the stage without any smoothing. C value is approximately 49,000.

just as effective at reducing X_{err} as are increases in v_2 just below the boundary. If enough rays of sufficiently varying geometry are not included in the inversion, there is a real possibility for low velocity zones to appear above the boundary. Such erroneous updates may allow the inversion to drift toward local minima.

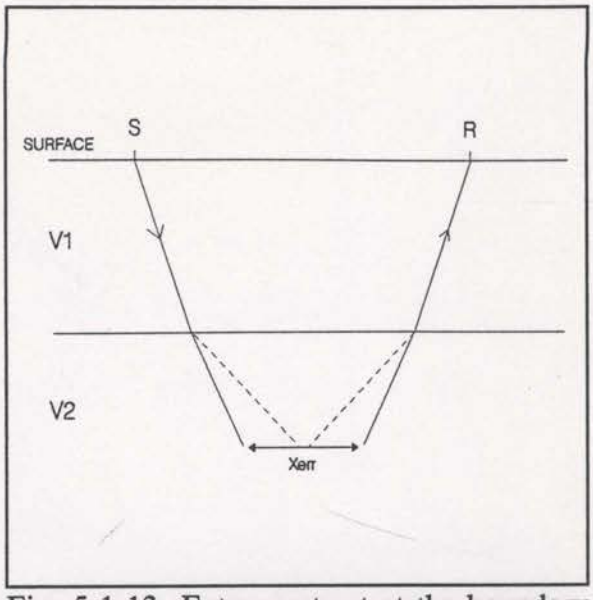


Fig. 5.1-13 Extra contrast at the boundary of the layers can remedy the condition of non-zero Xerr.

The sharp boundaries of the square anomaly in the synthetic model described in this section present similar problems as the layer boundary in Figure 5.1-13. Low velocity zones are apparent near the upper corners of the anomaly in Figure 5.1-12. Fast velocity zones are also apparent near the lower corners. Without introducing more traveltimes from new ray geometries, one way to help the inversion find the global minimum is to improve the starting model. With less modification necessary, these erroneous velocity zones will be less likely to appear. A starting model was constructed with the velocities within the anomalous region equal to the average of the background and true anomaly velocities at each depth. Effectively half of the anomaly has been placed in the starting model. This starting velocity model was also used as the prior information model (see section 3) in the inversion with the final image shown in Figure 5.1-14. It is not surprising that this image is the best result so far. There is much less evidence of erroneous velocity zones around the anomaly. The final constraint statistic, C, value is now approximately 20,000 instead of 49,000 as in the image of Figure 5.1-12. This

confirms that the previous inversion was trapped by a local minimum quite near the global minimum. A better starting model has avoided this local minimum. A greater multiplicity and variety of ray paths may also avoid problems with such local minima.

The above example of superior results with improved starting models suggests possible benefits from combining geological interpretation with reflection tomography. After an initial tomographic inversion the image can be geologically interpreted and a new simple starting model can be constructed. Such a model has the potential to improve the final results. This is similar to the expected effect of the stages of decreasing smoothing. However, the smoothing approach will inherently struggle with sharp discontinuities that are not present in the starting model. A combined and complementary use of geological interpretation and smoothing may be required for optimum results.

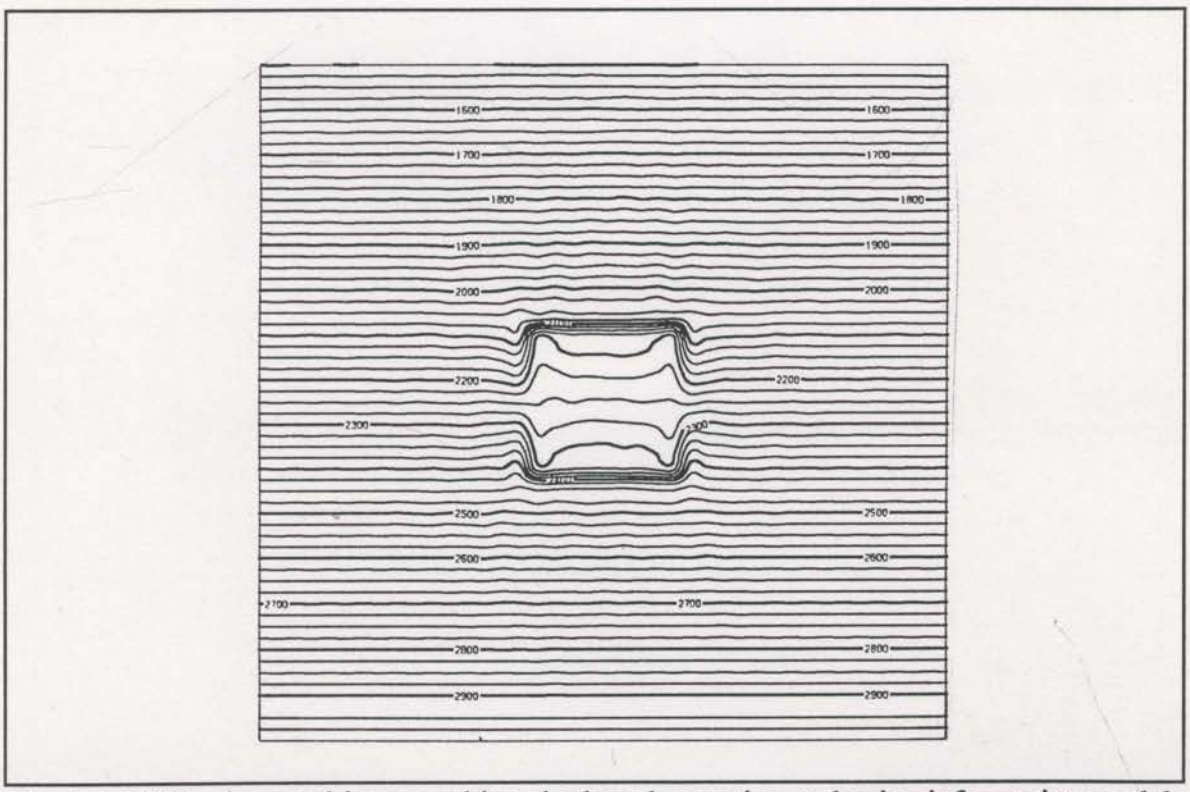


Fig. 5.1-14 The inverted image achieved when the starting and prior information models consisted of 50 percent of the anomalous variation. The final C value here is approximately 20,000.

5.2 Tests with fault model

The model created for the tests described in this section is a realistic layered model containing a single, high angle fault. This model was instructive in the optimum way to specify the starting velocity model. Useful and accurate results were only obtained once a suitable starting model was established.

The interval velocities and geometry of this fault model are shown in Figure 5.2-1. The model consists of three simple layers containing a normal fault with 100 metres of throw. The layer velocities are of realistic magnitude. Eight hundred rays with regular geometries (similar to those in section 5.1) were traced through this model. These rays produced the traveltime data used for the experimental inversions.

The starting model for the initial inversion of this synthetic data consisted of three layers without any hint of the fault. The layer velocities were the true velocities and the depths of the layer boundaries were the correct ones for the left side of the model. The final inversion image resulting from this starting model is disappointing (see Figure 5.2-2). The fault locations have barely been imaged and the depth of the layer boundaries on the right were unchanged. It appears that sharp discontinuities will tend not to be changed by a somewhat underdetermined image

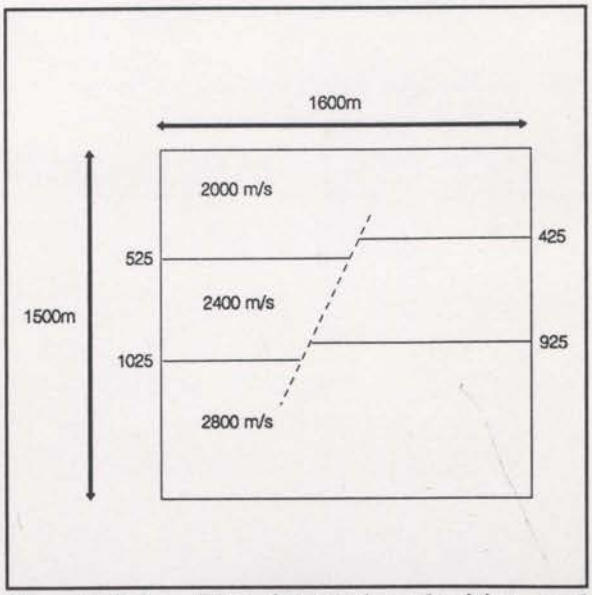


Fig. 5.2-1 The interval velocities and geometry of the fault model.

reconstruction. The main reason for this is that the required changes are very large. The sharp layer boundary of the starting model has been placed at the wrong depth for the right side of the model. Some velocities have to be changed from 2000 metres per second to 2400 metres per second, an increase of 20%. Sharp discontinuities suggest substantial prior knowledge about the velocity structure. One must be careful to ensure that unwarranted information is not expressed in the prior information model.

Consider an inversion where the starting velocity model is a simple linear function with depth. The prior information model (generally the starting model) does not express any knowledge of the layer boundaries in this case. The image of Figure 5.2-3 is the result of an inversion with such a starting (and prior information) model. The existence of a fault is clearly revealed - at least for the shallowest layer boundary. However, the linear starting model has not allowed precise imaging of the layer boundaries away from the fault[1].

In reality, the existence of layered geology would be known from nearby well logs or even CDP stacking velocities. The precise depth of the layer boundaries will not be known and the fault may not be clear, but the layers would be observed. The prior information model should exhibit the layered geology but be rather non-committal with respect to the depths of the boundaries. Such an approach leads to an initial model with correct layer velocities but with a 100 metre zone of linear change from one velocity to the next. The results of an inversion with such a starting model can be seen in Figure 5.2-4. The fault break has been well imaged, especially for the shallower layer boundary. The layer boundary to the right of the fault has moved upwards - to the left of the fault

[1] These will be harder to image in any case due to the surface restrictions of the raypaths.

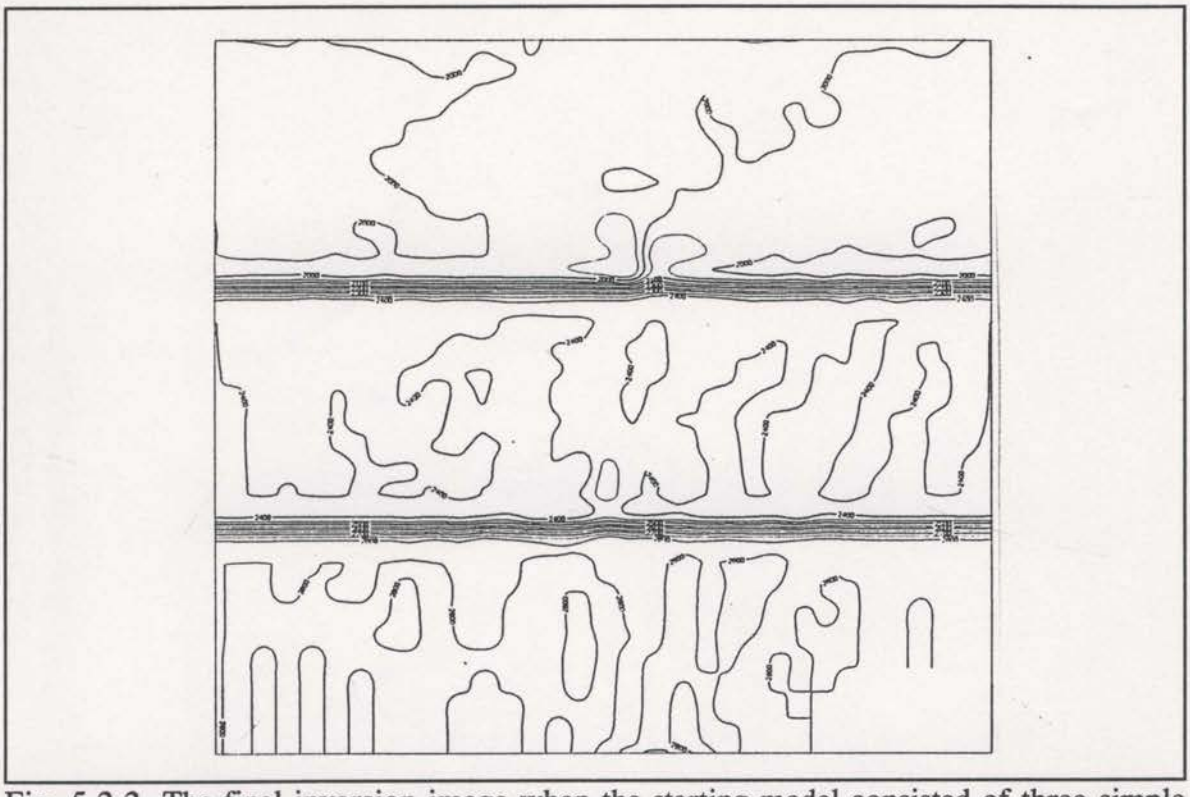


Fig. 5.2-2 The final inversion image when the starting model consisted of three simple layers with sharp boundaries.

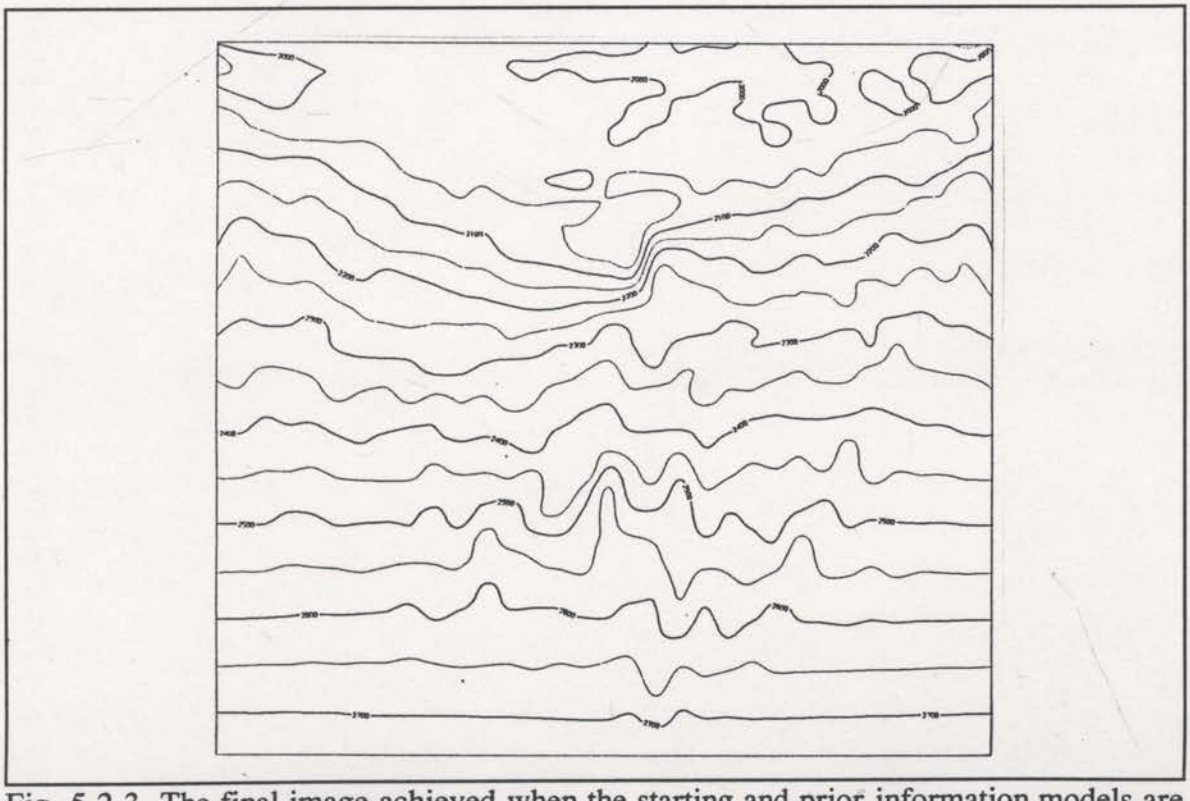


Fig. 5.2-3 The final image achieved when the starting and prior information models are simple linear increases with depth.

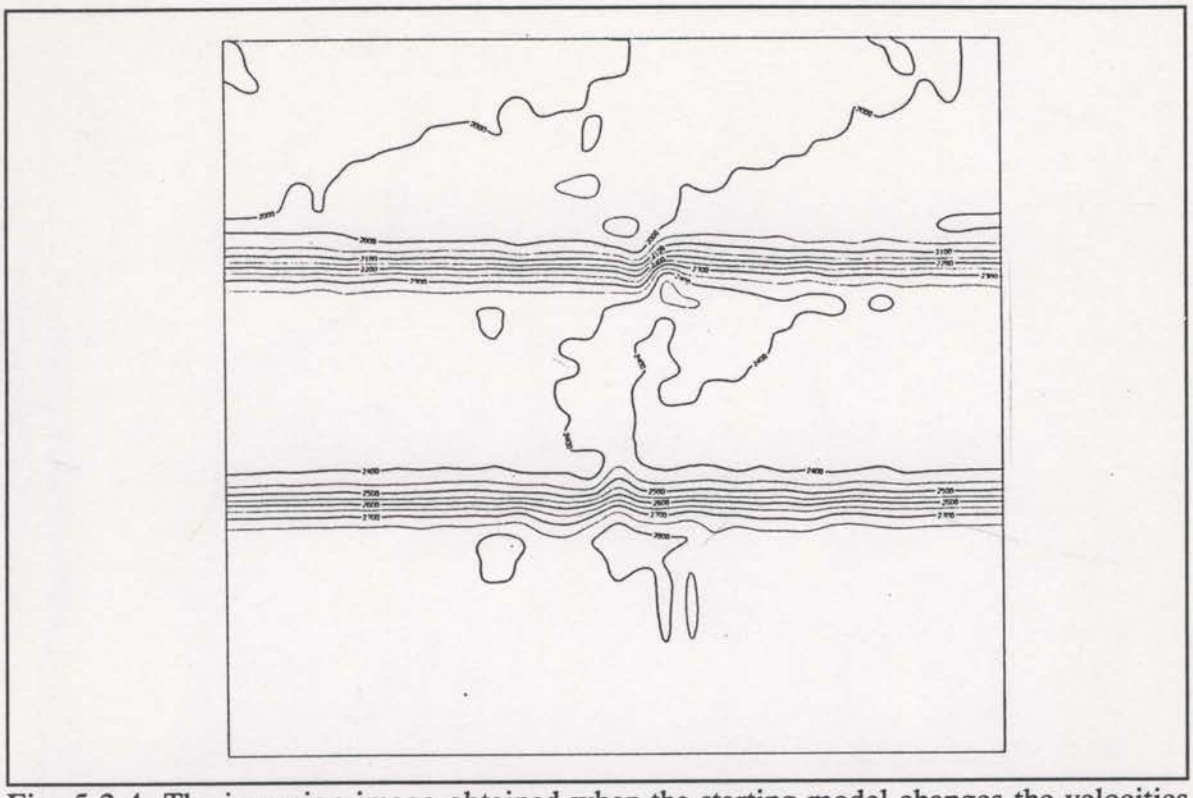


Fig. 5.2-4 The inversion image obtained when the starting model changes the velocities linearly over a 100 metre zone at the layer boundary.

it has moved downwards.

A closer look at this inversion reveals some apparent problems. The graph of the reduction of the constraint statistic (Figure 5.2-5) and change of the TEST parameter (Figure 5.2-6) shows disturbing oscillations and persistently high TEST values. When the constraint statistic increases, either the quadratic approximation within the subspace (see section 4) is poor or the distance moved within the subspace is large enough to be beyond the limit of reasonable approximation. A close investigation of this problem revealed that the distance limit in the inversion was allowed to be too large. The minimum of the quadratic approximation was far from the current location in image space, but the approximation is only valid for comparatively smaller distances. The oscillations observed in Figures 5.2-5 and 5.2-6 are signals of such approximation problems. The inversion code can be modified such that the distance limit is halved when oscillations are detected

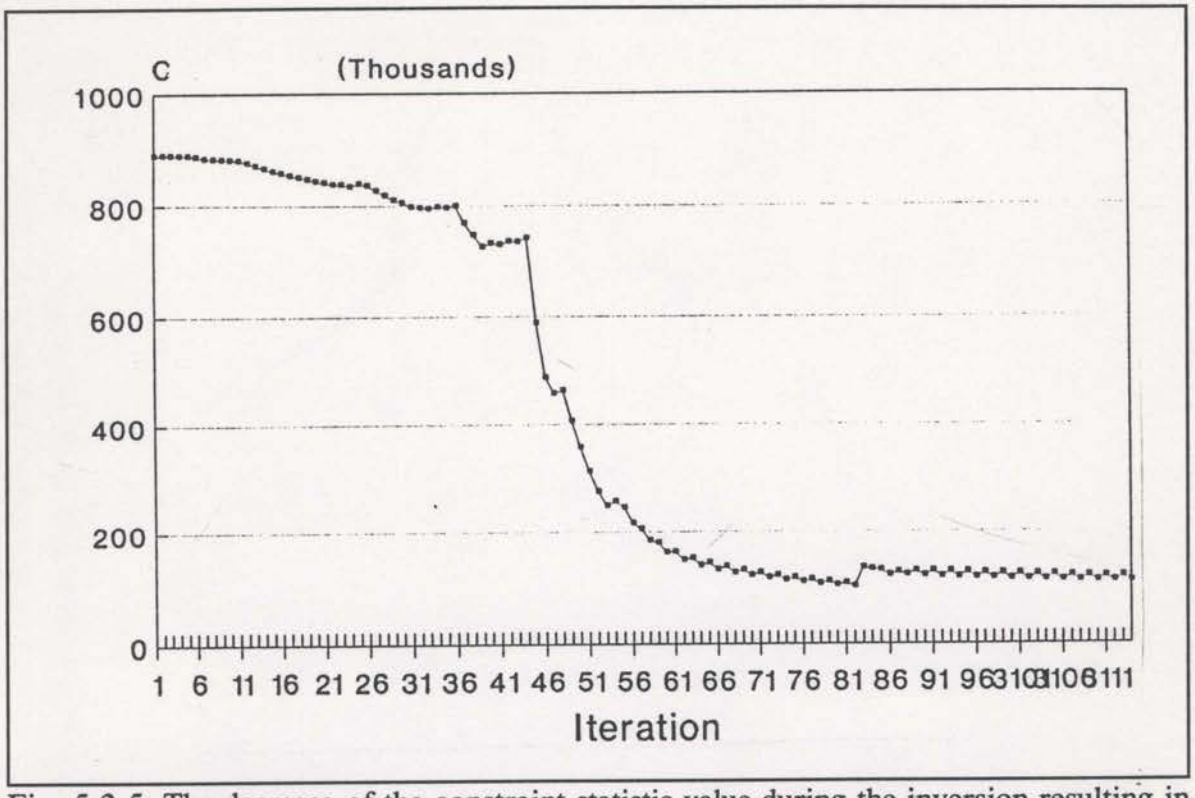


Fig. 5.2-5 The decrease of the constraint statistic value during the inversion resulting in the image of Figure 5.2-4.

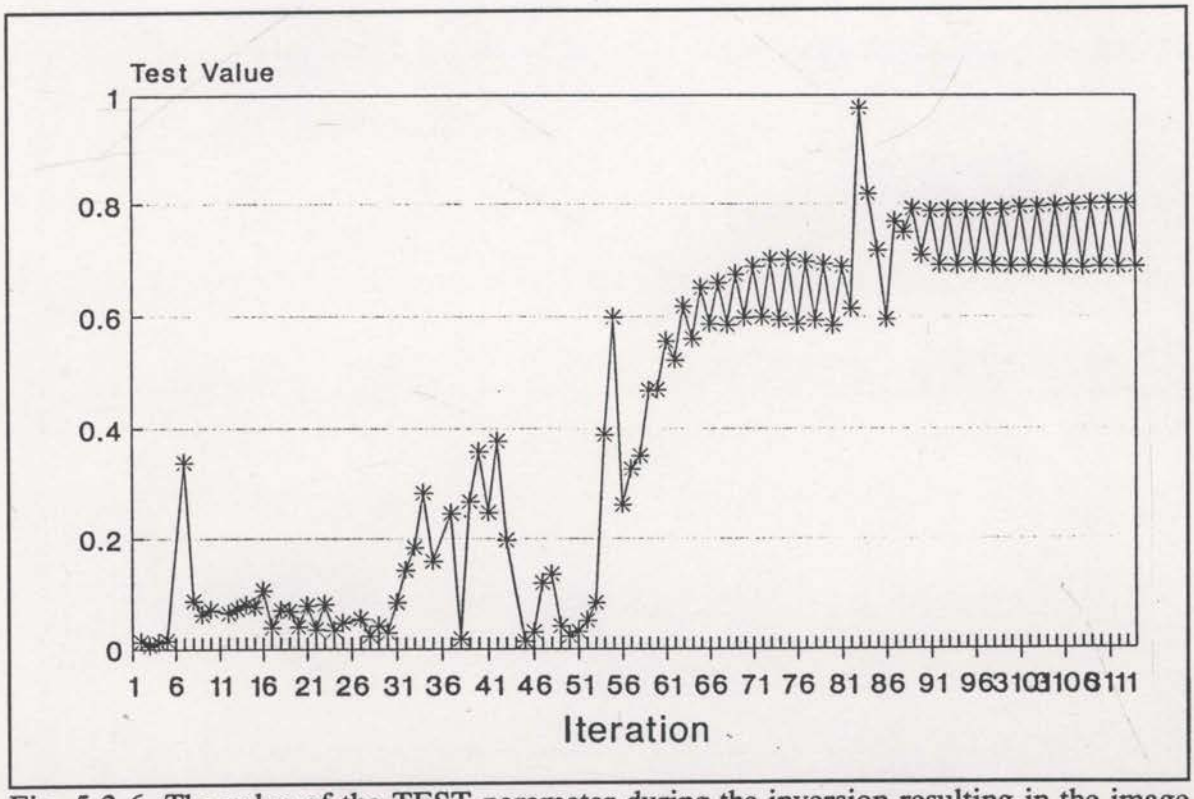


Fig. 5.2-6 The value of the TEST parameter during the inversion resulting in the image of Figure 5.2-4.

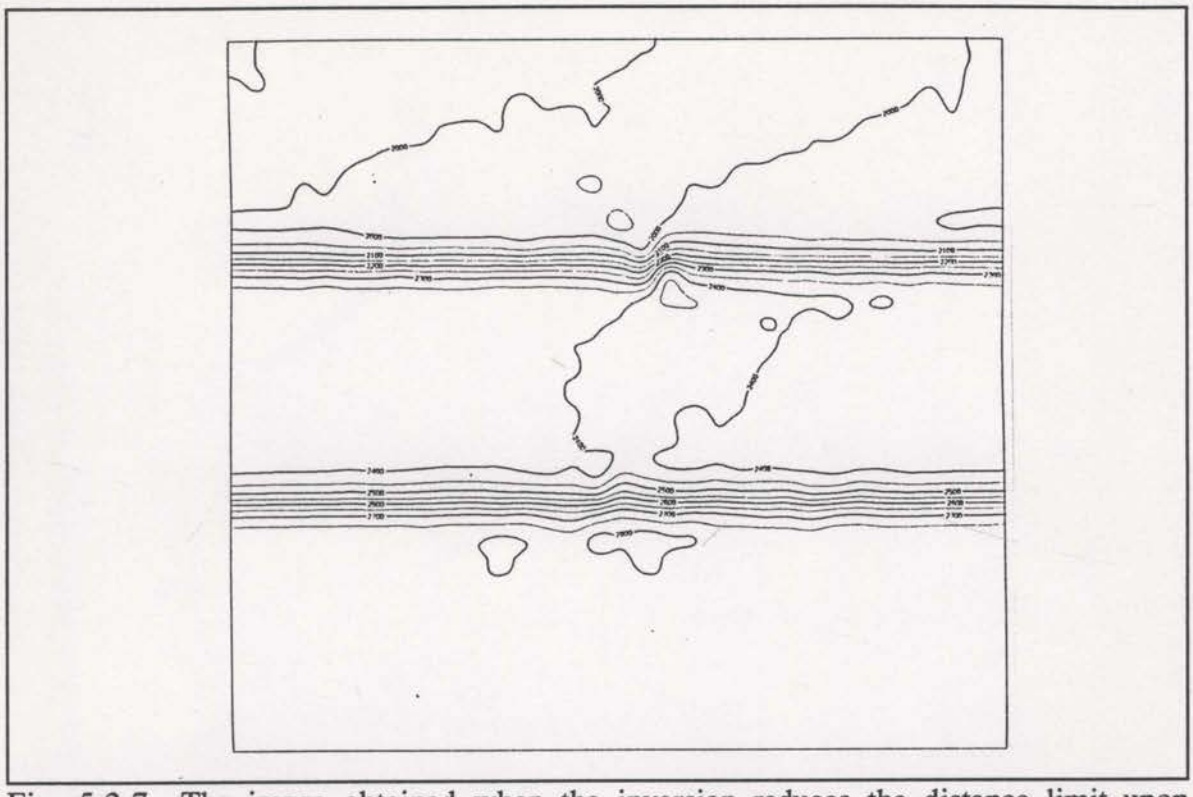


Fig. 5.2-7 The image obtained when the inversion reduces the distance limit upon detecting oscillations.

(this was discussed in section 4). Results of an inversion with this reducing distance limit can be seen in Figures 5.2-7, 5.2-8 and 5.2-9. Notice that the constraint statistic began to oscillate near the 55th iteration but then the resulting distance limit reductions produced a stable convergence by the 71st iteration. The TEST parameter also increases rapidly when the oscillations appeared (see Figure 5.2-9) but decreased rapidly when the distance limit was reduced. Oscillations of the constraint statistic and increases of the TEST parameter, are useful indicators of deficiencies in the local quadratic approximation, and form the basis of the automatic inversion control described in section 4.

These last two inversion results (Figures 5.2-4 and 5.2-7) effectively provide a comparison of results with and without entropy. The final TEST parameter values in Figures 5.2-6 and 5.2-9 show that the image of Figure 5.2-4 is far from the maximum entropy image whereas Figure 5.2-7 is a maximum entropy image. The final constraint

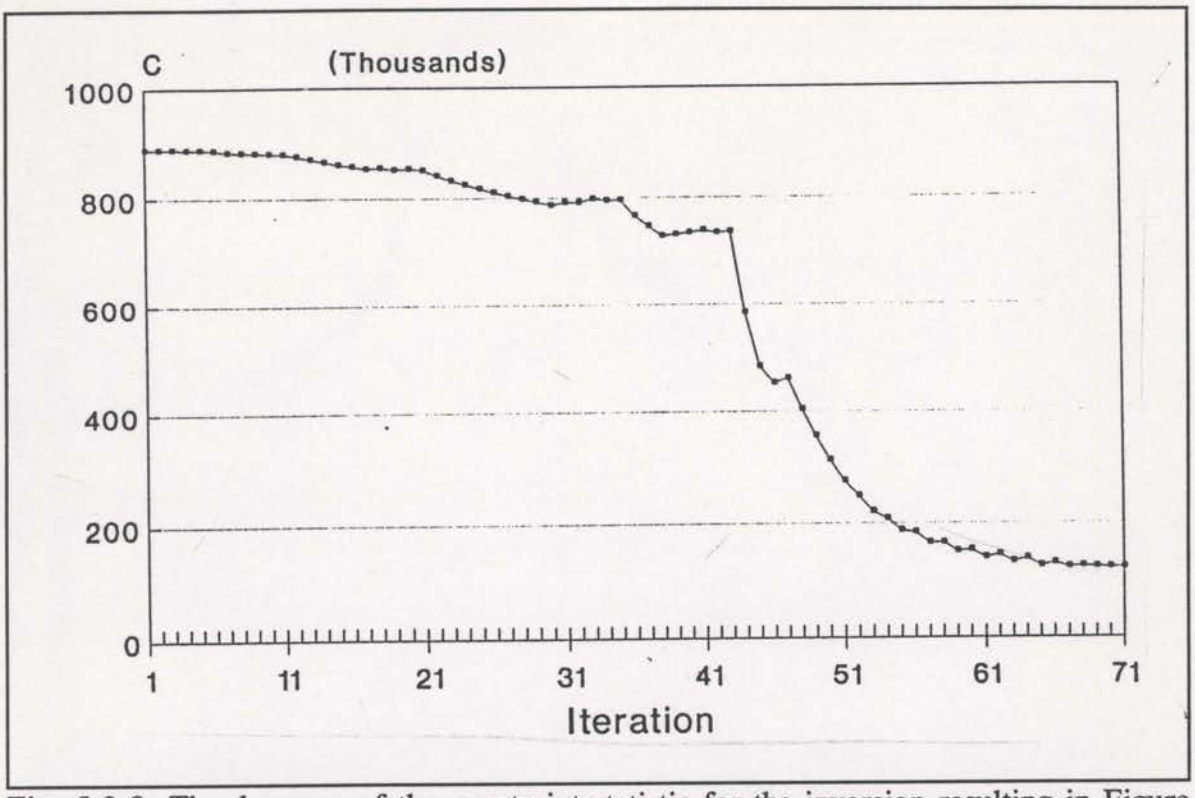


Fig. 5.2-8 The decrease of the constraint statistic for the inversion resulting in Figure 5.2-7.

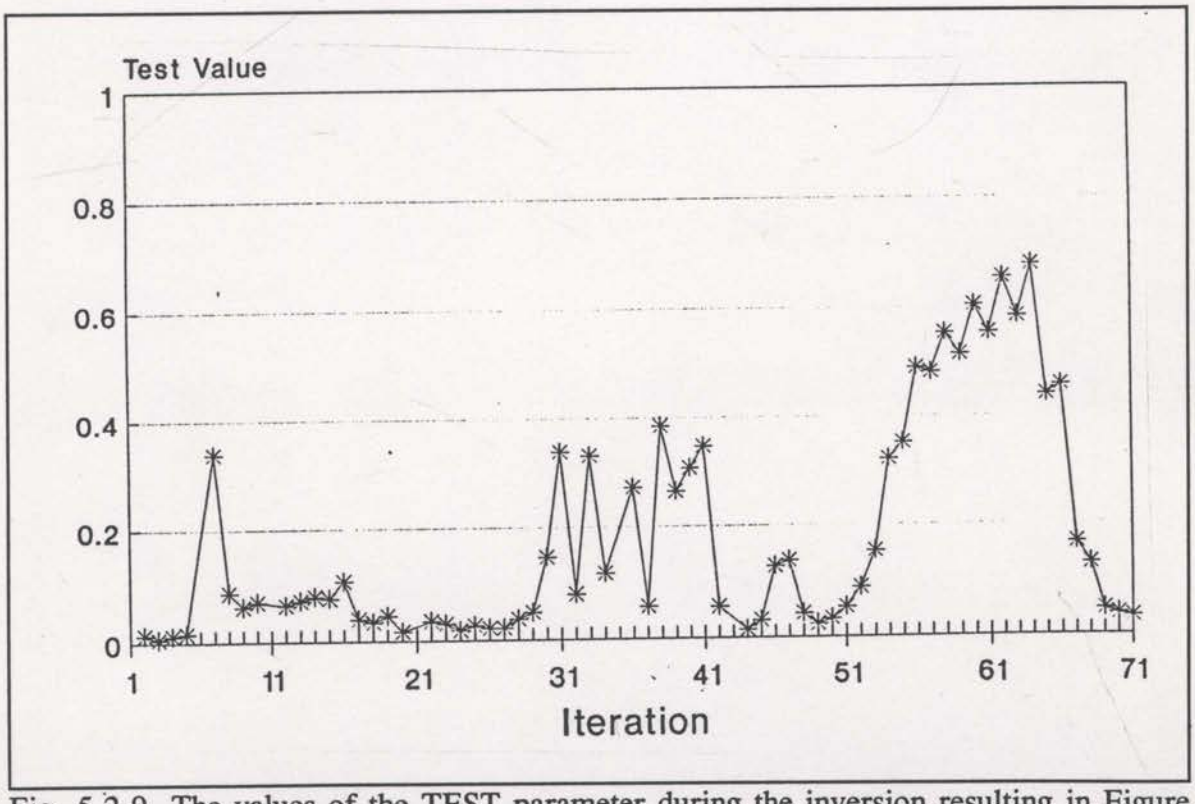


Fig. 5.2-9 The values of the TEST parameter during the inversion resulting in Figure 5.2-7.

statistic values are very similar (Figures 5.2-5 and 5.2-8). The maximum entropy image (Figure 5.2-7) exhibits less structure than the image of Figure 5.2-4. This extra structure represents the unwarranted image components that have been rejected by maximising the entropy function (see section 3). A more striking comparison between maximum entropy and non maximum entropy images was presented in section 3.7.

5.3 Tests with a layered/anomaly model

A synthetic data set was prepared for a subsurface model that consisted of simple layered geology and an anomalous rectangular area of faster velocity. The interval velocities and depths of these features are sketched in Figure 5.3-1. The anomalous zone has a velocity of 2700 m/s which is 20% above the velocity of the host layer. Random ray geometry was used to generate 800 traveltime data values. The source and receiver locations, as well as the surface intercept angles, were computed using random number generating software. This leads to a somewhat unrealistic data set but it should be one with a wide range of raypath geometries.

Using a simple linear velocity increase with depth as the starting model, the image of Figure 5.3-2 results (see section 2.5 for more details on this result). The constraint statistic values and the TEST parameter values for this inversion are shown in Figures 5.3-3 and 5.3-4. This result was obtained using the stages of decreasing smoothing as described in section 2.5. The inversion has clearly uncovered the presence of the anomaly without revealing the specifics of the anomaly or layers.

Without the assistance of stages of decreasing smoothing (see section 2.5), the inversion process is incapable of

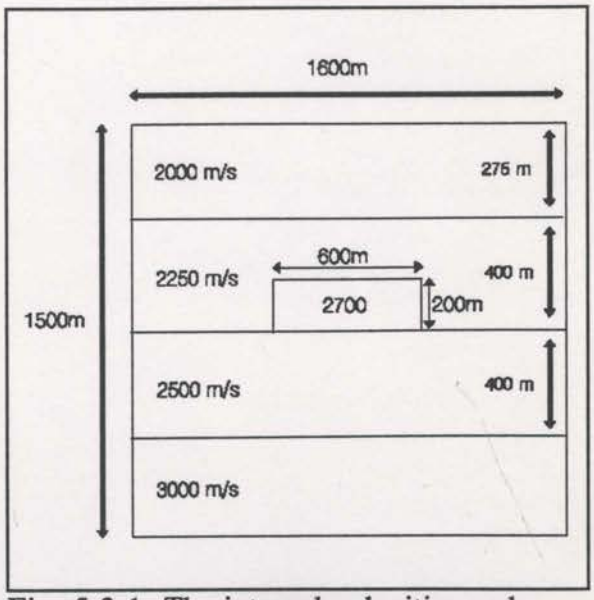


Fig. 5.3-1 The interval velocities and depths of the layered/anomaly model.

imaging the anomaly (see Figure 5.3-5). If entropy is not used as a constraint (see section 3) an image with additional spurious structure results (see Figure 5.3-6 and compare to Figure 5.3-2). Notice the extra structure at the lower left of the anomaly. The top corners of the anomaly are possibly better imaged, but the maximum entropy results (Figure 5.3-2) indicate that this additional structure is unjustifiable given the data provided.

With real reflection tomography problems, the state of knowledge of the subsurface velocities will generally be more sophisticated than a simple increase with depth. Where definite layers exist, as in the current model, the interval velocities of the layers will normally be estimated from nearby wells or CDP stacking velocities. In addition, the velocity of some layers will be known quite accurately (for example, the water layer in the marine exploration case). It will assist the inversion if such well

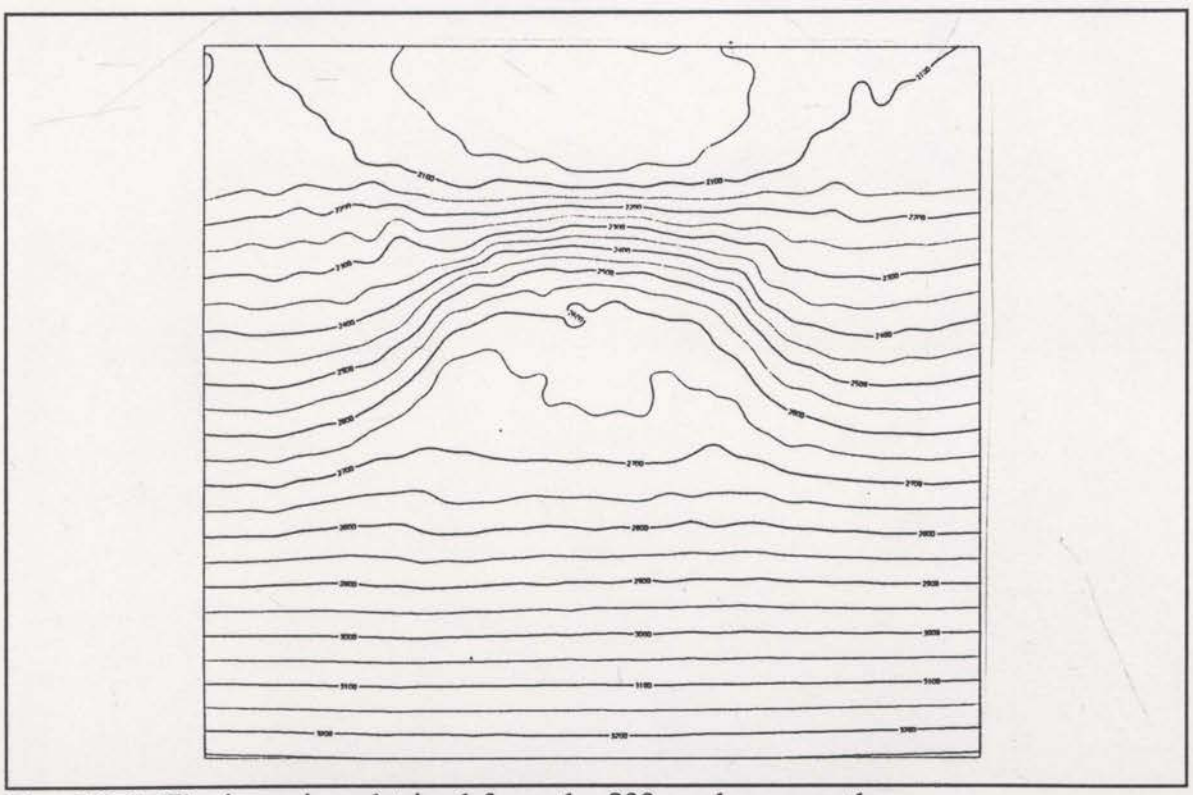


Fig. 5.3-2 The inversion obtained from the 800 random raypaths.

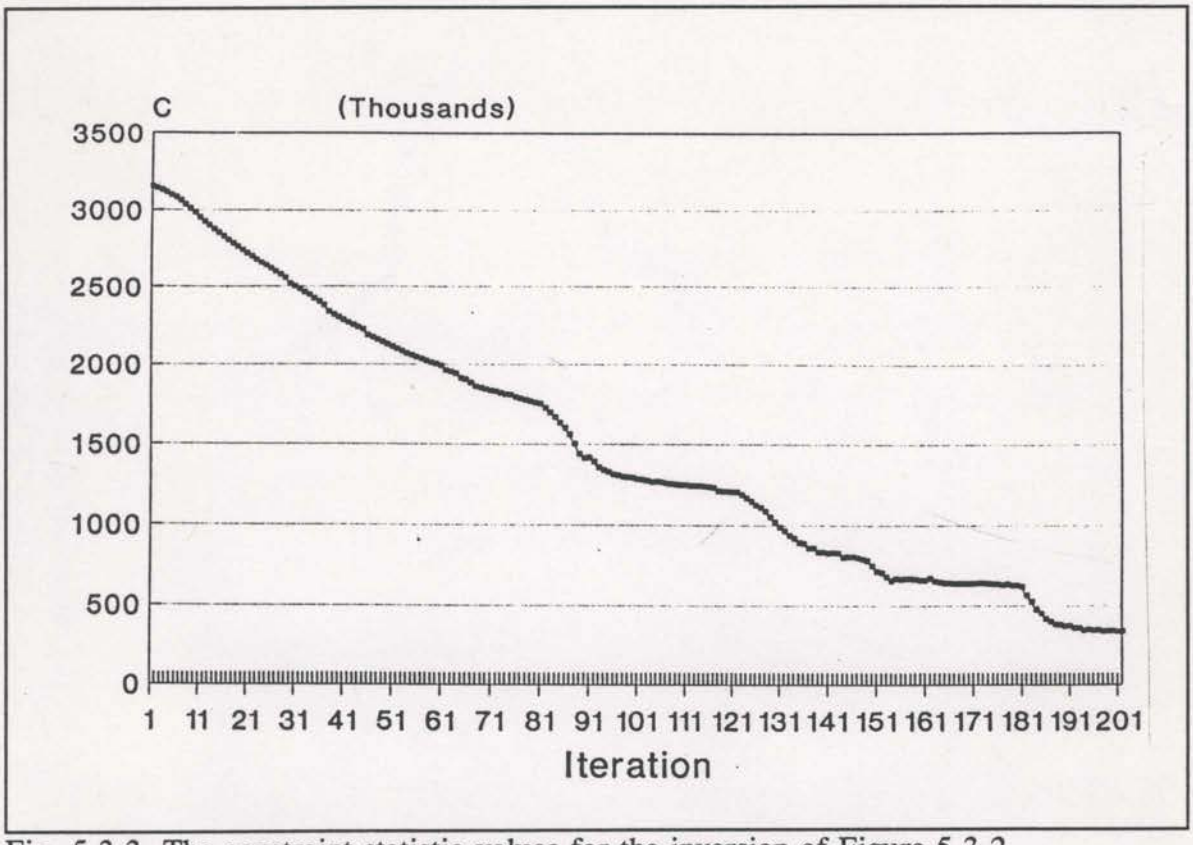


Fig. 5.3-3 The constraint statistic values for the inversion of Figure 5.3-2

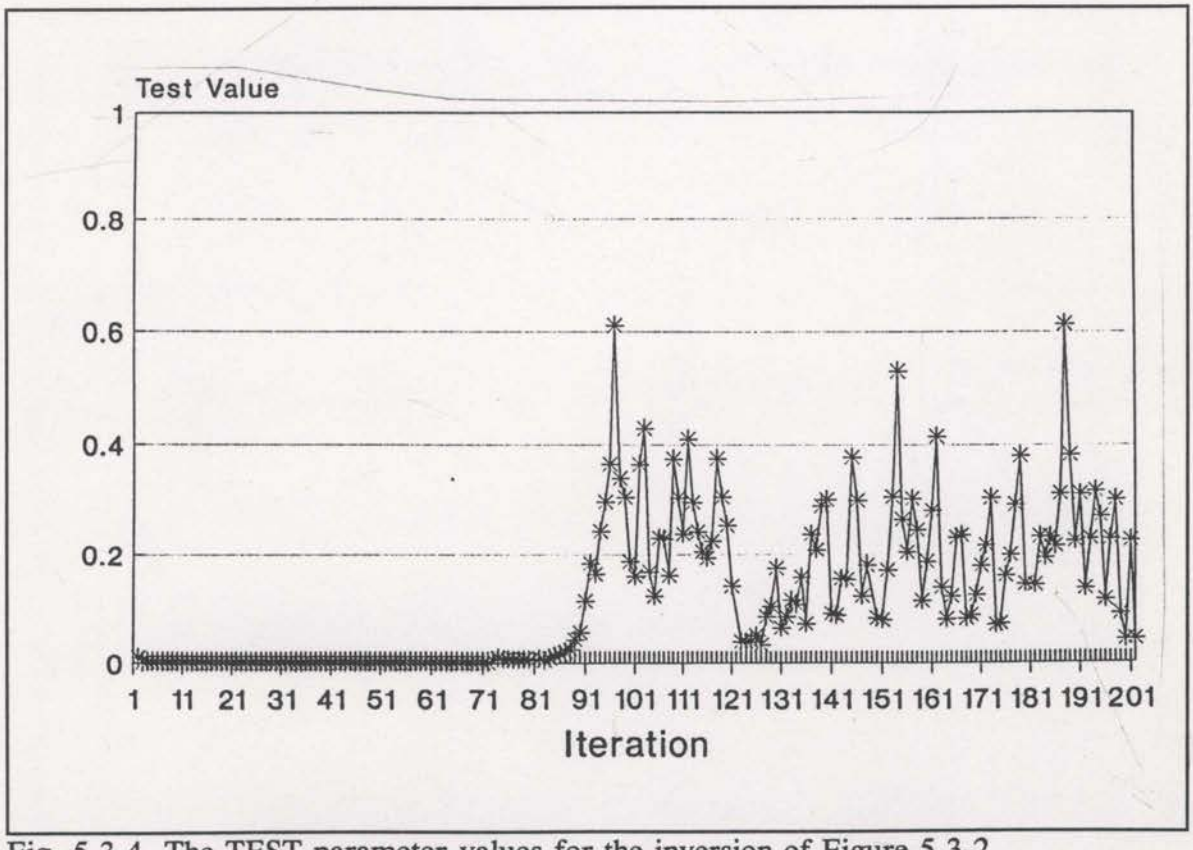


Fig. 5.3-4 The TEST parameter values for the inversion of Figure 5.3-2

known layers have their velocities held constant throughout the iterations. The starting model for the above results was a simple linear increase of velocity below the initial layer of 2000 m/s. If the velocity of this shallowest layer is not changed throughout the inversion, the image of Figure 5.3-7 results. The image of the anomaly is considerably sharper and the constant velocity layer around the anomaly is beginning to be properly imaged. Removing some of the velocities from the model parameters simplifies the inversion problem. However, if some velocities are fixed to incorrect values, the inversion may be prevented from achieving useful results.

Consider now a starting model consisting of the actual velocities of the layers, but without the presence of the anomaly. For the inversion of the data with this starting model, all of the velocities will be fixed except for those of the layer that contains the anomaly. The final image of this inversion is displayed in Figure 5.3-8. As expected, this image is superior to all earlier images.

There are many reasons why correct starting velocities can be modified during any given iteration of the inversion. These include noise, incorrect ray locations, ambiguity and a lack of ray coverage. It is generally assumed that such incorrect modifications will be insignificant when compared to the correct changes. However, removing the possibility of incorrect velocity updates in areas of relative certainty can help improve inversion results.

In real reflection tomography problems, the surface incidence angles will not be known exactly. Such circumstances can be evaluated using the current synthetic data if errors are added to the surface incidence angles of the rays. Figure 5.3-9 shows the image obtained after 10% gaussian noise has been added to the surface incidence angles. Figure 5.3-10 displays the results for 20% gaussian noise. With 10% noise the

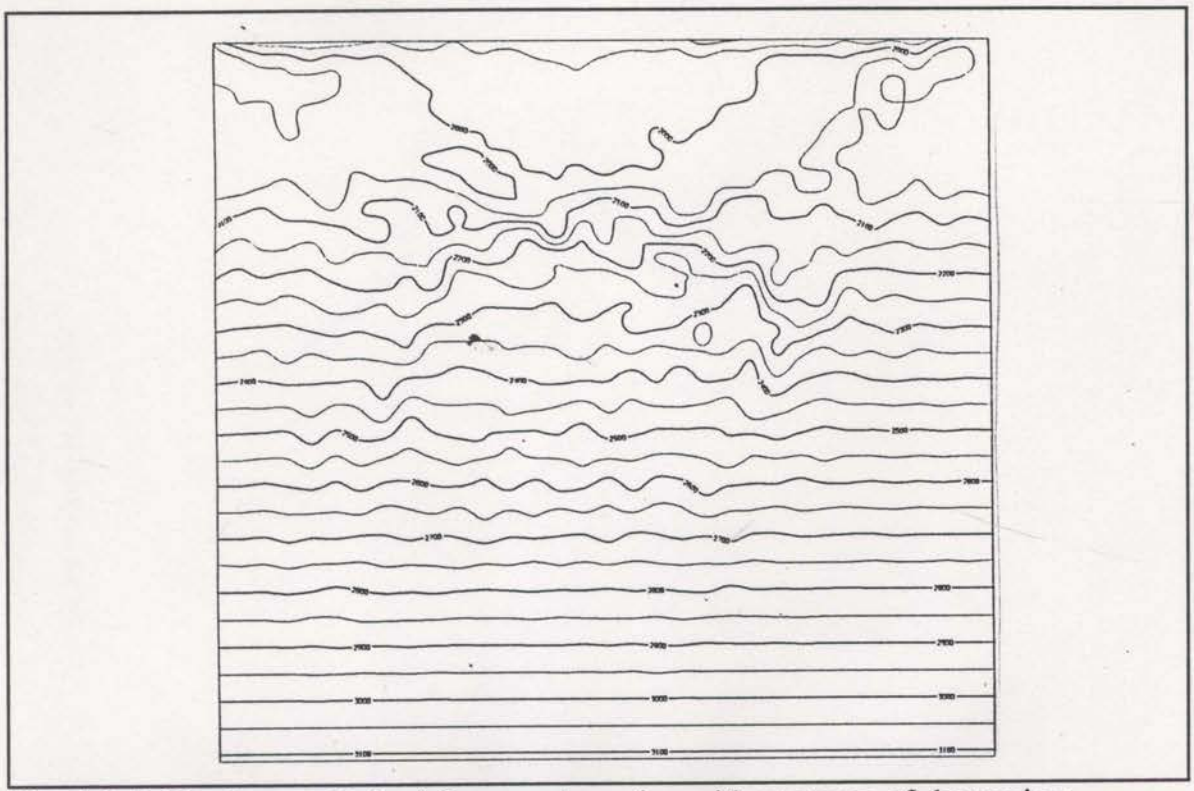


Fig. 5.3-5 The image obtained from an inversion without stages of decreasing smoothing. (Final C value approximately 700,000)

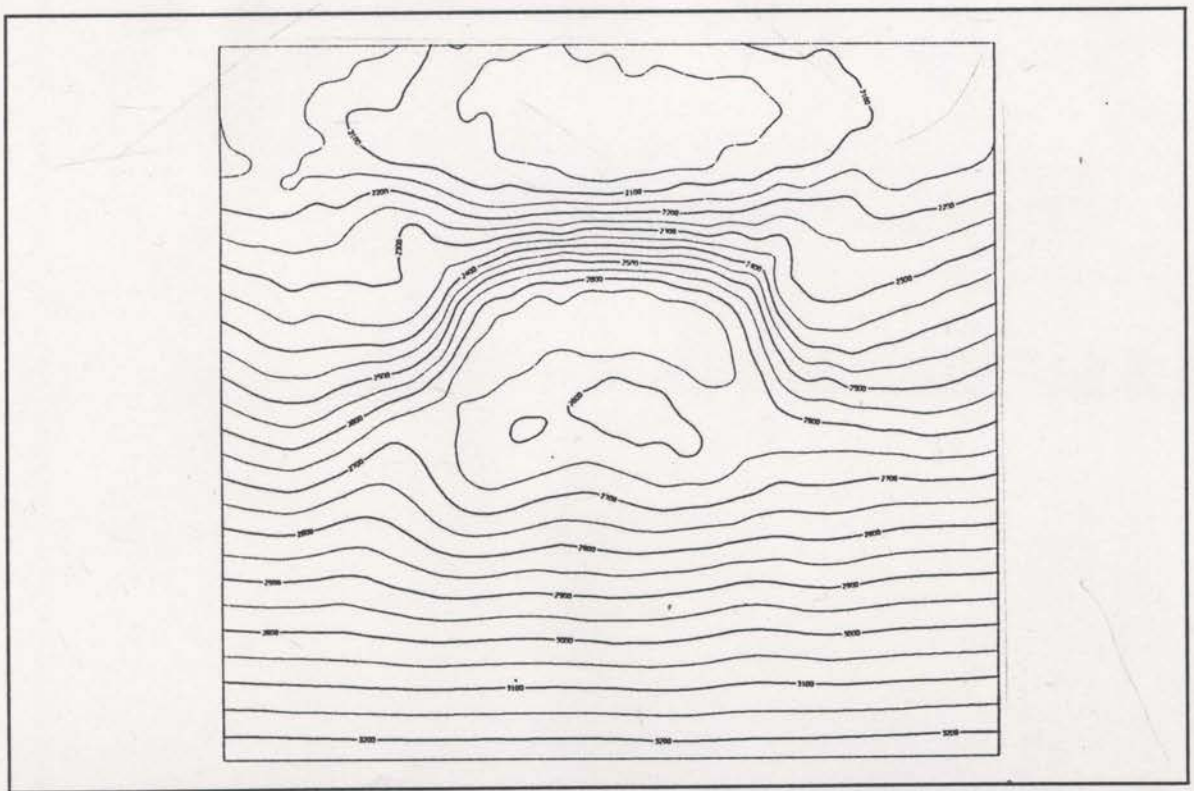


Fig. 5.3-6 The image obtained from an inversion without maximum entropy constraints. (Final C value approximately 200,000)

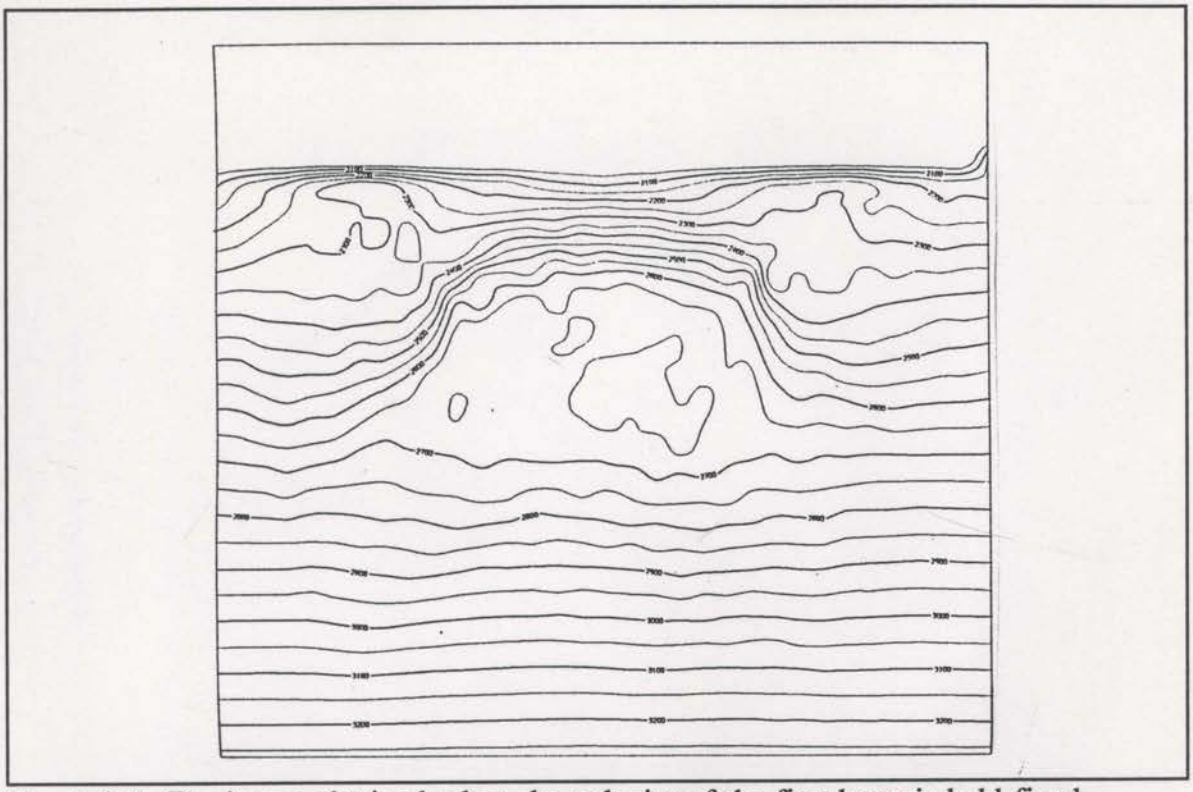


Fig. 5.3-7 The image obtained when the velocity of the first layer is held fixed. (Final C value approximately 200,000)

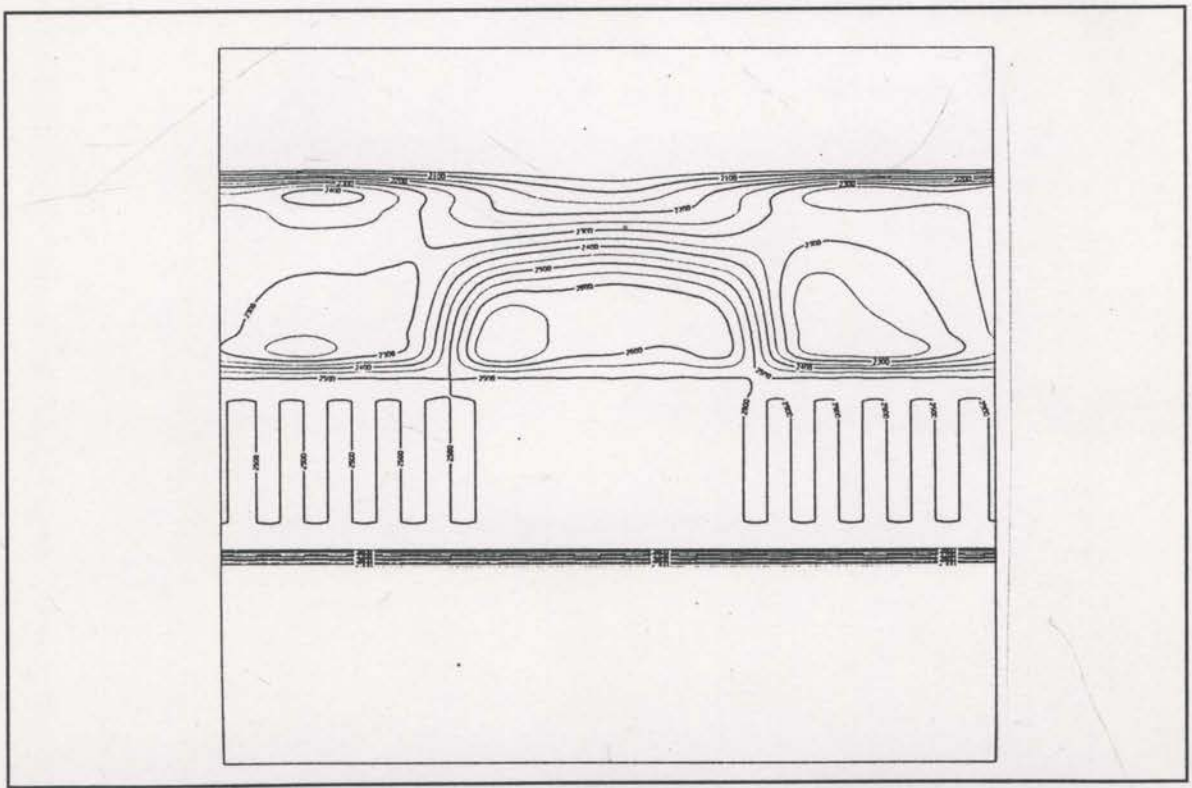


Fig. 5.3-8 The image obtained when the correct layer velocities are used as the starting model. All but the velocities for the anomalous layer were held constant at there correct values.

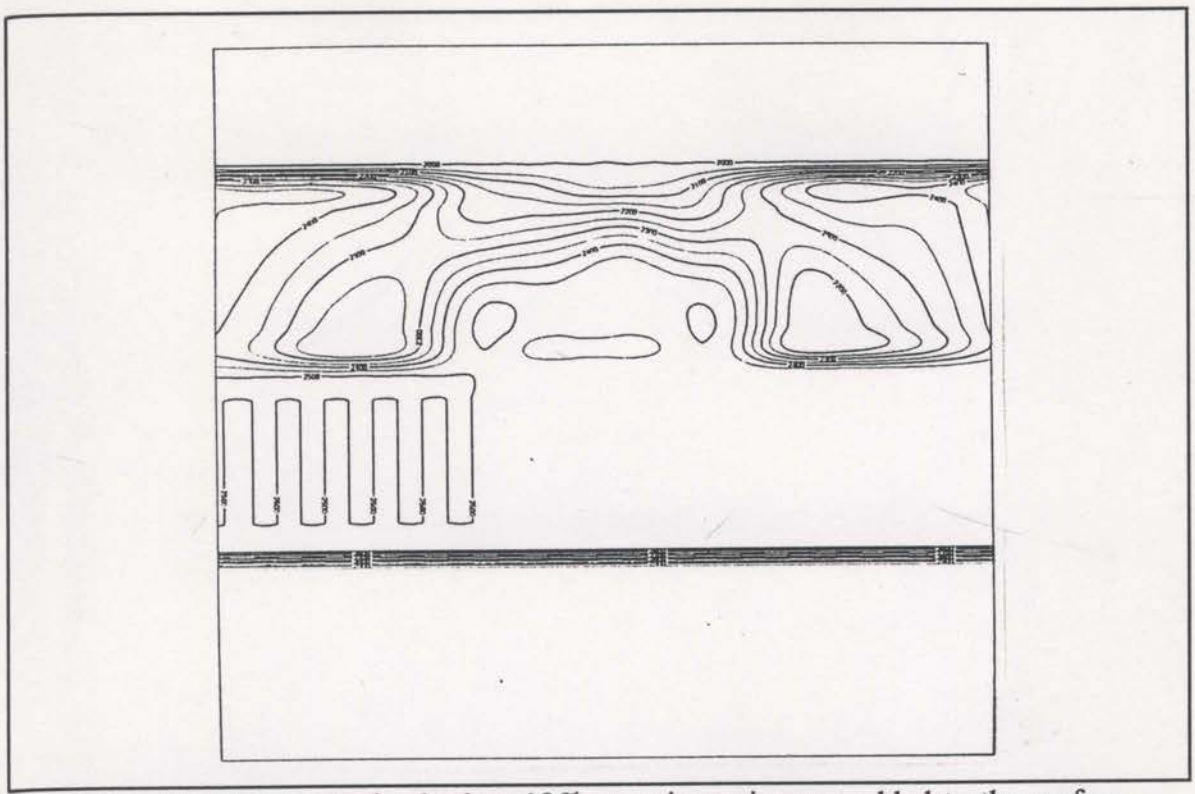


Fig. 5.3-9 The image obtained when 10% gaussian noise was added to the surface incidence angles. Compare to Figure 5.3-8

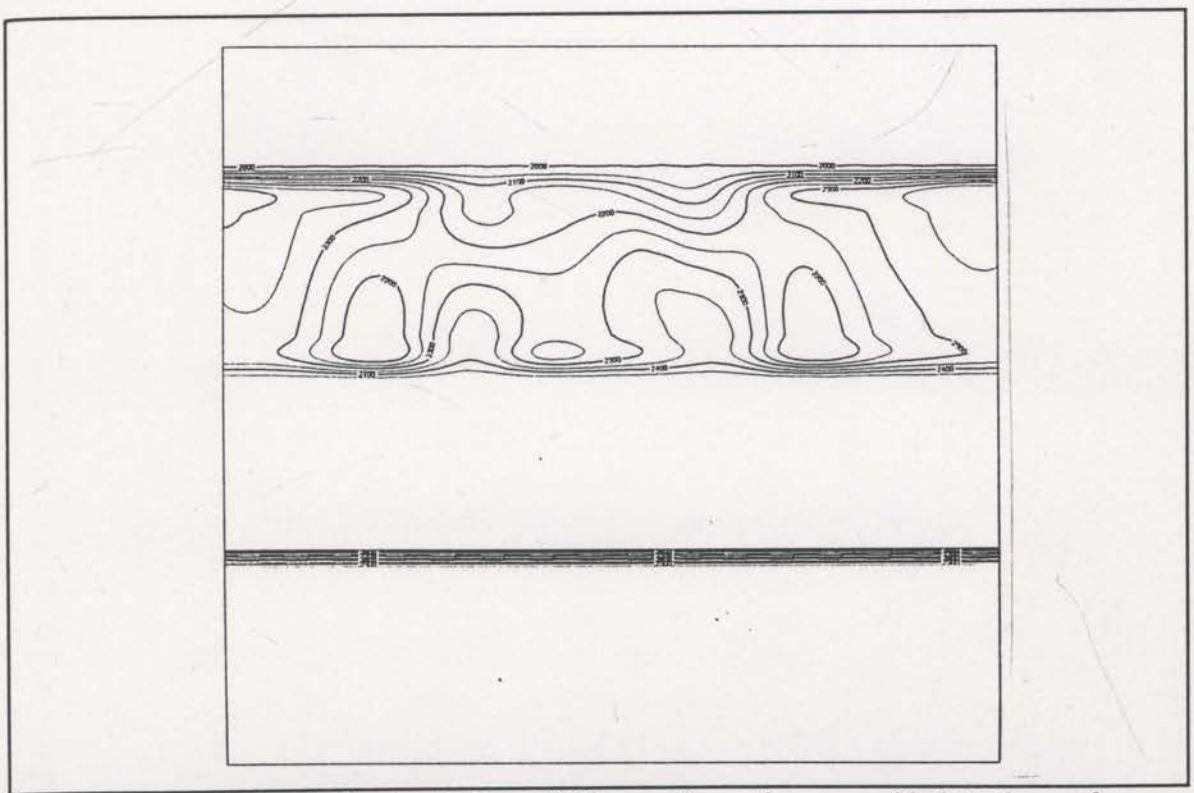


Fig. 5.3-10 The image obtained when 20% gaussian noise was added to the surface incidence angles. Compare to Figure 5.3-8

results are clearly degraded but still useful. With 20% noise, the results are of little use. The amount of surface incidence angle noise present in real data sets is clearly of significant importance.

5.4 Line A - Field data example

Real data examples present many problems that are not of concern with synthetic data examples. The data must be processed such that the automatic picking algorithm (see section 2.3) will pick reliable traveltimes and dips. Multiple energy and other noise must be sufficiently attenuated. Even with good data quality there will inevitably be some dip measurement errors. The examples in section 5.3 showed that these errors do not have to be much larger than 10% to cause serious problems. Added to these difficulties is the task of finding an adequate starting velocity model. Deep well control can make this generally troublesome task much easier.

Pre-stack depth migration provides a means for verifying subsurface velocity fields (see Al-Yahya, 1989). Iterative pre-stack depth migration can be used to define the subsurface interval velocity in the absence of well control. The real data example to be reviewed here, Line A, was the subject of exactly this type of iterative velocity evaluation. The final pre-stack depth migration is shown in Figure 5.4-1, and the corresponding velocity field is shown in Figure 5.4-2. This velocity field is then used as the starting velocity model needed for reflection tomography. The aim of the tomographic inversion is to improve this velocity model. An improved velocity model would produce a better pre-stack depth migrated image.

Traveltime and apparent dips were picked automatically using instantaneous frequency and wavenumber (see section 2.3). The many thousands of picks that resulted were decimated to approximately 2000 which were inverted using the entropy principle and stages of decreasing smoothing (half-widths of the gaussian smoothers

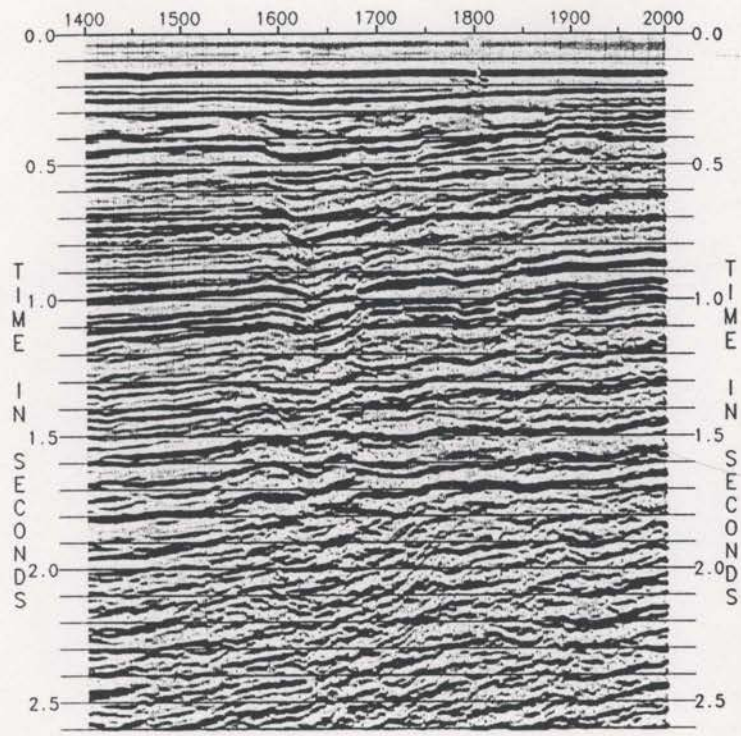


Fig. 5.4-1 The pre-stack depth migrated data - Line A. The velocity model used was obtained from iterations of pre-stack depth migration.

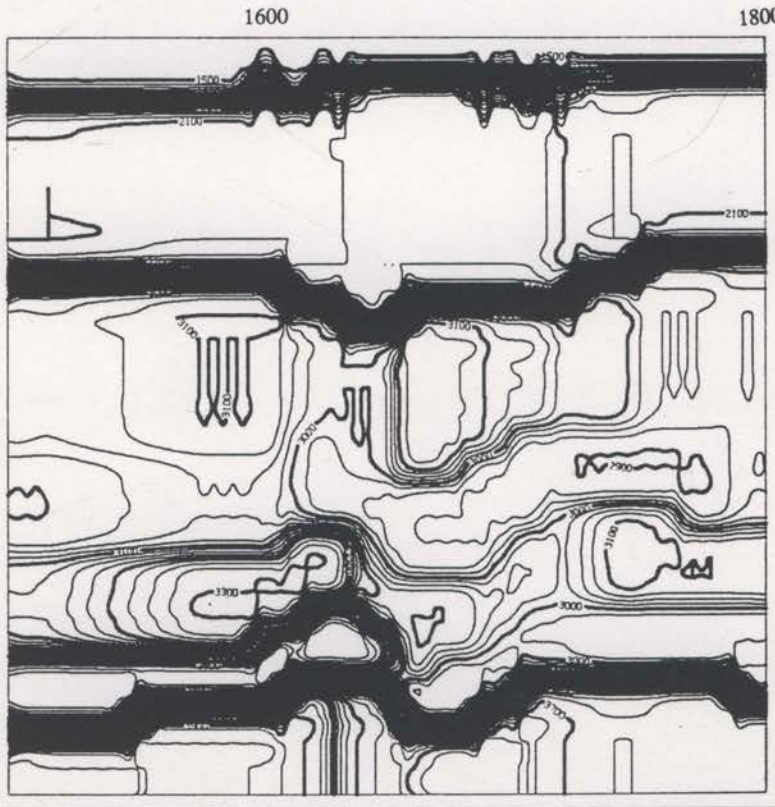


Fig. 5.4-2 The interval velocity model used for the pre-stack depth migration of Figure 5.4-1

were 400, 200, 100 and 0 metres). The final velocity field is shown in Figure 5.4-3.

The inversion result (Figure 5.4-3) exhibits higher frequency structure than the starting model (Figure 5.4-2). It is not possible to accurately infer high frequency structure during velocity model building with iterations of pre-stack depth migration. The rate of spatial change of velocity that is possible with tomographic inversion is limited only by the chosen discrete node spacing and the quality of the data. Generally, the changes made appear realistic and the subsequent pre-stack depth migration results of Figure 5.4-4 demonstrate an improved image. Notice the stronger image of the shallower events and the changed attitude near the major faulting toward the left.

Accurate pre-stack depth migrations require accurate interval velocity fields and are characterised by a lack of residual moveout on the migrated common location

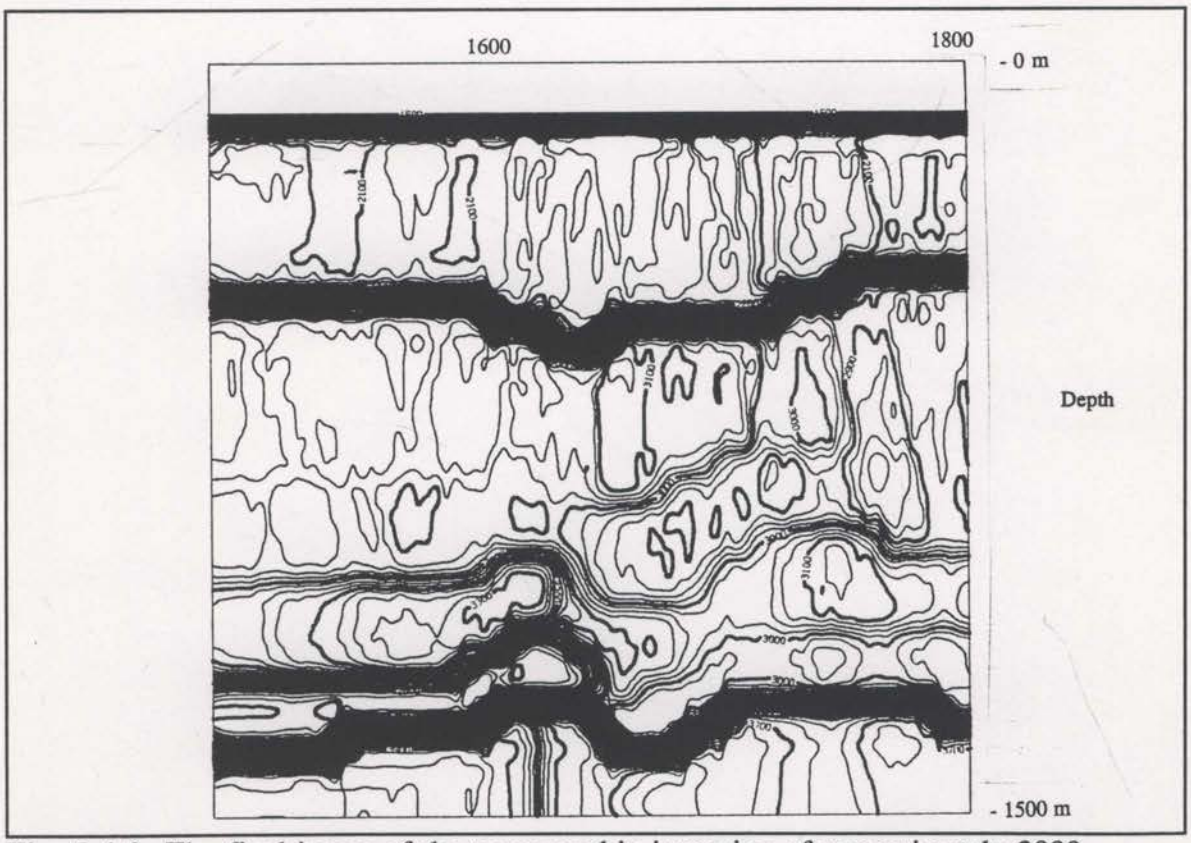


Fig. 5.4-3 The final image of the tomographic inversion of approximately 2000 traveltimes.

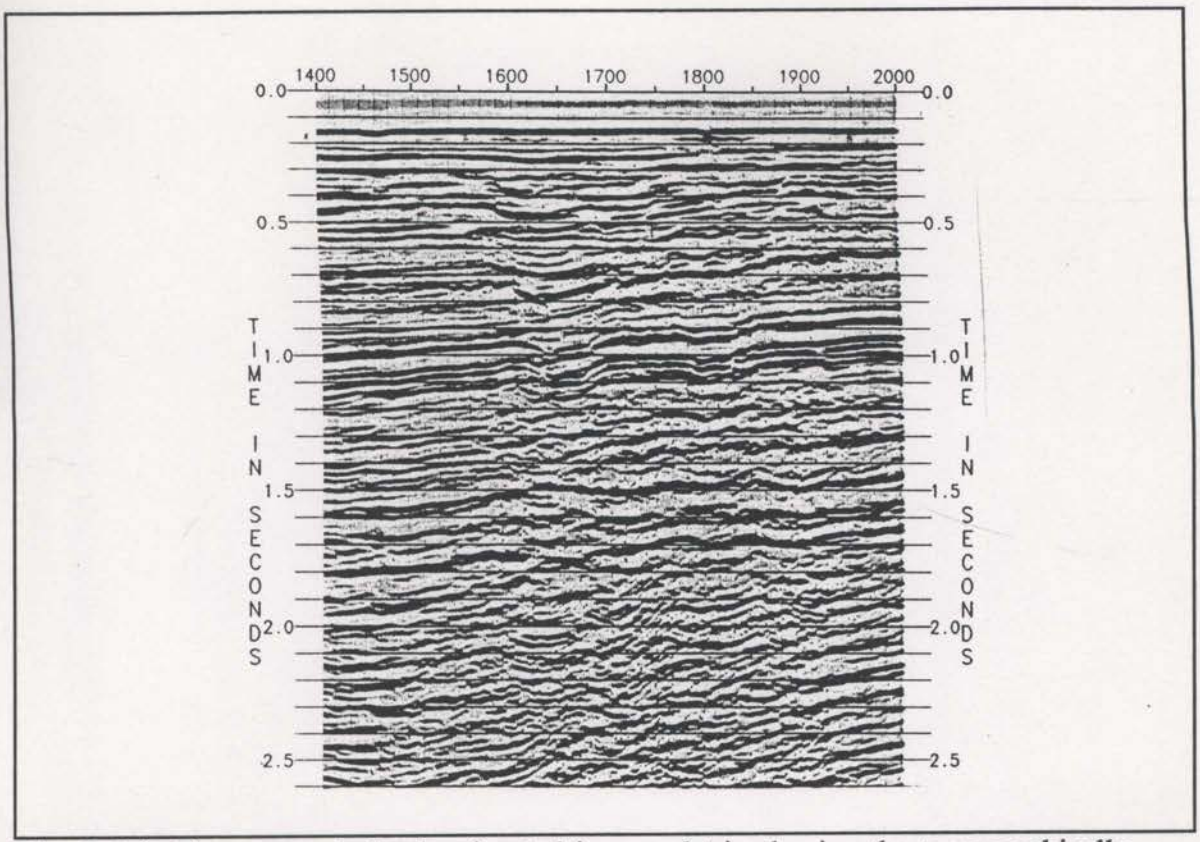


Fig. 5.4-4 The pre-stack depth migrated image obtained using the tomographically updated velocities of Figure 5.4-3

gathers (see Al-Yahya, 1989). Figures 5.4-5 and 5.4-6 display some migrated common location gathers using the velocity models of Figures 5.4-2 and 5.4-3. The data created with the tomographically derived velocities (Figure 5.4-6) exhibits generally flatter events, confirming that the updates made by the reflection tomography algorithm have been accurate.

It appears that reflection tomography can be useful in the role of updating previously estimated interval velocity fields. It has the potential to add higher frequency velocity variations than are obtainable by conventional means.

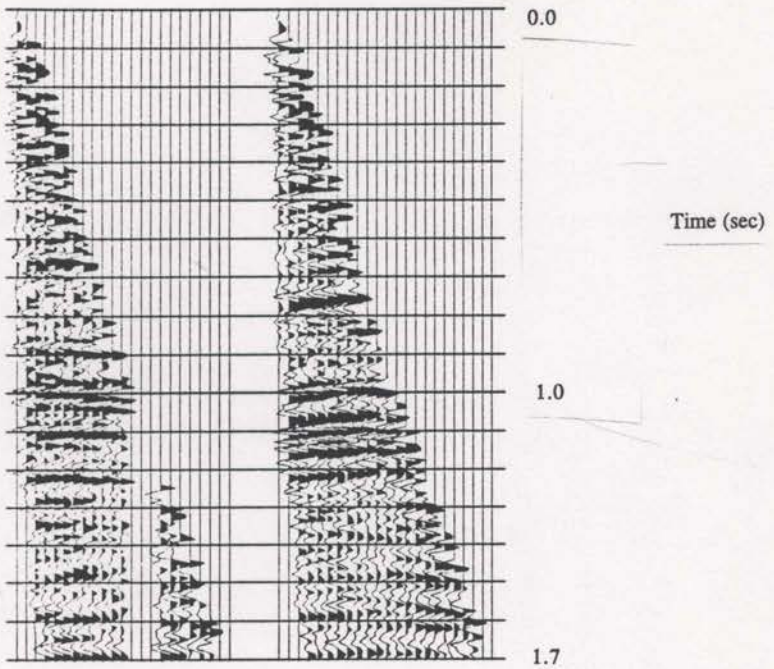


Fig. 5.4-5 Migrated common location gathers using the starting velocity model (Figure 5.4-2).

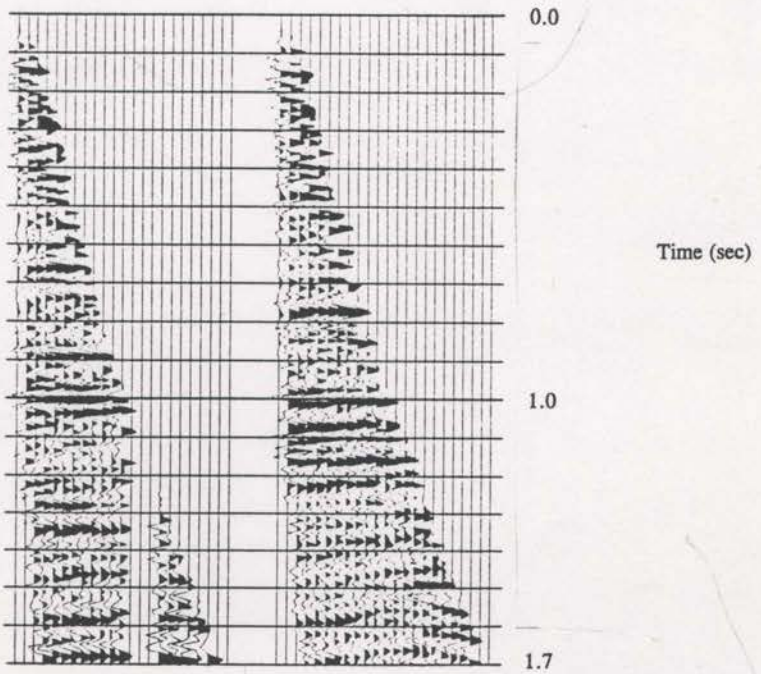


Fig. 5.4-6 Migrated common location gathers obtained using the tomographically updated velocity field.

5.5 Line B - Field data example

The example reviewed in this section was processed and evaluated in a similar way to Line A (described in the previous section). However, some further tests were performed on this line to explore a way of coping with the lack of precise knowledge regarding the depths of the layer boundaries.

The interval velocity model obtained from iterative pre-stack depth migration is shown in Figure 5.5-1, and the corresponding migrated image is in Figure 5.5-2. The processed gathers were picked using the complex attribute picker described in Section 2.3 . Approximately 2000 of these traveltimes were supplied to the reflection tomography software that inverted them using stages of decreasing smoothing and maximum entropy constraints. The final velocity field image is displayed in Figure 5.5-3. The corresponding pre-stack depth migration (Figure 5.5-4) shows improved imaging - especially at the edges of the major faulting (around times 0.7 to 1.0 seconds). As with Line A (see section 5.4), migrated common location gathers show generally less residual moveout (see Figures 5.5-5 and 5.5-6). It can be concluded that the tomographic velocity updates are accurate.

It was shown, using synthetic data in section 5.2, that if the initial velocity model has a layer boundary specified at the incorrect depth, the inversion may be stifled. The required velocity changes may be too large to be accommodated. With both the current data and the data of Line A in the previous section, no well control was available so the depths to the layer boundaries were not accurately known. Only the two-way traveltimes of the reflections are known accurately. If the initial guess of

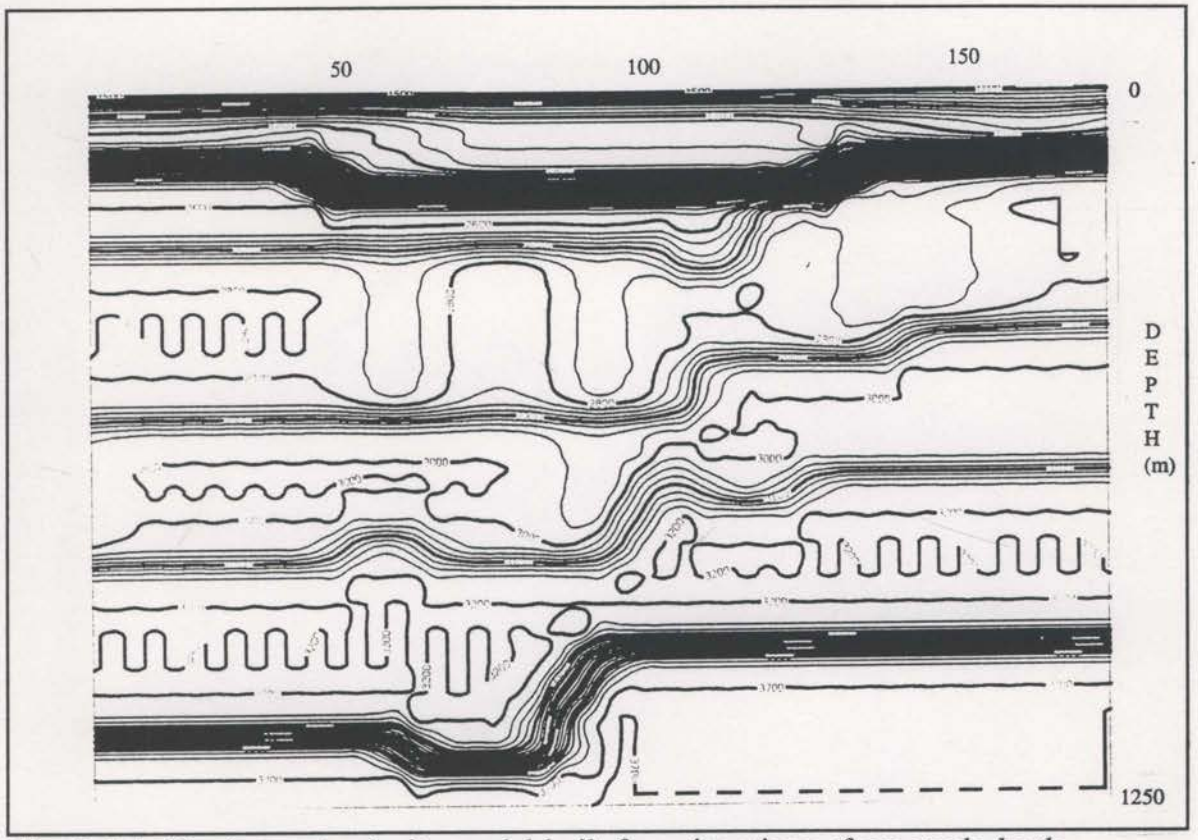


Fig. 5.5-1 The interval velocity model built from iterations of pre-stack depth migration.

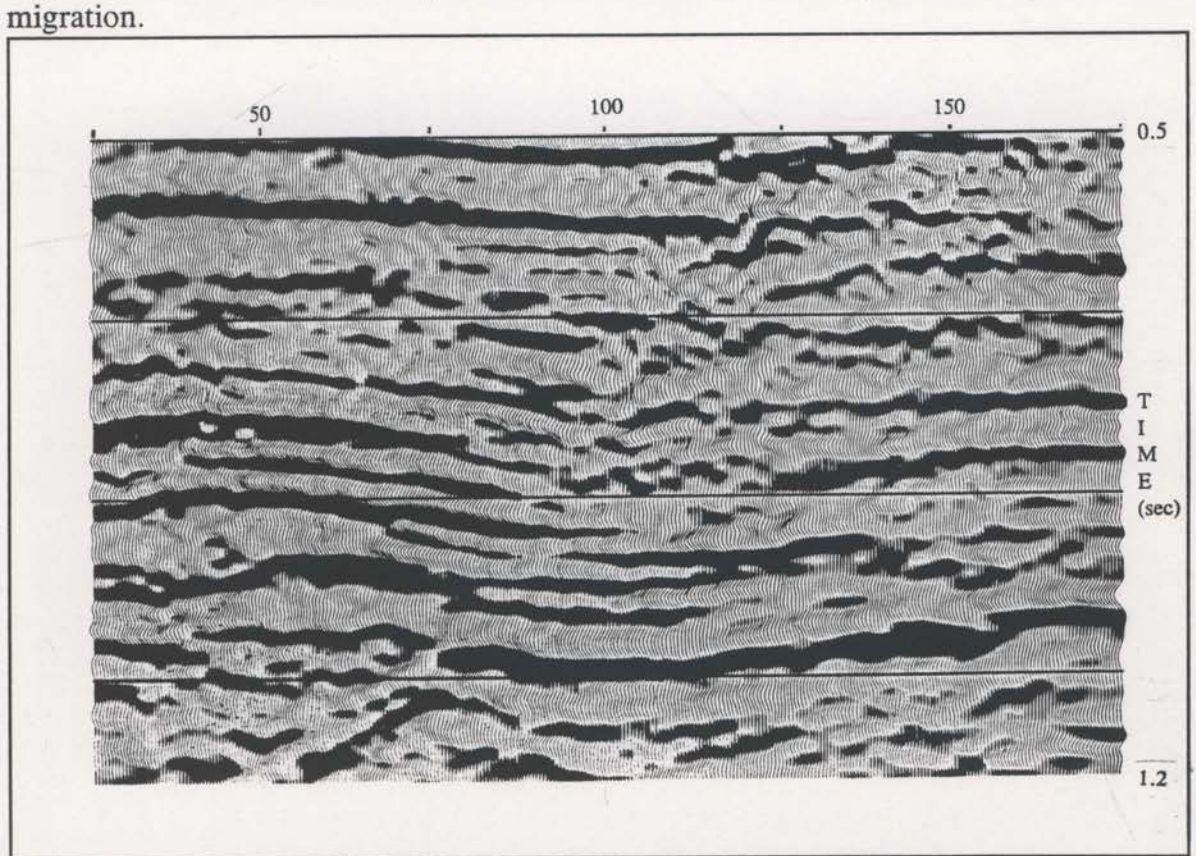


Fig. 5.5-2 The pre-stack depth migrated data obtained using the velocity model of Figure 5.5-1

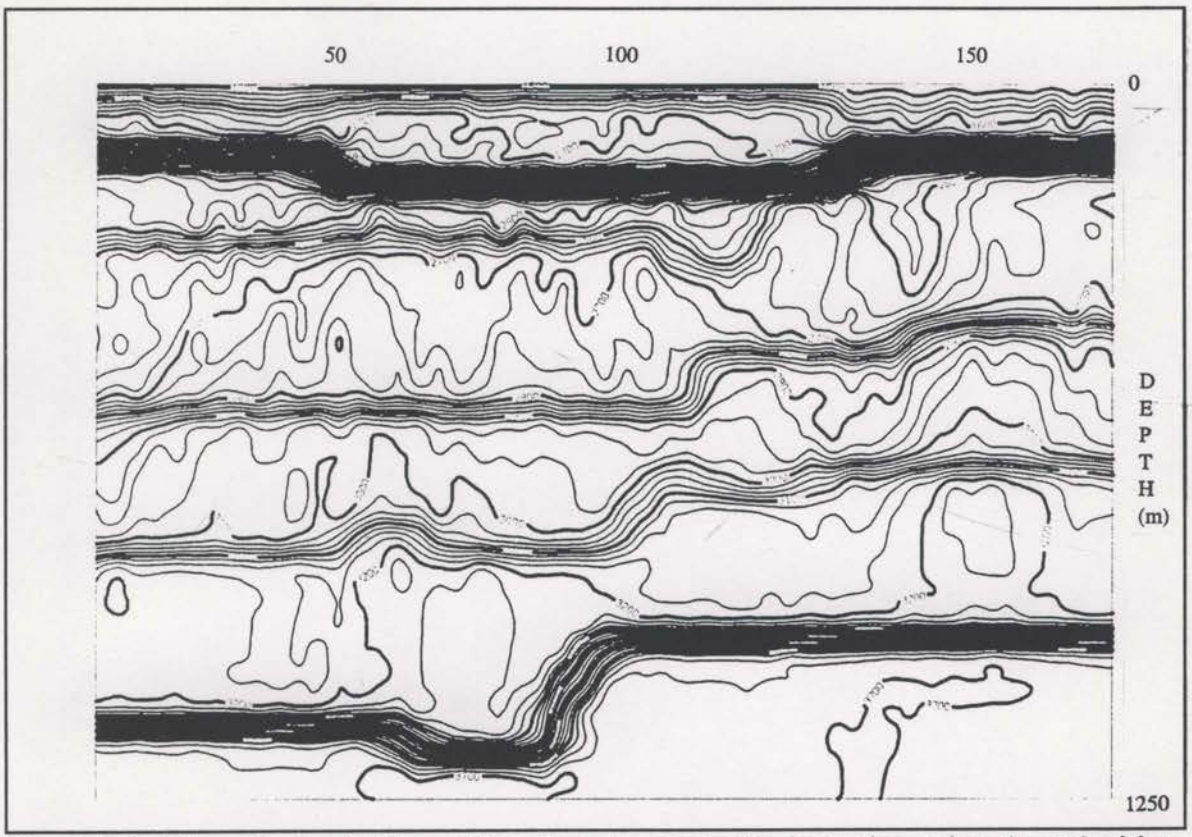


Fig. 5.5-3 The velocity field resulting from tomographic inversion using the velocities of Figure 5.5-1 as the starting model.

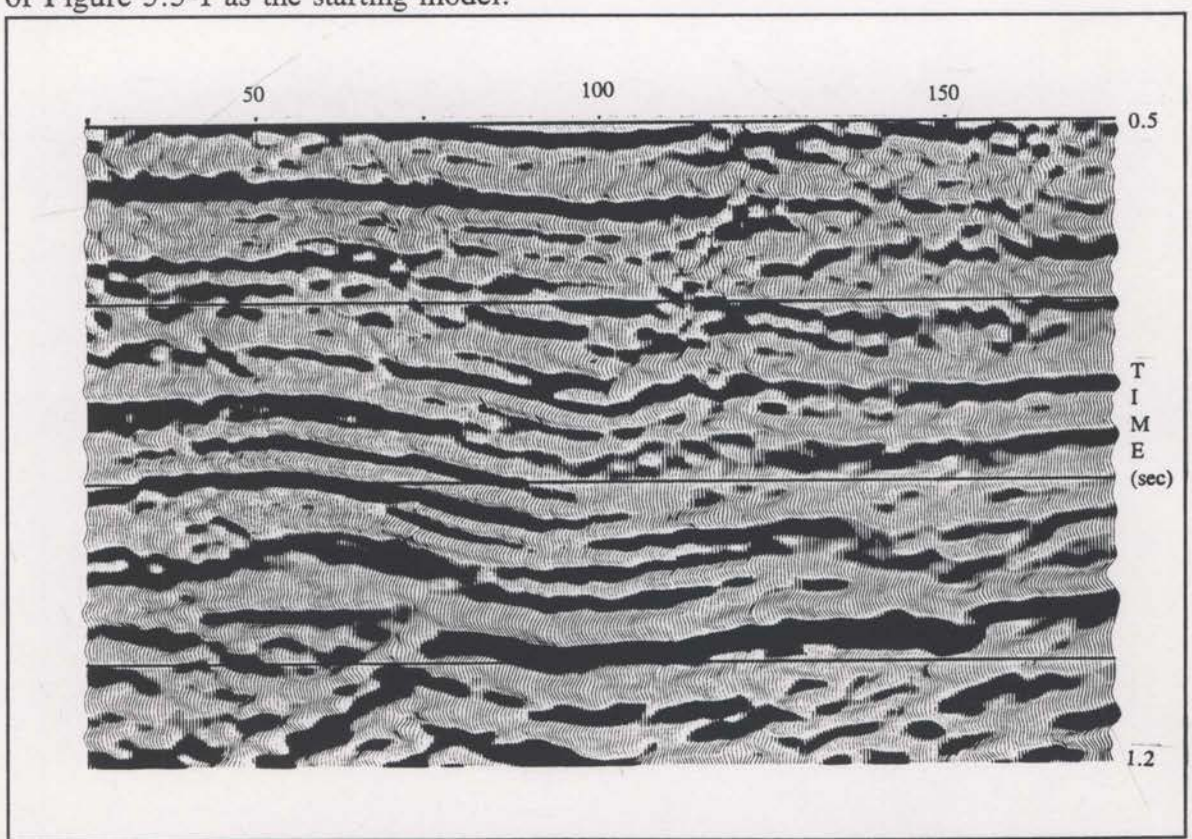


Fig. 5.5-4 The pre-stack depth migrated image obtained using the tomographically updated velocities of Figure 5.5-3.

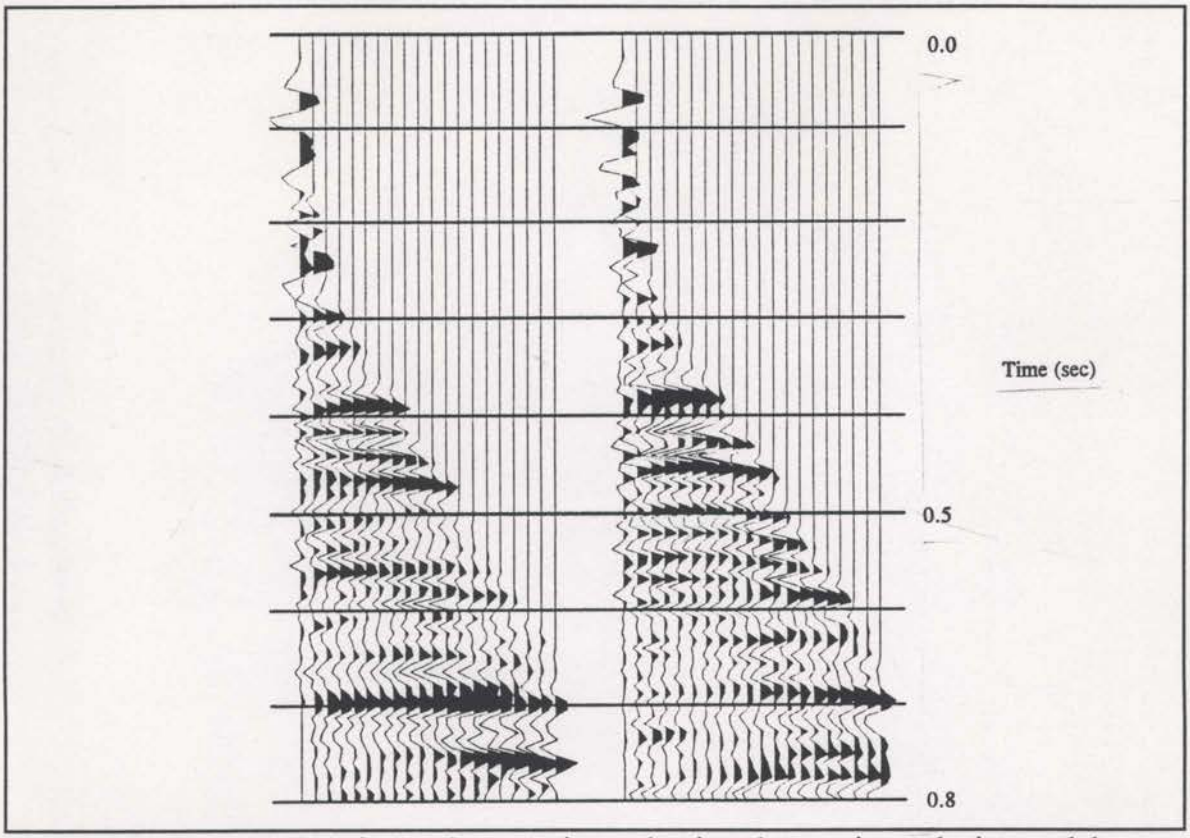


Fig. 5.5-5 Common location gathers - migrated using the starting velocity model (Figure 5.5-1).

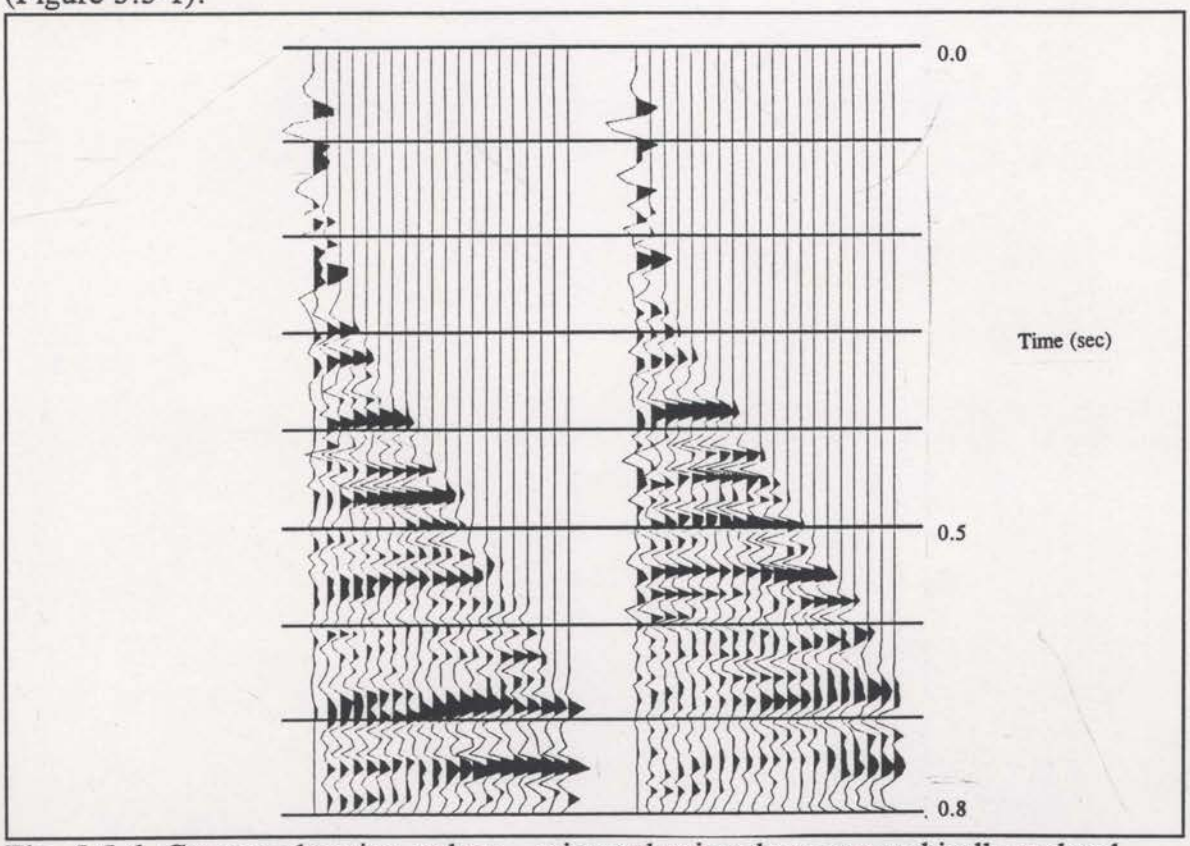
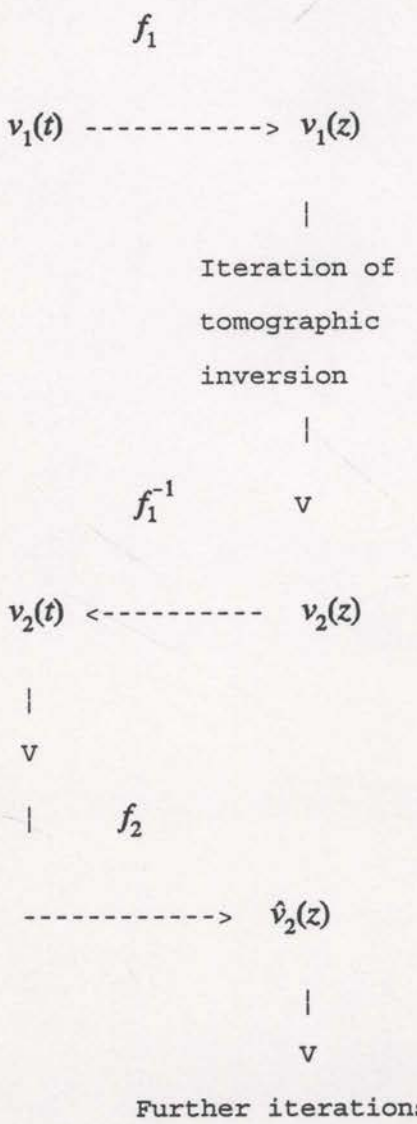


Fig. 5.5-6 Common location gathers - migrated using the tomographically updated velocities (Figure 5.5-3). Generally less residual moveout here.

the interval velocities is wrong, then the implied depths will be wrong. Such potential problems could be avoided if the tomographic inversion updated a velocity field relative to time instead of depth.

Velocities specified relative to depth are mandatory for the raytracing component of the inversion process. However, it is possible to use a velocity field specified in time coordinates if it is temporarily mapped to depth for each iteration. Consider the following mapping strategy;



where f_1 is the first mapping from time to depth, and f_1^{-1} is the inverse mapping (from

depth to time). The starting velocity model is specified in time. This is mapped to depth using the initial interval velocities and passed to the first iteration of the inversion. The interval velocities are changed in this iteration but the implied layer boundaries tend to remain at the same depths (see section 5.2). This implies that the times to the layer boundaries are changing even though these times are known quite accurately. An approximate way to change the depths instead of the times is to map the depth model back to time using the inverse of the previous forward mapping (ie. putting the layer boundaries back at the same times but with new velocities) and the creating a new forward mapping with the updated velocities. Such mapping, if performed between each iteration, will keep the times to the layer boundaries approximately constant and allow the implied depths to vary.

Such a time based inversion was run successfully on the data for Line B. The final velocity model and migrated data are displayed in Figures 5.5-7 and 5.5-8. Even though the velocity model (Figure 5.5-7) has some differences to that of Figure 5.5-3, the pre-stack depth migrated data is virtually unaffected. If the starting model had a larger velocity error, this technique may have greater effect.

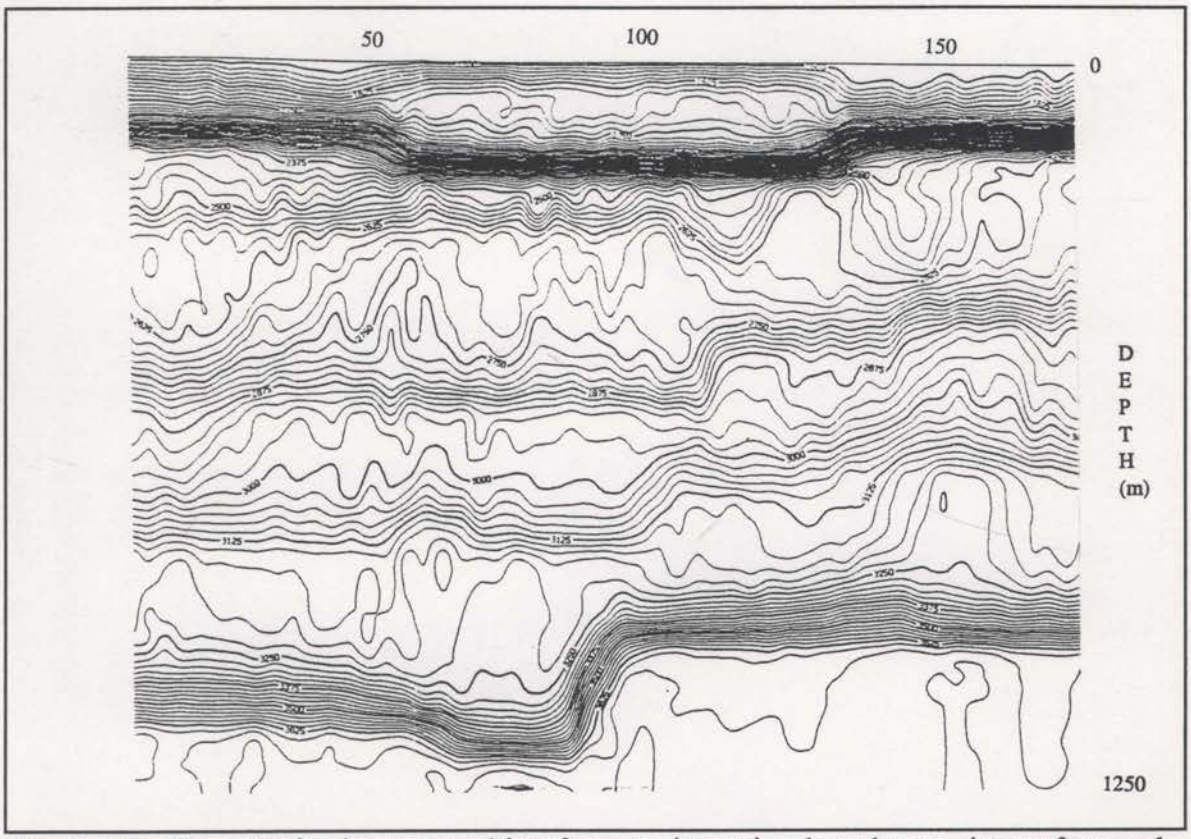


Fig. 5.5-7 The velocity image resulting from an inversion based on a time referenced velocity model.

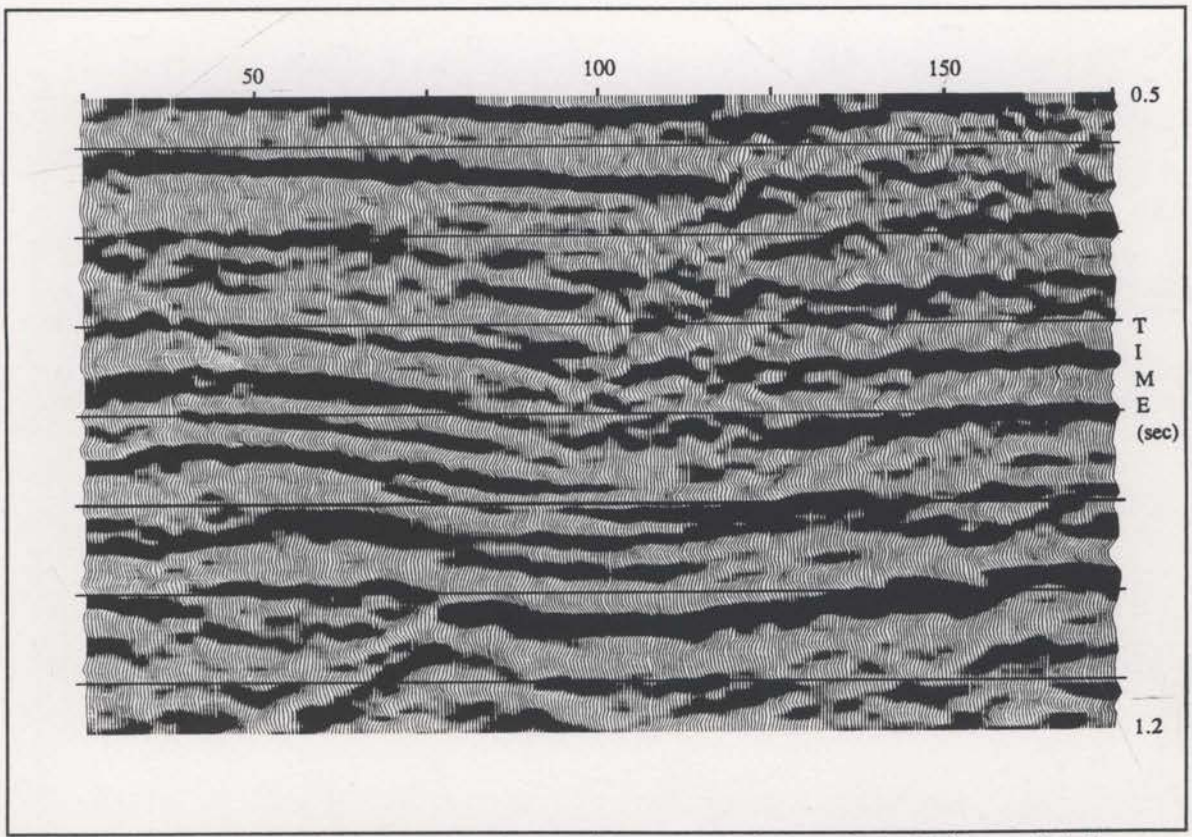


Fig. 5.5-8 The migrated image obtained with the velocity model of Figure 5.5-7.

§6 DISCUSSION

The modified reflection tomography algorithm that has been constructed in the preceding sections, not only simplifies the traveltime interpretation but at the same time includes classes of reflected data that are either difficult or impossible to include in the conventional tomographic inversion. Any event that has reflected or scattered from a single depth horizon can be picked and included in the inversion.

Diffraction events are a very important class of data that can be used in the current inversion algorithm. The conventional algorithm could not utilise such data since it is difficult to get two-point raytracing to trace a point reflector. These data will be very important in structurally complex areas. It is possible for diffracted arrivals to dominate in such areas. This feature also helps to alleviate the need for picking from approximately migrated traces (eg. Jacobs et al., 1992).

Fault plane reflections are another class of energy that is easily included in the tomographic inversion. These data can be very important because fault planes are often very steep and can provide some very high angle raypaths, enriching the angular coverage of the data. Extra angular coverage will improve the resolution of the reconstructed image and reduce the underdetermined nature of the inverse problem.

When the structure of the reflecting surfaces is significant, it is possible to get more than one reflection from the same surface to be recorded at any given receiver. These reflections are commonly called "multiple arrivals". Conventional reflection tomography can only use one of these arrivals - generally the first. Lailly et al., 1992, suggests that the first of the multiple arrivals is often the weakest and may not provide the

most reliable data. The tomographic inversion algorithm developed in this thesis naturally allows all of the multiple arrivals. Each arrival is distinguished by unique surface incidence angles (see section 2.2) and are all treated equally.

In this thesis it was assumed that the automatic traveltime picks (see section 2.3) will inevitably have some mispicks. Section 2.4 developed a data weighting scheme to help reduce the influence of these mispicks. However, these weights were based purely on the measured value of $X_{err(i)}$; this is not necessarily a reliable indicator of bad data. Some useful information will generally be suppressed as well. The following section looks at the possibility of using the entropy principle to assist in assigning more useful weights. This procedure has not been attempted so far in practice.

6.1 Further applications of entropy

In section 2.6 an arbitrary weighting scheme was developed in an attempt to limit the effect of erroneous data values on the inversion. The weights were based on the size of the individual X_{err} values (see equation (2.6-4)). Large X_{err} values may well be the result of valid velocity anomalies that need to be imaged. Even though these weights do assist the inverse procedure, a alternative weighting scheme may give better performance.

The discussion of section 4 indicates that the entropy principle can be used in problems of inference, and where more than one set of choices satisfies given constraints. It is quite possible that the entropy principle can play a role in assigning a better weighting scheme.

The critical information used by the inversion process is the set of partial derivatives of the constraint statistic with respect to each discrete velocity value, $\frac{\partial C}{\partial v_j}$ (see section 3.4). This value is the sum of the $\frac{\partial X_{err(i)}}{\partial v_j}$ values for each ray influenced by this velocity, weighted by $2X_{err(i)}$ (see equation (2.4-2)). It is suspected that a ray resulting from an erroneous data value will consistently have $\frac{\partial X_{err(i)}}{\partial v_j}$ values that differ markedly from the mean value for that velocity node. Define the following function,

$$F_i = \sum_{j-1}^{N} \left(\frac{\partial \bar{X}_{err}}{\partial v_j} - \frac{\partial X_{err(i)}}{\partial v_j} \right)^2 \qquad (6.1\text{-}1)$$

where there are N discrete velocities values in the model and $\frac{\partial \bar{X}_{err}}{\partial v_j}$ represents the average derivative for each velocity node. It is now possible to define weights, ω_i, based on these F_i values rather than the $X_{err(i)}$ values. Further, define the weighted F_i total,

$$\hat{F} = \sum_i \omega_i F_i \qquad (6.1\text{-}2)$$

Following the arguments of section 4, the weights, ω_i, can be specified with the aid of the entropy principle. With an entropy function defined as,

$$S_\omega = \sum_i (\omega_i - \omega_i \ln \omega_i), \qquad (6.1\text{-}3)$$

it is possible to find weights, ω_i, that decrease $\hat{F}$ by some predefined amount and maximise S_ω. These weights would discriminate against rays with Frechet derivatives that consistently deviated from the average derivative of the other rays. It is possible that such weights could damp the erroneous rays better than those based on the magnitude of $X_{err(i)}$.

6.2 3D Applications

The reflection tomography algorithm described and tested in this thesis assumes that the seismic data are collected along a straight line, and that all reflected raypaths between the sources and receivers exist within the vertical plane passing through this line. The algorithm comprehends energy motion in two dimensions (2D) only. When the geological structures are complex, it becomes more difficult to approximate the 3D world with a 2D model. Three-dimensional seismic volumes are increasingly being collected because of this problem. It is important that the reflection tomography technique is capable of accurately imaging 3D problems.

Raytracing without reflector depth parameterisation (see section 2.2) in three dimensions requires surface intercept azimuths as well as angles, for both the source and receiver ends of the ray. With 3D seismic surveys collected on land, the intercept angle and azimuth can be determined directly if three-component receivers are used. Offshore, such three-component receivers are not practical. Conventional marine 3D seismic surveys are recorded as a closely spaced collection of parallel linear traverses (2D seismic lines). The direction of these parallel lines is called the 'inline' direction. It is sufficient to define inline and crossline components of the surface intercept angles, or inline angles and the ray azimuths, for the 3D raytracing (see Figure 6.2-1). The crossline components of the surface intercept angles are difficult to estimate. The inline components of the intercept angles can be estimated from common source and common

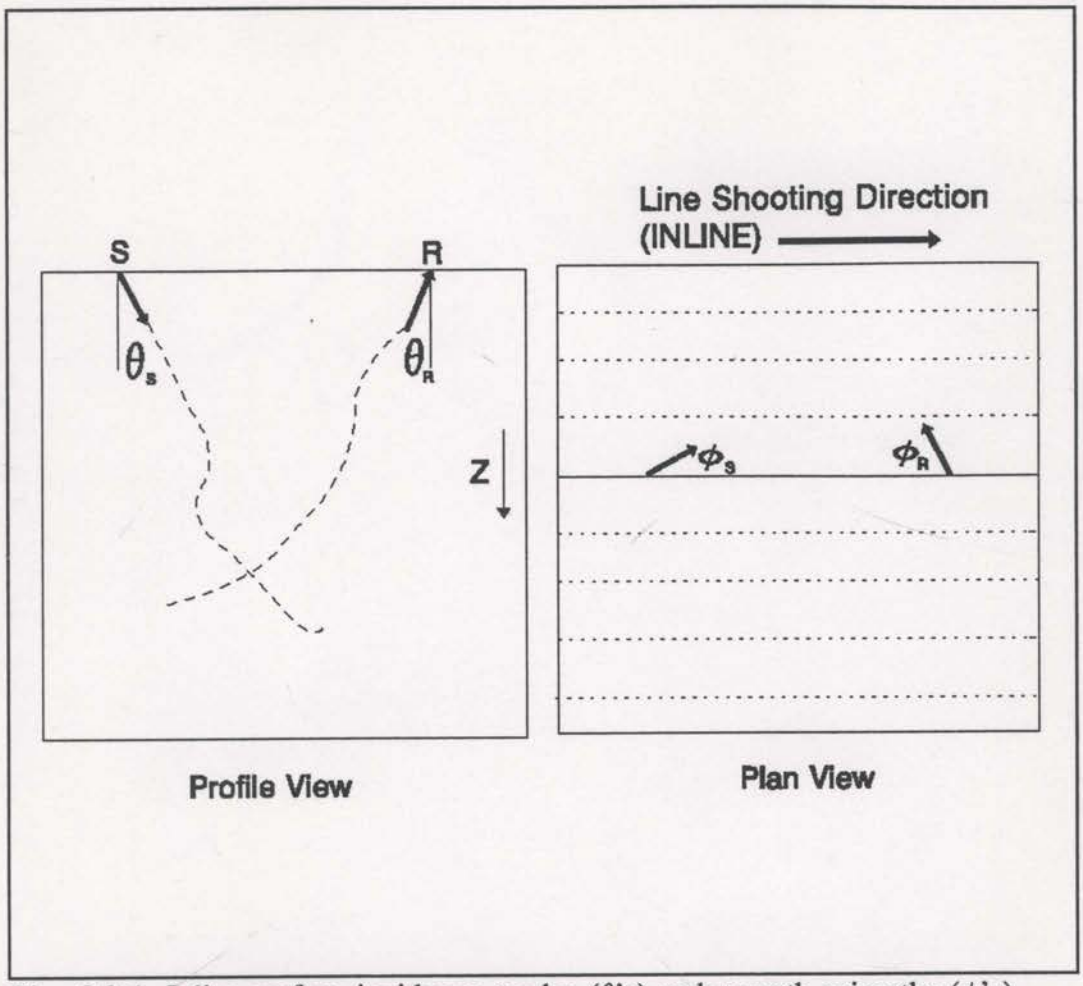


Fig. 6.2-1 Inline surface incidence angles (θ's) and raypath azimuths (ϕ's) suffice to begin 3D raytracing without pre-defined reflectors.

receiver gathers in the same way as for 2D data[1] (see section 2.2). The crossline component may be estimated by gathering data from adjacent lines. If traces recorded by the same receiver on the cable are gathered and examined, across the lines, the crossline component of the intercept angle could be estimated. This approach is hindered by two problems; the crossline data spacing is usually much larger than the inline data spacing (often 50 metres versus 12.5 metres), and gathering crossline data traces from a 3D data volume can be very computer intensive.

It may be possible to perform successful 3D reflection tomographic inversions

[1] Cable feathering would cause some complications.

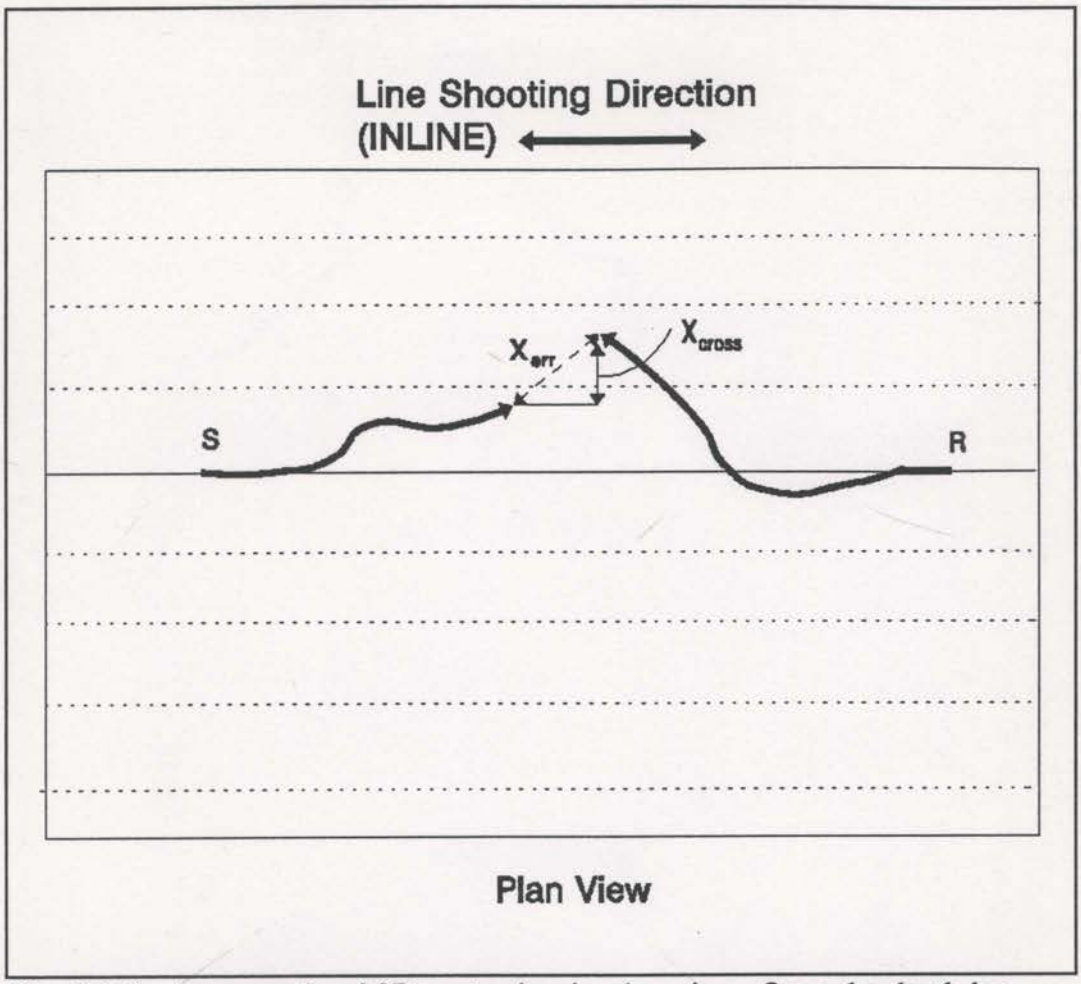


Fig. 6.2-2 An example of 3D raytracing in plan view. Once the depth has been reached where the total traveltime equals the measured time, X_{cross} is the crossline component of X_{err}.

without having to estimate the crossline intercept angles from the data. If the ray azimuths (see Figure 6.2-1) are initially assumed to be zero, then raytracing would result in something like that shown in Figure 6.2-2. This figure is a plan view showing that when the depth is reached where the traced traveltime equals the measured traveltime, there will generally be a crossline component, X_{cross}, of the X_{err} value. The source and receiver azimuths that were assumed to be zero can now be adjusted to remove the X_{cross} component of the error. Such an azimuth adjustment can be made during every iteration. As long as the 3D starting velocity model is sufficiently close to

the actual velocity field, it is assumed that this iterative angle adjustment will converge to the true azimuth values.

An increasingly popular method of offshore seismic data collection consists of placing the receiver cable on the sea floor (known as 'ocean bottom cable' surveys). A major advantage of this method is that both a hydrophone and an onshore style velocity phone can be placed in the ocean bottom cable, allowing incidence angles and azimuths to be determined directly. This data collection scheme also detaches the sources from the receiver lines so that full angular coverage (although surface restricted) can be recorded. With ocean bottom cables, a 3D survey can be recorded as outlined in Figure 6.2-3. Such data collection should present better behaved tomographic inversions.

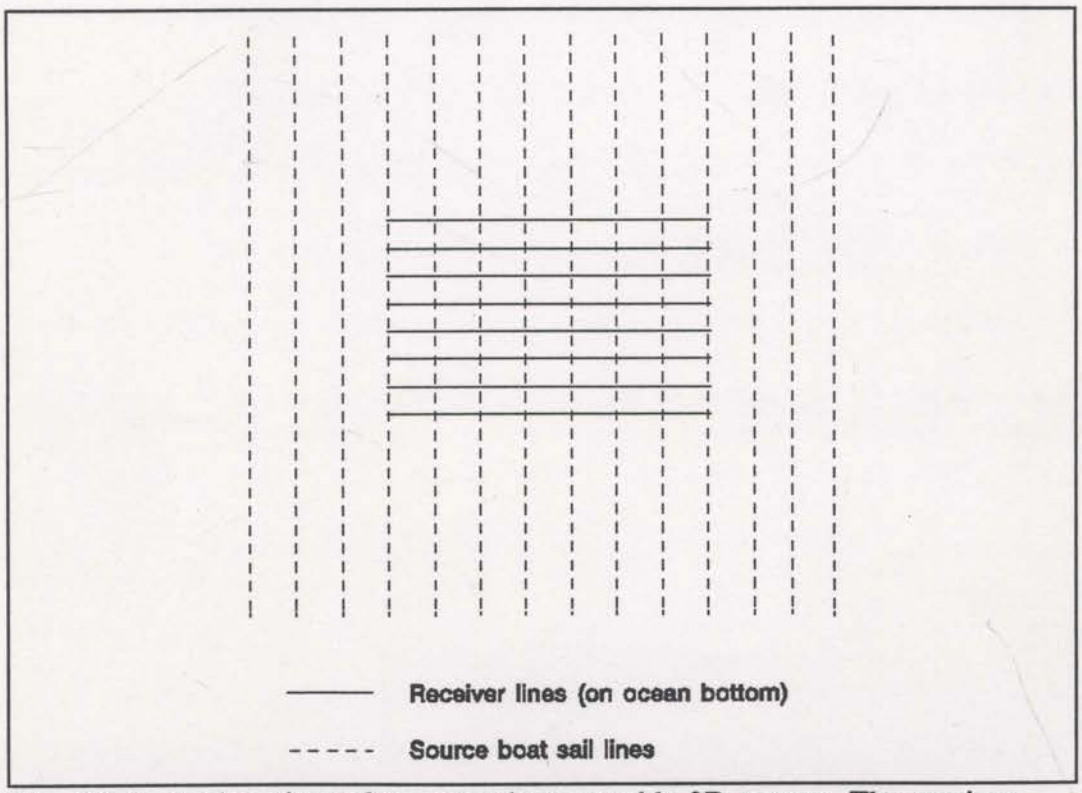


Fig. 6.2-3 A plan view of an ocean bottom cable 3D survey. The receiver cables are laid on the sea floor (solid lines) and the source boat is sailed (dashed lines) perpendicular to the receiver direction.

§7 CONCLUSION

Many difficulties associated with conventional reflection tomography can be overcome. Raytracing without pre-defined reflector depths makes the traveltime interpretation simple rather than laborious - even automatic traveltime interpretation algorithms work satisfactorily. Maximising the entropy of the image plays a central role in stabilising the inversion of these traveltime data by ensuring that unjustifiable features are not reconstructed at any iteration. This is achieved by enforcing minimum commitment to the velocity of any given cell. Maximising entropy also helps to avoid ambiguity problems. Stages of decreasing smoothing applied to the constraint statistic of the inversion process is also useful for avoiding local minima.

An additional benefit offered by unrestricted traveltime data interpretation is that diffraction events, fault plane reflections and multiple arrivals can be picked and added into the inversion quite naturally. Any primary reflected energy can be used. Therefore, not only does the new raytracing approach allow many more data to be picked, it also allows for wider and fuller angular coverage. Synthetic data examples have indicated that wider angular coverage is more beneficial than simply having more traveltime data.

Synthetic data examples have demonstrated that stages of decreasing ~~constraint statistic~~ smoothing can be extremely powerful. It can enable large anomalies (greater than 10% change from the starting model) to be imaged which went unnoticed by conventional inversions. When noise is added to the data used in the inversions, the ability to extract velocity anomalies is reduced. Tests with synthetic data showed that 10% surface angle noise is tolerable whereas 20% noise is not.

Using the entropy principle is of significant assistance to tomographic inversion. This principle prohibits velocity features from appearing that are not needed to satisfy the data. It should not be seen as a smoothness constraint - sharp changes can be imaged if required by the data. Maximising the entropy of the image during each iteration has a stabilising effect on the inversion as spurious structure is prevented from materialising. Also, the lack of unwarranted structure in the final solution considerably assists interpretation.

The entropy principle has a long history and has been used in many disciplines. It can be difficult to justify its use in tomographic image reconstruction. However, contingency table analysis and some very simple tomography problems lead to the required justification. Other popular inversion techniques use smoothness or minimum change constraints, and can lead to unwarranted, spurious structure in the solution image. The popular SIRT algorithm uses a minimum change (or minimum norm) constraint and synthetic examples have shown that more spurious structure exists on SIRT reconstructions compared to maximum entropy ones. Maximum entropy images are clearer and more interpretable.

Raytracing without fixed reflectors means that Fermat's principle cannot be invoked for Frechet derivative estimation. Initially it appeared that full re-raytracing would be needed to obtain the Frechet derivatives, but a 'semi-analytic' derivative estimation scheme proved to be accurate and efficient. The scheme requires raytracing through only a couple of discrete cells for each ray.

Raytracing without reflectors requires the apparent dips of the data on both common source and common receiver gathers. An algorithm based on the complex attributes of the traces has proven to be useful for this purpose, as well as being

computationally efficient.

The successes achieved with synthetic data sets are not as easily obtained with real data sets. The inherent noise problem with real data causes difficulties. Initially, the data must be adequately processed to attenuate multiple reflections as well as both coherent and random noise. Ideally, the data input to the automatic picking algorithm should consist of primary reflections only. Even so, the unavoidable noise in the data means that the starting velocity model for a tomographic inversion needs to be reasonably accurate. Fortunately, conventional interval velocity analysis methods can lead to a reasonably accurate starting model.

Two real data sets were considered and in both cases the starting velocity models were constructed from iterations of pre-stack depth migration. Pre-stack depth migration is also useful for evaluating the results of the tomographic inversion. After picking the data automatically and conducting the inversion with all the updates described above, the updated velocity model produced significant pre-stack depth migration improvements. Reflection tomography has a clear role in updating and improving the best velocity models obtained through conventional means.

Reflection tomography can, in principle, be easily extended to three-dimensions. Raytracing without reflector parameterisation requires ray azimuth estimation which may prove to be difficult. With conventional marine 3D recording, the azimuth can be obtained by considering adjacent lines. However, it may be possible to perform a successful 3D inversion without azimuth estimation. The azimuths could be iteratively obtained during the normal inversion iterations as long as the 3D starting model is adequate. Land 3D survey and marine ocean bottom cable 3D surveys offer the most promise in terms of 3D angular coverage and easy azimuth estimation.

ACKNOWLEDGMENTS

Thanks are extended to Halliburton Geophysical Services whose support throughout the research effort was invaluable.

I am also grateful for the assistance of Dr. Peter Buchen of the University of Sydney. His tireless enthusiasm and accurate direction were crucial to the completion of this thesis. Steve Brown, also of the University of Sydney, assisted by supplying initial maximum entropy inversion software which saved many hours of programming time.

Most importantly I must thank my wife, Sharon, and children, Jessica and Brittany. Without their relentless understanding and patience this thesis would not have been possible.

REFERENCES

Ables, J.G., 1974, Maximum entropy spectral analysis: Astron. Astrophys. Suppl., **15**, 383-393.

Al-Yahya, K., 1989, Velocity analysis by iterative profile migration, Geophysics, **54**, 718-729.

Bassrei, A., 1990, Inversion of seismic data by a relative entropy algorithm, 60th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1111-1114.

Berkhout, A.J., 1987, Applied seismic wave theory: Elsevier Science Publishers.

Biondi, B., 1988, Interval velocity estimation from beam-stacked data, 58th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 680-683.

Biondi, B., 1990, Velocity estimation by beam-stack: field-data results, 60th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1271-1274.

Bishop, T.N., Bube, K.P., Cutler, R.T., Langan, R.T., Love, P.L., Resnick, J.R., Shuey, R.T., Spindler, D.A., and Wyld, H.W., 1985, Tomographic determination of velocity and depth in laterally varying media, Geophysics, **50**, 903-923.

Bording, R.P., Gersztenkorn, A., Lines, L.R., Scales, J.A., and Treitel, S., 1987, Applications of seismic travel-time tomography, Geophys. J. R. Astr. Soc., **90**, 285-303.

Bracewell. R.N., 1978, The Fourier transform and its applications: Second Edition, McGraw Hill.

Bregman, N.D., Chapman, C.H. and Bailey, R.C., 1989, Travel time and amplitude analysis in seismic tomography, J. Geophys. Res., **94**, 7577-7587.

Burg, J.P., 1967, Maximum entropy spectral analysis, 37th Ann. Internat. Mtg., Soc.

REFERENCES

Ables, J.G., 1974, Maximum entropy spectral analysis: Astron. Astrophys. Suppl., **15**, 383-393.

Al-Yahya, K., 1989, Velocity analysis by iterative profile migration, Geophysics, **54**, 718-729.

Bassrei, A., 1990, Inversion of seismic data by a relative entropy algorithm, 60th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1111-1114.

Berkhout, A.J., 1987, Applied seismic wave theory: Elsevier Science Publishers.

Biondi, B., 1988, Interval velocity estimation from beam-stacked data, 58th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 680-683.

Biondi, B., 1990, Velocity estimation by beam-stack: field-data results, 60th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1271-1274.

Bishop, T.N., Bube, K.P., Cutler, R.T., Langan, R.T., Love, P.L., Resnick, J.R., Shuey, R.T., Spindler, D.A., and Wyld, H.W., 1985, Tomographic determination of velocity and depth in laterally varying media, Geophysics, **50**, 903-923.

Bording, R.P., Gersztenkorn, A., Lines, L.R., Scales, J.A., and Treitel, S., 1987, Applications of seismic travel-time tomography, Geophys. J. R. Astr. Soc., **90**, 285-303.

Bracewell, R.N., 1978, The Fourier transform and its applications: Second Edition, McGraw Hill.

Burg, J.P., 1967, Maximum entropy spectral analysis, 37th Ann. Internat. Mtg., Soc.

Carrion, P., 1991, Dual tomography for complex imaging, Geophysics, **56**, 1395-1404.

compensation tomography, Geophysical Prospecting, **41**, 367-380.

Catlin, D.E., 1989, Estimation, control and the discrete kalman filter: Springer-Verlag.

Chiu, S.K.L. and Stewart, R.R., 1987, Tomographic determination of three-dimensional seismic velocity structure using well logs, vertical seismic profiles, and surface seismic data, Geophysics, **52**, 1085-1098.

Cornwell, T.J., 1983, Is Jaynes' maximum entropy principle applicable to image reconstruction?, *in* Roberts, J.A., Ed., Indirect Imaging: Cambridge Univ. Press, 291-296.

Delprat, F. and Lailly, P., 1990, What information is contained in reflection tomography data, 60th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1290-1293.

De Vries, D. and Berkhout, A.J., 1984, Velocity analysis based on minimum entropy, Geophysics, **49**, 2132-2142.

Dusaussoy, N.J. and Abdou, I.E., 1991, The extended MENT algorithm: a maximum entropy type algorithm using prior knowledge for computerised tomography, IEEE Trans. on Signal Proc., **39**, 1164-1180.

Etgen, J.T., 1988, Velocity analysis using prestack depth migration: linear theory, 58th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 909-912.

Farra, V. and Madariaga, R., 1988, Non-linear reflection tomography, Geophysical Journal, **95**, 135-147.

Finn, C.J., and Backus, M.M., 1991, Maximum-likelihood traveltime inversion for a

Expl. Geophys.

Carrion, P., Vesnaver, A., Boehm, G., and Pettenati, F., 1993, Aperture compensation tomography, Geophysical Prospecting, **41**, 367-380.

Catlin, D.E., 1989, Estimation, control and the discrete kalman filter: Springer-Verlag.

Chiu, S.K.L. and Stewart, R.R., 1987, Tomographic determination of three-dimensional seismic velocity structure using well logs, vertical seismic profiles, and surface seismic data, Geophysics, **52**, 1085-1098.

Cornwell, T.J., 1983, Is Jaynes' maximum entropy principle applicable to image reconstruction?, *in* Roberts, J.A., Ed., Indirect Imaging: Cambridge Univ. Press, 291-296.

Delprat, F. and Lailly, P., 1990, What information is contained in reflection tomography data, 60th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1290-1293.

De Vries, D. and Berkhout, A.J., 1984, Velocity analysis based on minimum entropy, Geophysics, **49**, 2132-2142.

Dusaussoy, N.J. and Abdou, I.E., 1991, The extended MENT algorithm: a maximum entropy type algorithm using prior knowledge for computerised tomography, IEEE Trans. on Signal Proc., **39**, 1164-1180.

Etgen, J.T., 1988, Velocity analysis using prestack depth migration: linear theory, 58th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 909-912.

Farra, V. and Madariaga, R., 1988, Non-linear reflection tomography, Geophysical Journal, **95**, 135-147.

Finn, C.J., and Backus, M.M., 1991, Maximum-likelihood traveltime inversion for a

3-D velocity-depth model, 61st Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 897-900.

Frieden, B.R., 1972, Restoring with maximum likelihood and maximum entropy: J. Opt. Soc. Am., **62**, 511-518.

Gill, P.E. and Murray, W., 1974, Numerical methods for constrained equalisation: Academic Press.

Good, I.J., 1963, Maximum entropy for hypothesis formulation, especially for multidimensional contingency tables: Annals Math. Stat., **34**, 911-934.

Gull, S.F., 1988, Bayesian inductive inference and maximum entropy, *in* Erickson, G.J., and Smith, C.R., Eds., Maximum-entropy and Bayesian methods in science and engineering (vol. 1): Kluwer Academic Publishers, 53-74.

Gull, S.F., and Daniell, G.J., 1978, Image reconstruction from incomplete and noisy data: Nature, **272**, 686-690.

Gull, S.F., and Skilling, J., 1983, The maximum entropy method, *in* Roberts, J.A., Ed., Indirect Imaging: Cambridge Univ. Press, 267-279.

Harlan, W.S., 1989, Tomographic estimation of seismic velocities from reflected raypaths, 59th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 922-924.

Harlan, W.S., Hanson, D.W., and Boyd, M., 1991a, Common-offset depth migrations and traveltime tomography, 61st Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 971-973.

Harlan, W.S., Hanson, D.W., and Boyd, M., 1991b, Traveltime tomography with multioffset common-reflection points, 61st Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 974-976.

Jackson, D.D., 1979, The use of a priori data to resolve non-uniqueness in linear inversion, Geophys. J. R. Astr. Soc, **57**, 137-157.

Jacobs, J.A.C., Delprat-Jannaud, F., Ehinger, A., and Lailly, P., 1992, Sequential migration-aided relection tomography: a tool for imaging complex structures, 62nd Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1054-1057.

Jaynes, E.T., 1957, Information theory and statistical mechanics: Phys. Rev., **106**, 620-630.

Jaynes, E.T., 1982, On the rationale of maximum-entropy methods: Proc. IEEE, **70**, 939-952.

Jaynes E.T., 1984, Prior information and ambiguity in inverse problems: SIAM-AMS Proc., **14**, 151-166.

Jaynes, E.T., 1985, Where do we go from here?, *in* Smith, C.R., and Grandy, W.T., Eds., Maximum-entropy and Bayesian methods in inverse problems: D. Reidel Publishing Company, 21-58.

Jaynes, E.T., 1988a, The evolution of Carnot's principle, *in* Erickson, G.J., and Smith, C.R., Eds., Maximum-entropy and Bayesian methods in science and engineering (vol. 1): Kluwer Academic Publishers, 267-281.

Jaynes, E.T., 1988b, How does the brain do plausible reasoning?, *in* Erickson, G.J., and Smith, C.R., Eds., Maximum-entropy and Bayesian methods in science and engineering (vol. 1): Kluwer Academic Publishers, 1-24.

Jupp, D.L.B. and Vozoff, K., 1975, Stable iterative methods for the inversion of geophysical data, Geophys. J. R. Astr. Soc., **42**, 957-976.

Justice, J.H., Vassiliou, A.A. Singh, S., Logel, J.D., Hansen, P.A., Hall, B.R., Hutt,

P.R. and Solanki, J.J., 1989, Acoustic tomography for monitoring enhanced oil recovery, The Leading Edge, **8**, no. 2, 12-19.

Kennett, B.L.N., 1988, Seismic velocity field estimation - strategies for large-scale nonlinear inverse problems, Exploration Geophysics, **19**, 297-298.

Kennett, B.L.N. and Harding, A.J., 1985, Is ray theory adequate for reflection seismic modelling? (a survey of modelling methods), First Break, **3**, no. 1, 9-14.

Kennett, B.L.N., Sambridge, M.S., and Williamson, P.R., 1988, Subspace methods for large inverse problems with multiple parameter classes, Geophys. Journal, **94**, 237-247.

Kennett, B.L.N. and Williamson, P.R., 1988, Subspace methods for large-scale nonlinear inversion, *in* Mathematical Geophysics, N.J. Vlaar et al., Eds., D.Reidel Publishing Company, 139-154.

Khinchin, A.I., 1957, Mathematical foundations of information theory: Dover Publications, Inc.

LaBrecque, D.J., 1990, Comparison of maximum entropy and smoothest inversion for resistivity tomography, 60th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 546-549.

Lailly, P., Delprat-Jannaud, F., Becache, E., and Chovet, E., 1992, Reflection tomography: how to cope with multiple arrivals, 62nd Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 741-744.

Lailly, P., Svay, J.C, and Versteeg, R.J., 1990, Some remarks on diffent optimisation criteria for the velocity model, 60th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1283-1286.

Landa, E., Beydoun, W. and Tarantola, A., 1989, Reference velocity estimation from

prestack waveforms: coherency optimisation by simulated annealing, Geophysics, **54**, 984-990.

Langan, R.T., Lerche, I., and Cutler, R.T., 1985, Tracing of rays through heterogeneous media: An accurate and efficient procedure, Geophysics, **50**, 1456-1465.

Leger, M., Nguyen, L.L., Carlini, A., and Cornini, S., 1989, Reflection traveltime inversion: a gas detection method, 59th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 854-857.

Lines, L., 1991, Applications of tomography to borehole and reflection seismology, The Leading Edge, **10**, no. 7, 11-17.

Lynn, W.S. and Claerbout, J.F., 1982, Velocity estimation in laterally varying media, Geophysics, **47**, 884-897.

Marquardt, D.W., 1963, An algorithm for least-squares estimation of non-linear parameters, J. Soc. Indust. Appl. Math., **11**, 431-441.

McGillivray, P.R. and Oldenburg, D.W., 1990, Methods for calculating Frechet derivatives and sensitivities for the non-linear inverse problem: a comparsion study, Geophysical Prospecting, **38**, 499-524.

Menke, W., 1984, Geophysical data analysis: discrete inverse theory: Academic Press.

Michelena, R.J. and Harris, J.M., 1991, Tomographic traveltime inversion using natural pixels, Geophysics, **56**, 635-644.

Moser, T.J., 1991, Shortest path calculation of seismic rays, Geophysics, **56**, 59-67.

Oldenburg, D.W. and Ellis, R.G., 1991, Inversion of geophysical data using an approximate inverse mapping, Geophys. J. Int., **105**, 325-353.

Phillips, W.S. and Fehler, M.C., 1991, Traveltime tomography: a comparison of

popular methods, Geophysics, **56**, 1639-1649.

Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., 1989, Numerical recipes: Cambridge Univ. Press.

Reitsch, E., 1977, The maximum entropy approach to inverse problems, J. Geophys., **42**, 489-506.

Reitsch, E., 1985, On an alleged breakdown of the maximum-entropy principle, *in* Smith, C.R., and Grandy, W.T., Eds., Maximum-entropy and Bayesian methods in inverse problems: D. Reidel Publ. Co., 67-82.

Scheuer, T.E., and Oldenburg, D.W., 1988, Local phase velocity from complex seismic data, Geophysics, **53**, 1503-1511.

Shaw, P.R. and Orcutt, J.A., 1985, Waveform inversion of seismic refraction data and applications to young Pacific crust, Geophys. J. R. Astr. Soc., **82**, 375-414.

Shannon, C.E., 1948, The mathematical theory of communication, Bell Syst. Tech. J., **27**, 379-423, 623-656.

Sherwood, J.W.C., 1989, Depth sections and interval velocities from surface seismic data, The Leading Edge, **8**, no. 9, 44-49.

Shore, J.E., 1984, Inversion as logical inference - theory and applications of maximum entropy and minimum cross-entropy, SIAM-AMS Proc., **14**, 139-149.

Singh, R.P. and Singh, Y.P., 1991, RAYPT - a new inversion technique for geotomographic data, Geophysics, **56**, 1215-1227.

Skilling, J., 1988, The axioms of maximum entropy, *in* Erickson, G.J., and Smith, C.R., Eds., Maximum-entropy and Bayesian methods in science and engineering (vol. 1): Kluwer Academic Publishers, 173-187.

Skilling, J., and Bryan, R.K., 1984, Maximum entropy image reconstruction: general

algorithm: Mon. Not. R. Astr. Soc., **211**, 111-124.

Skilling, J., and Gull, S.F., 1984, The entropy of an image: SIAM-AMS Proc. (Inverse Problems), **14**, 167-189.

Skilling, J., and Gull, S.F., 1985, Algorithms and applications, *in* Smith, C.R., and Grandy, W.T., Eds., Maximum-entropy and Bayesian methods in inverse problems: D. Reidel Publ. Co., 83-132.

Smith, C.R., Inguva, R. and Morgan, R.L., 1984, Maximum-entropy inverses in physics, SIAM-AMS Proc., **14**, 127-137.

Stewart, R.R., 1989, Exploration seismic tomography: fundamentals: SEG Continuing Education Course Notes.

Stork, C., 1992, Reflection tomography in the postmigrated domain, Geophysics, **57**, 680-692.

Stork, C. and Clayton, R.W., 1991, Linear aspects of tomographic velocity analysis, Geophysics, **56**, 483-495.

Sword, C., 1986, Tomographic determination of interval velocities from picked seismic data - theory and synthetic results: Stanford Exploration Project Report, **48**, 1-33.

Sword, C., 1987, Tomographic determination of interval velocities from reflection seismic data: the method of controlled directional reception, Ph.D. Dissertation, Stanford University.

Taner, M.T., and Koehler, F., 1969, Velocity Spectra - digital computer derivation and applications of velocity functions, Geophysics, **34**, 859-881.

Tarantola, A., 1987, Inverse Problem Theory: Elsevier Science Publishers.

Telford, W.M., Geldart, L.P., Sheriff, R.E., and Keys, D.A., 1976, Applied

Geophysics, Cambridge University Press.

Tikhonov, A.N. and Arsenin, V.Y., 1977, Solutions of ill-posed problems: John Wiley and Sons.

Toldi, J.L., 1989, Velocity analysis without picking, Geophysics, **54**, 191-199.

Ulrych, T.J., Leaney, W.S., Jensen, O.G. and Bassrei, A., 1990, Inference, inversion, Bayes, and entropy, 60th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1115-1118.

van der Made, P.M., and van Riel, P., 1988, Estimating complex velocity models for depth migration, 58th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 886-889.

van Trier, J., 1988, Migration-velocity analysis using geological constraints, 58th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 897-900.

van Trier, J., 1990, Reflection tomography after depth migration: field data results, 60th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1279-1282.

Vidale, J.E., 1989, Finite-difference calculation of traveltimes in 3-D, 59th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 1096-1098.

White, R.E., 1991, Properties of instantaneous seismic attributes, The Leading Edge, **10**, no. 7, 26-32.

Whiting, P.M., 1991a, Maximum entropy reflection tomography, Exploration Geophysics, **23**, 459-464.

Whiting, P.M., 1991b, Practical reflection tomography and maximum entropy imaging, 61st Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 986-989.

Whiting, P.M., 1993, Entropy and tomographic image reconstruction, Submitted to Geophysics.

Whiting, P.M., 1993, Practical reflection tomography for velocity field estimation, Submitted to Geophysics.

Williamson, P.R., 1990, Tomographic inversion in reflection seismology, Geophys. J. Int., **100**, 255-274.

Zhou, B., Sinadinovski, C., and Greenhalgh, S.A., 1992, Non-linear inversion travel-time tomography: imaging high-contrast inhomogeneities, Exploration Geophysics, **23**, 459-464.

APPENDIX

The FORTRAN subroutines listed in this appendix are described in the earlier text. The subroutines are listed in alphabetical order with the subroutine name posted at the top right of every page. Note that MEMNL$8 is really a collection of a number of subroutines acting as a single routine.

```
@PROCESS DC(GAUS),OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS DIRECTIVE('*VD:')
c-------------------------------------------------------------------------------
c----- subroutine to compute the frechet derivatives for submission as    -----
c----- the response matrix to MEM_NL                                      -----
c----- this subroutine uses finite difference perturbations               -----
c----- uses Chuck Sword's XERR statistic                                  -----
c----- this routine is called for each ray involved in the data times     -----
c-------------------------------------------------------------------------------
      subroutine frechx(index,sloc1,tang1,sloc2,tang2,datatime,xerr,R)
c-------------------------------------------------------------------------------
c----- index  =  index number of ray as stored in geom file
c----- sloc1  =  surface location of ray1
c----- tang1  =  take off angle of ray1
c----- sloc2  =  surface location of ray2
c----- tang2  =  take off angle of ray2
c----- datatime=  actual measured time of ray
c----- xerr   =  Chuck Sword's statistic
c-----
c----- n    = number of rows in velocity grid
c----- m    = number of columns in velocity grid
c----- nint = spatial increment between grid rows
c----- mint = spatial increment between grid columns
c-----
c----- L     = gaussian half-width (in metres)
c----- lc    = constant multiplier of the model covariance matrix (use 1.0)
c-----
c----- bfx   = (defunct variable - submit 1)
c----- bfz   = (defunct variable - submit 1)
c-----
c----- nmax  = maximum number of velocity nodes
c----- mmax  = maximum number of data rays
c----- max   = maximum of nmax and mmax
c-----
c----- -->common block /velocity/
c----- vel(n,m)= values of velocity at grid points
c-----
c----- -->common block /travel/
c----- ray1travel(n*m,6) = array that stores cell,theta0,x,z,t for ray1's path
c----- num               = number of points in ray1travel
c----- ray2travel(n*m,6) = array that stores cell,theta0,x,z,t for ray2's path
c----- num2              = number of points in ray2travel
c-----
c----- i,j,k= temporary integer variables
c----- t1,t2,t3,t4,t5= temporary double-precision variables
c-----
c----- tri   = coordinate indices of the apex of the current triangle
c-----
c----- row   = the row of the current cell
c----- col   = the column of the current cell
c----- trinode1,2,3 = velocity node indices of apices of current triangle
c----- rec1  = integer variable for record numbers
c-----
c----- R(ndat,n*m) = response matrix (frechet derivatives)
c-----
c-------------------------------------------------------------------------
      real*4 nint,mint
      real*4 ray1travel,ray2travel
      real*4 sloc1,tang1,sloc2,tang2
      real*4 vel
      real*4 datatime,xerr,newxerr
      real*4 C,ggC
      real*4 lc
      real*4 R
      real*8 t1,t2,t3,t4,t5
      logical triflag
      logical affected
      integer*4 i,j,k
      integer*4 n,m
      integer*4 bfx,bfz
      integer*4 index
```

```
      integer*4 L
      integer*4 nmax,mmax,max
      integer*4 num,num2
      integer*4 row,col
      integer*4 tri
      integer*4 trinode1,trinode2,trinode3
      integer*4 rec1
      integer*4 p,mult,rem,block

      parameter(n=41,m=41,nint=50.0e0,mint=50.0e0)
      parameter(bfx=1,bfz=1)
      parameter(nmax=1681,mmax=184,max=1681)
      parameter(ndat=184)

      dimension vel(n,m)
      dimension ray1travel(n*m,6),ray2travel(n*m,6)
      dimension C(nmax,nmax),ggC(nmax,max)
      dimension R(ndat,n*m)
      dimension affected((n/bfz)*(m/bfx))
      dimension tri(3,2)
      common /velocity/vel
      common /travel/ray1travel,num,ray2travel,num2
c
c----- set constants and initial variables
c
      t1=0.0d0
      t2=0.0d0
      t3=0.0d0
      t4=0.0d0
      t5=0.0d0
      do 1 i=1,(n/bfz)*(m/bfx)
         affected(i)=.false.
1        continue
c
c----- compute all velocity nodes affected by the rays
c----- begin working along path of ray1
c
      do 1000 i=1,num
c
c----- first compute the 'tri' nodes affecting this cell
c
         p=int(ray1travel(i,1))
         rem=mod(p,(2*(m-1)))
         mult=(p-rem)/(2*(m-1))
         row=mult+1
         if(rem.eq.0) row=row-1
         if(rem.gt.0) then
            col=int((rem-1)/2.0)+1
         else
            col=m-1
            endif
         tri(1,1)=col
         tri(1,2)=row
         tri(2,1)=col+1
         tri(2,2)=row+1
         if(mod(ray1travel(i,1),2.0e0).eq.0.0e0) then
            tri(3,1)=col+1
            tri(3,2)=row
         else
            tri(3,1)=col
            tri(3,2)=row+1
            endif
c
c----- compute the corresponding velocity node indices
c
         trinode1=(tri(1,2)-1)*m+tri(1,1)
         trinode2=(tri(2,2)-1)*m+tri(2,1)
         trinode3=(tri(3,2)-1)*m+tri(3,1)
c
c----- compute the affected blocks
c
```

```
            do 500 j=1,3
               rem=mod(tri(j,1),bfx)
               mult=(tri(j,1)-rem)/bfx
               if(rem.eq.0) then
                  col=mult
               else
                  col=mult+1
                  endif
               rem=mod(tri(j,2),bfz)
               mult=(tri(j,2)-rem)/bfz
               if(rem.eq.0) then
                  row=mult
               else
                  row=mult+1
                  endif
               k=(row-1)*m/bfx+col
               if(k.gt.n*m) k=n*m
               if(k.lt.1) k=1
               if(j.eq.1) trinode1=k
               if(j.eq.2) trinode2=k
               if(j.eq.3) trinode3=k
500            continue
c
c----- set the required 'affected' flags
c
            affected(trinode1)=.true.
            affected(trinode2)=.true.
            affected(trinode3)=.true.
1000        continue
c
c----- continue computing the velocity nodes affected by the rays
c----- now do the same for ray2
c
         do 2000 i=1,num2
c
c----- first compute the 'tri' nodes affecting this cell
c
            p=int(ray2travel(i,1))
            rem=mod(p,(2*(m-1)))
            mult=(p-rem)/(2*(m-1))
            row=mult+1
            if(rem.eq.0) row=row-1
            if(rem.gt.0) then
               col=int((rem-1)/2.0)+1
            else
               col=m-1
               endif
            tri(1,1)=col
            tri(1,2)=row
            tri(2,1)=col+1
            tri(2,2)=row+1
            if(mod(ray2travel(i,1),2.0e0).eq.0.0e0) then
               tri(3,1)=col+1
               tri(3,2)=row
            else
               tri(3,1)=col
               tri(3,2)=row+1
               endif
c
c----- compute the corresponding velocity node indices
c
            trinode1=(tri(1,2)-1)*m+tri(1,1)
            trinode2=(tri(2,2)-1)*m+tri(2,1)
            trinode3=(tri(3,2)-1)*m+tri(3,1)
c
c----- compute the affected blocks
c
            do 1500 j=1,3
               rem=mod(tri(j,1),bfx)
               mult=(tri(j,1)-rem)/bfx
               if(rem.eq.0) then
```

```
               col=mult
            else
               col=mult+1
               endif
            rem=mod(tri(j,2),bfz)
            mult=(tri(j,2)-rem)/bfz
            if(rem.eq.0) then
               row=mult
            else
               row=mult+1
               endif
            k=(row-1)*m/bfx+col
            if(k.gt.n*m) k=n*m
            if(k.lt.1) k=1
            if(j.eq.1) trinode1=k
            if(j.eq.2) trinode2=k
            if(j.eq.3) trinode3=k
1500        continue
c
c----- set the required 'affected' flags
c
         affected(trinode1)=.true.
         affected(trinode2)=.true.
         affected(trinode3)=.true.
2000     continue
c
c----- now begin raytracing through the perturbed velocity fields
c
      do 3000 i=1,(n/bfz)*(m/bfx)
         if(.not.affected(i)) goto 3000
c
c----- compute the top left node of the block
c
         rem=mod(i,m/bfx)
         mult=(i-rem)/(m/bfx)
         if(rem.eq.0) then
            col=m/bfx
            row=mult
         else
            col=rem
            row=mult+1
            endif
         col=(col-1)*bfx+1
         row=(row-1)*bfz+1
c
c----- multiply affected velocities by 1.0025
c
            do 3500 j=col,col+bfx-1
               do 3600 k=row,row+bfz-1
                  vel(k,j)=vel(k,j)*1.0025e0
3600              continue
3500           continue
c
c----- re-raytrace the ray
c
33       newxerr=0.0e0
         call xerrrt(sloc1,tang1,sloc2,tang2,datatime,newxerr)
c
c----- divide affected velocities by 1.0025 to restore them to original values
c
            do 3700 j=col,col+bfx-1
               do 3800 k=row,row+bfz-1
                  vel(k,j)=vel(k,j)/1.0025e0
                  t1=t1+dble(vel(k,j))
3800              continue
3700           continue
            t1=(t1/dble(bfx*bfz))*.0025d0
c
c----- put ((newxerr-xerr)/t1) into response matrix
c
         R(index,i)=((newxerr-xerr)/sngl(t1))
```

```
c
c----- loop back to do rest of the velocity nodes
c
3000      continue

      return
      end
```

```
@PROCESS DC(GAUS),OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS DIRECTIVE('*VD:')
c-------------------------------------------------------------------------------
c----- subroutine to compute the model covariance matrix for gaussian     -----
c----- smoothing                                                          -----
c----- the half-width of the gaussian is 'L'                              -----
c----- the constant weighting of the covariance matrix is 'lc'            -----
c----- subroutine also computes first/second order derivative components  -----
c-------------------------------------------------------------------------------
      subroutine gauscalc(L)
c-------------------------------------------------------------------------------
c----- nq = n = number of rows in velocity grid
c----- mq = m = number of columns in velocity grid
c----- nint = spatial increment between grid rows
c----- mint = spatial increment between grid columns
c-----
c----- L     = gaussian half-width (in metres)
c----- lc    = constant multiplier of the model covariance matrix (use 1.0)
c-----
c----- nmaxq = maximum number of velocity nodes
c----- mmaxq = maximum number of data rays
c----- maxq  = maximum of nmaxq and mmaxq
c-----
c----- C(0:nq-1,0:mq-1)=model covariance table defining the gaussian smoothing
c-----                  This is the output of this subroutine.
c-----
c----- i,j,k= temporary integer variables
c----- t1,t2,t3,t4,t5= temporary double-precision variables
c-----
c-------------------------------------------------------------------------------
      real*4 nint,mint
      real*4 nint2,mint2
      real*4 sum
      real*4 C
      real*4 lc
      real*8 t1,t2,t3,t4,t5
      integer*4 i,j,k
      integer*4 i1,i2,i3,i4,zz
      integer*4 L
      integer*4 n,nq,m,mq,ndat,dne
      integer*4 rem,mult
      integer*4 icol,irow,jcol,jrow
      integer*4 nmax,nmaxq,mmax,mmaxq,max,maxq

      parameter(nq=90,mq=210)
      parameter(nmaxq=17000,mmaxq=4200,maxq=17000)

      dimension C(0:nq-1,0:mq-1)

      common /conparm/n,m,nint,mint,nmax,mmax,max,ndat,dne
      common /gaus/C
c
c----- set constants and initial variables
c
      lc=1.0
      t1=0.0d0
      t2=0.0d0
      t3=0.0d0
      t4=0.0d0
      t5=0.0d0
      sum=0.0
      nint2=nint**2
      mint2=mint**2
c
c----- compute the model covariance table, C
c----- first compute the constant factor that is to be scaled relative to its
c----- distance away
c
      if((2.0*float(L)).le.mint) then
         t5=0.0d0
      else
```

```
            t5=dble(exp(-0.5e0*(mint/float(L))**2))
            endif
c
c----- if t5 is zero, set the C matrix equal to the identity
c
      if(t5.eq.0.0d0) then
         do 90 i=0,n-1
            do 95 j=0,m-1
               C(i,j)=0.0
95             continue
90          continue
         C(0,0)=lc*1.0
         return
         endif
c
c----- now loop through for possible row and column differences
c
      do 100 i=0,n-1
         do 105 j=0,m-1
c
c----- calculate distance between points
c
            t1=dble((float(i)*nint)**2)
            t1=t1+dble((float(j)*mint)**2)
            t1=dsqrt(t1)
c
c----- now compute relevant value of the gaussian
c
            if(t1.gt.10.0*L) then
               C(i,j)=0.0
               goto 105
               endif
            t2=t1/dble(mint)
            t2=t2**2
            C(i,j)=lc*sngl(t5**t2)
105         continue
100      continue

      return
      end
```

```
@PROCESS DC(GAUS),OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS DIRECTIVE('*VD:')
c ---------------------------------------------------------------------
c
c Brief Update History
c --------------------
c MEMNL$5 - considerably modified jan 91, peter whiting
c MEMNL$6 - considerably modified ->dec 91, peter whiting
c MEMNL$8 - modified to read C matrix from dynamic common
c                ->dec 92, peter whiting
c         - also convert to ESSL subroutines for the intensive
c           processing  -> jan 92, peter whiting
c --------------------
c
c The notes directly below are Steve Brown's original comments.
c
c          Maximum entropy image reconstruction subroutine.
c          ------- ------- ----- -------------- ----------
c
c  This package implements the ME reconstruction algorithm presented in
c  J.Skilling and R.K.Bryan, (1984) Mon.Not.R.astr.Soc. 211, 111-124.
c
c  This routine solves for the maximum entropy image for both linear
c  and non-linear image reconstruction problems.
c  Each call to the routine MEM_NL takes one iteration from the current
c  image towards the maximum entropy image.
c
c  The program tries to follow the notation of Skilling and Bryan as
c  much as possible.  All page and equation references in the program
c  comments refer to their paper.
c
c  Written:  Steve Brown
c            Department of Applied Mathematics
c            University of Sydney
c            NSW Australia.
c
c Version 1.  13-October-89  (Not fully tested but works OK so far)
c
c ---------------------------------------------------------------------
c
      subroutine MEMNL(n, m, I, D, F, dm, dn, A, chi0, Rchange,
     &                                          entropy, chi2, test, nlin)

c  This routine is called by the driving program.  It advances the
c  solution one iteration.
c
c  Input Parameters:
c            n - number of image elements in the image vector I.
c            m - number of data elements in the data vector D.
c            I - array of n image values.  On input I holds the current
c                image.  I corresponds to the f of Skilling and
c                Bryan [Eqn 1].
c            D - array of m experimental data values the image is to
c                be fitted to [Eqn 2].
c            F - for non-linear problems this is an array of m model
c                data values corresponding to the image in I.  For
c                linear problems F is a work array and must be
c                dimensioned to m or larger.
c            R - response matrix.  It has physical dimensions (dm,dn)
c                with the first (m,n) containing the actual response
c                matrix elements.  The response matrix is defined as
c                R(k,j) = dF(k)/dI(j)
c           dm - dimensioned rows of array R.
c           dn - dimensioned columns of array R.
c            A - default intensity [Page 114, Eqn 6].
c         chi0 - the desired chi-squared fit statistic [Eqn 4].
c      Rchange - used only for linear image reconstruction.  Rchange must
c                be set to one on the first call and on any subsequent
c                call where the response matrix, image vector or data
c                vector have been changed.  Rchange should be set to
c                zero if these parameters are unchanged and the code will
```

```
c               execute a bit faster.
c         nlin - must be set to one for non-linear problems, zero otherwise.
c
c  Output Parameters:
c             I - updated image vector.
c       entropy - for linear problems entropy is set to the current image
c                 entropy [Eqn 6].  For non-linear problems it contains the
c                 entropy of the input image.
c          chi2 - for linear problems chi2 is set to the current chi-squared
c                 statistic [Eqn 4]. For non-linear problems it contains the
c                 chi-squared value at the start of the iteration.
c          test - for linear problems test is set to the entropy test
c                 recommended by Skilling and Bryan [Eqn 37] for the current
c                 image.  For non-linear problems it contains the test value
c                 of the initial image.
c
c  The parameter dne sets the number of search directions used.
c
c  Limits:
c       The number of image points n must be <= parameter nmax and the
c       number of data points m must be <= mmax.
c
c  Warning !!!
c       The parameters nmax and mmax are used in most of the subprograms
c       and you must change them all if you want to use larger image or
c       data vectors
c
c End of Steve Brown's comments.
c ------------------------------
c
c----- nxx   = number of rows in discrete velocity model
c----- mxx   = number of columns in discrete velocity model
c----- nmaxq = maximum number of velocity nodes
c----- mmaxq = maximum number of data rays
c----- maxq  = maximum of nmax and mmax
c
      integer   n, m, dn, dm, nmax, mmax, Rchange, ne, dne, nlin
      integer   nmaxq,mmaxq,maxq,dneq
      integer   nxx,mxx,ndat
      real*4    nint,mint
      real*4    I(n), D(m), F(m), A, chi0, entropy,
     &          chi2, test

c  nmax is the largest allowed image size.  mmax is the largest allowed
c  data size.  dne sets the number of search directions used.

       parameter(nmaxq=17000,mmaxq=4200,maxq=17000)
      parameter (dneq = 4)

      real*4    gS(nmaxq), gC(nmaxq), S0, C0, Itot,
     &          sum1, sum2, Smu(dneq), Cmu(dneq), gamma(dneq),
     &          e(nmaxq,dneq), x(dneq), almin, Caim

       common /conparm/nxx,mxx,nint,mint,nmax,mmax,max,ndat,dne

c  The gradients and values of S and C are stored in a common so their
c  values are preserved between subroutine calls.

      common /mem5cm/ gS, gC, C0, S0, Itot


      if ((n .gt. nmax) .or. (m .gt. mmax)) then
          print *, 'Error in MEM: nmax or mmax are too small'
          stop
      endif

c  If first call or the response matrix R has changed it is necessary to
c  calculate the gradient information.

      if ((Rchange .eq. 1) .or. (nlin .eq. 1)) then
          call CALCGRAD(n, m, I, D, F, dm, dn, A, nlin)
```

```
            call CALCTEST(n, test)
        endif

c  Calculate the normalised search directions.

        ne = dne
        call SEARCH(e, ne, I, n, m, dm, dn)

c  Now diagonalise the search subspace and construct quadratic models
c  for S and C.

        call DIAGMOD(e, gamma, Smu, Cmu, ne, n, m, dm, dn, I)

c  Calculate the constants needed by the CONTROL subroutine.

        call CONSTS(gamma, ne, almin, Caim, chi0, C0, Cmu, Itot)

c  Now calculate the increments x to move in each of the search
c  directions.

        call CONTROL(x, S0, C0, Smu, Cmu, gamma, Caim, almin, ne)

c  Update the image vector along the search directions.

        call UPDATE(x, e, ne, I, n)

c  Calculate the gradient information which is needed to calculate test.

        if (nlin .ne. 1) then
            call CALCGRAD(n, m, I, D, F, dm, dn, A, nlin)

c  Calculate test.

            call CALCTEST(n, test)
        endif

        entropy = S0
        chi2 = C0

        return
        end
@PROCESS DC(GAUS),OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS DIRECTIVE('*VD:')
c
c ---------------------------------------------------------------------
c
        subroutine CALCGRAD(n, m, I, D, F, dm, dn, A, nlin)

        integer   n, m, dn, dm, nmax, j, k, l, nlin, mmax, x, max
        integer   nmaxq,mmaxq,maxq
        integer   nxx,mxx,ndat,dne
        integer*4 zz,fcb(20),rcb(20)
        integer*4 i1,i2,i3,i4,vcount,Cind
        logical set
        real*4 nint,mint
        real*4    A, I(n), D(m), F(m)
        real*4 t1,sumpl,tol
        real*4 sig,ai,bi,XXerr,XXerr2,tvar

         parameter(nq=90,mq=210)
         parameter(nmaxq=17000,mmaxq=4200,maxq=17000)
        real*4 Am(nmaxq),R(nmaxq)
        real*4    gS(nmaxq), gC(nmaxq), S0, C0, sum1, sum2,sum3
       &          Itot, temp(nmaxq) , C , Ctmp
        real*8  la,li
         dimension C(0:nq-1,0:mq-1),Ctmp(nq*mq),Cind(nq*mq)
        dimension tvar(mmaxq)
        dimension sig(mmaxq),set(nmaxq)
        dimension ai(mmaxq),bi(mmaxq),XXerr(mmaxq),XXerr2(mmaxq)

        common /mem5cm/ gS, gC, C0, S0, Itot
```

```
      common /backg/Am
      common /gaus/C
      common /frechet/fcb,rcb
      common /setvel/set
      common /var/sig,ai,bi,XXerr,XXerr2
      common /conparm/nxx,mxx,nint,mint,nmax,mmax,max,ndat,dne

      tol=1.0e-10
c
c----- Finds the current chi-squared value
c
          sum2 = 0.0
          do 30 k = 1, m
              sum2 = sum2 + ((F(k) - D(k))**2)*sig(k)
30        continue
          C0 = sum2

c  Finds the current image entropy [Eqn 6] and gradient of S [Eqn 7a].
      Itot=0.0
      sum1 = 0.0
      do 40 j = 1, n
          la = dlog(dble(Am(j)))
          li = dlog(dble(I(j)))
          gS(j) = real(la - li)
          sum1 = sum1 + I(j)*(gS(j)+1.0)
          Itot = Itot + I(j)
40    continue
      S0 = sum1

c  Finds the gradient of C [Eqn 8].
c----- note the Covariance term (see TN.10)
       do 48 k = 1, m
          tvar(k)=XXerr(k)*sig(k)*2.0+ai(k)*XXerr2(k)
48        continue
       do 49 j = 1, n
          temp(j)=0.0
49        continue
      do 50 k = 1, m
          x=(k-1)*n
          call dmi014(R(1),n,x,rcb,fcb,1,zz)
          if(zz.ne.0) stop 'error reading frechet'
          do 60 j = 1, n
             temp(j)=temp(j)+R(j)*tvar(k)
60           continue
50    continue
      if(C(1,1).eq.0.0) then
         call scopy(n,temp,1,gC,1)
         return
         endif
      do 51 j = 1, n
          if(set(j)) then
             gC(j)=0.0
             goto 51
             endif
          sum1 = 0.0
          sum3 = 0.0
          vcount=0
          do 61 k = 1, n
             if(set(k)) goto 61
             t1=C(int((j-k)/mq),abs(mod(j-1,mq)-mod(k-1,mq)))
             if(t1.eq.0.0) goto 61
             vcount=vcount+1
             Ctmp(vcount)=t1
             Cind(vcount)=k
61           continue
          sum1=sdoti(vcount,Ctmp(1),Cind(1),temp(1))
          sum3=sasum(vcount,Ctmp(1),1)
          if(abs(sum1).gt.tol) then
             gC(j) = sum1/sum3
          else
             gC(j) = 0.0
```

```
            endif
51     continue

       return
       end
@PROCESS DC(GAUS),OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS DIRECTIVE('*VD:')
c
c ---------------------------------------------------------------------
c
       subroutine SEARCH(e, ne, I, n, m, dm, dn)

       integer   ne, n, m, nmax, mmax, dne, j, k, l, p, dm, dn
       integer   nmaxq,mmaxq,maxq,dneq
       integer   ndat,fcb(20),rcb(20)
       integer x,y,z,max,v,q,zz
       integer i1,i2,i3,i4,i5,i6,vcount,Cind
       integer*4 nxx,mxx
       logical set
       real*4 nint,mint,C,Ctmp
       real*4    I(n)
       real*4 t1,t2,t3,t4,t5,sum3

        parameter(nq=90,mq=210)
       parameter(nmaxq=17000,mmaxq=4200,maxq=17000)
       parameter (dneq = 4)
       real*4    gS(nmaxq), gC(nmaxq), S0, C0, temp(nmaxq), temp2(mmaxq),
      &          temp3(nmaxq),temp1(nmaxq),Itot,e(nmaxq,dneq),sum1, sum2
       real*4 sig(mmaxq),R(nmaxq)
       real*4 ai(mmaxq),bi(mmaxq),XXerr(mmaxq),XXerr2(mmaxq)
        dimension C(0:nq-1,0:mq-1),Ctmp(nq*mq),Cind(nq*mq)
       dimension set(nmaxq)

       common /mem5cm/ gS, gC, C0, S0, Itot
       common /var/sig,ai,bi,XXerr,XXerr2
       common /setvel/set
       common /gaus/C
       common /frechet/fcb,rcb
       common /conparm/nxx,mxx,nint,mint,nmax,mmax,max,ndat,dne

c  Routine finds dne search directions according to Eqn 20 and the
c  paragraph above Eqn 19.

c  Finds gC The resulting search direction is normalised.
       sum1 = 0.0
       call scopy(n,gC(1),1,e(1,1),1)
       sum1=snorm2(n,e(1,1),1)
       if (sum1 .eq. 0.0) then
           sum1 = 1.0
           endif
       do 20 j = 1, n
           e(j,1) = e(j,1) / sum1
20     continue
       if (ne .eq. 1) return

c  Finds gS The resulting search direction is normalised.
       sum1 = 0.0
       call scopy(n,gS(1),1,e(1,2),1)
       sum1=snorm2(n,e(1,2),1)
       if (sum1 .eq. 0.0) then
           sum1 = 1.0
           endif
       do 40 j = 1, n
           e(j,2) = e(j,2) / sum1
40     continue

c  Loops around multiplying direction p-2 by matrix ggC to get
c  direction p which is then normalised.
c  Need to do this in four parts now to get ggC=C.Rt.R.C

       do 50 p = 3, ne
```

```
      if(C(1,1).eq.0.0) then
         call scopy(n,e(1,p-2),1,temp1,1)
         goto 64
         endif
      do 60 k = 1, n
          if(set(k)) then
             temp1(k)=0.0
             goto 60
             endif
          sum2 = 0.0
          sum3 = 0.0
          vcount=0
          do 70 j = 1, n
             if(set(j)) goto 70
             t1=C(int((j-k)/mq),abs(mod(j-1,mq)-mod(k-1,mq)))
             if(t1.eq.0.0) goto 70
             vcount=vcount+1
             Ctmp(vcount)=t1
             Cind(vcount)=j
70           continue
          sum2=sdoti(vcount,Ctmp(1),Cind(1),e(1,p-2))
          sum3=sasum(vcount,Ctmp(1),1)
          temp1(k) = sum2/sum3
60    continue

64    do 61 k = 1, m
          sum2 = 0.0
          x=(k-1)*n
          call dmi014(R(1),n,x,rcb,fcb,1,zz)
          if(zz.ne.0) stop 'error reading frechet'
          sum2=sdot(n,temp1,1,R,1)
          temp2(k)=sum2*(2.0*sig(k)+4.0*ai(k)*XXerr(k)+
     +                bi(k)*XXerr2(k))
61    continue

      do 8872 j = 1, n
         temp3(j)=0.0
8872     continue
      do 62 k = 1, m
          x=(k-1)*n
          call dmi014(R(1),n,x,rcb,fcb,1,zz)
          if(zz.ne.0) stop 'error reading frechet'
          call syax(n,temp2(k),R(1),1,temp3(1),1)
62    continue

      if(C(1,1).eq.0.0) then
         call scopy(n,temp3,1,temp,1)
         goto 67
         endif
      do 63 k = 1, n
          if(set(k)) then
             temp(k)=0.0
             goto 63
             endif
          sum2 = 0.0
          sum3 = 0.0
          vcount=0
          do 73 j = 1, n
             if(set(j)) goto 73
             t1=C(int((j-k)/mq),abs(mod(j-1,mq)-mod(k-1,mq)))
             if(t1.eq.0.0) goto 73
             vcount=vcount+1
             Ctmp(vcount)=t1
             Cind(vcount)=j
73           continue
          sum2=sdoti(vcount,Ctmp(1),Cind(1),temp3)
          sum3=sasum(vcount,Ctmp(1),1)
          temp(k) = sum2/sum3
63    continue

67    sum1 = 0.0
```

```
          call szaxpy(n,2.0,temp1(1),1,temp(1),1,e(1,p),1)
          sum1=snorm2(n,e(1,p),1)

          if (sum1 .eq. 0.0) then
              sum1 = 1.0
              endif
          do 100 j = 1, n
              e(j,p) = e(j,p) / sum1
100       continue
50    continue

      return
      end
@PROCESS DC(GAUS),OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS DIRECTIVE('*VD:')
c
c ---------------------------------------------------------------------
c
      subroutine DIAGMOD(e, gamma, Smu, Cmu, ne, n, m, dm, dn , I)

      integer   ne, n, m, nmax, mmax, dne, j, k, ll, p, q, qq, tmp,
     &          dm, dn, max
      integer   i1,i2,i3,i4,zz,nmaxq,mmaxq,maxq,dneq,vcount,Cind
      integer   nxx,mxx,ndat,x,fcb(20),rcb(20)
      logical   set
      real*4    sumf,sumf2,twoA,sum3,L,L0
      real*4    nint,mint,C,Ctmp,t1

       parameter(nq=90,mq=210)
       parameter(nmaxq=17000,mmaxq=4200,maxq=17000)
      parameter (dneq = 4)
      real*4 I(nmaxq),Ilog(nmaxq),temp2(nmaxq,dneq)
      real*4 length(dneq)
      real*4    gS(nmaxq), gC(nmaxq), S0, C0, Itot, temp(mmaxq,dneq),
     &          temp3(nmaxq,dneq),
     &          sum1, sum2, g(dneq,dneq), MM(dneq,dneq), Smu(dneq),
     &          Cmu(dneq), gamma(dneq), e(nmaxq,dneq), V(dneq,dneq),
     &          eigval(dneq), e2(nmaxq,dneq)
      real*4 sig(mmaxq),R(nmaxq)
      real*4 ai(mmaxq),bi(mmaxq),XXerr(mmaxq),XXerr2(mmaxq)
       dimension C(0:nq-1,0:mq-1),Ctmp(nq*mq),Cind(nq*mq)
      dimension set(nmaxq)

      common /mem5cm/ gS, gC, C0, S0, Itot
      common /penalt/length,L,L0
      common /var/sig,ai,bi,XXerr,XXerr2
      common /setvel/set
      common /gaus/C
      common /frechet/fcb,rcb
      common /conparm/nxx,mxx,nint,mint,nmax,mmax,max,ndat,dne

c  Routine constructs and diagonalises the search subspace according to
c  the procedures in Eqn 24 and Section 3.7.1.

c  Calculate the metric tensor g [Eqn 24b].
1000    continue
          do 8 p=1,dne
             do 9 q=1,dne
                g(p,q)=0.0
9               continue
8            continue
          do 10 p = 1, ne
              do 20 q = p, ne
                  sum1 = 0.0
                  do 30 j = 1, n
                      sum1 = sum1 + e(j,p)*e(j,q)/I(j)
30                continue
                  g(p,q) = sum1
                  g(q,p) = sum1
20            continue
10        continue
```

```
c  Diagonalise g. Eigenvectors are columns in the array V
         call JACOBI(g,ne,dne,eigval,V,qq)

c  If any of the eigenvalues of g are small then throw out that direction
c  and repeat the calculation of g etc.  This protects against linear
c  dependence in the search directions.  NOTE the factor 1.e-4 is an
c  arbitrary choice.
         tmp = 0
         p = 1
 1001    continue
             if (abs(eigval(p)).lt.amax1(eigval(1),eigval(2),eigval(3),
     +           eigval(4))/10000.0) then
c    +           eigval(4),eigval(5),eigval(6))/10000.0) then
                 do 40 q = p, ne-1
                     call scopy(n,e(1,q+1),1,e(1,q),1)
                     eigval(q) = eigval(q+1)
40               continue
                 ne = ne - 1
                 tmp = 1
                 if (p .le. ne) goto 1001
             endif
             p = p + 1
         if (p .le. ne) goto 1001

      if (tmp .eq. 1) goto 1000

c  Rescale the search directions so that the metric is cartesian.  The
c  new directions are held in the matrix e2.
      do 60 j = 1, n
          do 70 p = 1, ne
              sum1 = 0.0
              do 80 q = 1, ne
                  sum1 = sum1 + e(j,q)*V(q,p)
80            continue
              e2(j,p) = sum1 / sqrt(abs(eigval(p)))
70        continue
60    continue

c  Now calculate M (held in matrix MM) [Eqn 24d].
c  Need extra work to do R.C.e
          do 18 p=1,dne
             do 19 q=1,dne
                MM(p,q)=0.0
19           continue
18        continue
      if(C(1,1).eq.0.0) then
         do 66 p=1,ne
            call scopy(n,e2(1,p),1,temp3(1,p),1)
66          continue
         goto 67
         endif
      do 90 k = 1, n
         if(set(k)) then
            do 9191 p=1,ne
               temp3(k,p)=0.0
9191           continue
            goto 90
            endif
         vcount=0
         do 110 j = 1, n
            if(set(j).or.set(k)) goto 110
            t1=C(int((j-k)/mq),abs(mod(j-1,mq)-mod(k-1,mq)))
            if(t1.eq.0.0) goto 110
            vcount=vcount+1
            Ctmp(vcount)=t1
            Cind(vcount)=j
110         continue
         do 100 p = 1, ne
            sum1=sdoti(vcount,Ctmp(1),Cind(1),e2(1,p))
            sum3=sasum(vcount,Ctmp(1),1)
            temp3(k,p) = sum1/sum3
```

```
100       continue
90     continue
67     do 91 k = 1, m
           x=(k-1)*n
           call dmi014(R(1),n,x,rcb,fcb,1,zz)
           if(zz.ne.0) stop 'error reading frechet'
           do 101 p = 1, ne
               sum1=sdot(n,R(1),1,temp3(1,p),1)
               temp(k,p)=sum1*(2.0*sig(k)+4.0*ai(k)*XXerr(k)+
     +                         bi(k)*XXerr2(k))
101        continue
91     continue
       do 120 p = 1, ne
           do 130 q = p, ne
               sum1=sdot(m,temp(1,p),1,temp(1,q),1)
               sum2=sdot(n,e2(1,p),1,temp3(1,q),1)
               MM(p,q) = sum1+2.0*sum2
               MM(q,p) = MM(p,q)
130        continue
120    continue

        write(6,'(2x,a14,i5)') 'ne is now (c) ',ne

c  Diagonalise M. Eigenvectors are columns in the array V
       call JACOBI(MM,ne,dne,gamma,V,qq)

c  Rescale the search directions so that M is diagonal.  The
c  new directions are held in the matrix e.
       do 150 j = 1, n
           do 160 p = 1, ne
               sum1 = 0.0
               do 170 q = 1, ne
                   sum1 = sum1 + e2(j,q)*V(q,p)
170            continue
               e(j,p) = sum1
160        continue
150    continue

c
c----- compute lengths of these final search directions to be passed
c----- to CONTROL for correct distance penalty implementation
c
        do 155 p=1,ne
           length(p)=snorm2(n,e(1,p),1)
155        continue

c  Calculate the final components of the quadratic models Smu and Cmu
c  [Eqns 24a and 24c].
       do 180 p = 1, ne
          Smu(p)=sdot(n,e(1,p),1,gS(1),1)
          Cmu(p)=sdot(n,e(1,p),1,gC(1),1)
180        continue

       return
       end
@PROCESS DC(GAUS),OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS DIRECTIVE('*VD:')
c
c ---------------------------------------------------------------------
c
       subroutine CONSTS(gamma, ne, almin, Caim, chi0, C0, Cmu, Itot)

       integer   ne, p, dne, dneq
       integer   nxx,mxx,nmax,mmax,max,ndat
       real*4    chi0,chi0old
       real*4    nint,mint
       real*4    gamma(ne), Cmu(ne), almin, Caim, L0, C0, Itot,
      &          sum1, Cmin,L
       real*4 dlimit

        parameter(dneq=4)
```

```
      real*4 length(dneq)
      common /dist/dlimit
      common /penalt/length,L,L0
      common /conparm/nxx,mxx,nint,mint,nmax,mmax,max,ndat,dne

c  Routine calculates the constants needed by the control procedure.

c  Find Cmin [Eqn 28].
      sum1 = 0.0
      do 10 p = 1, ne
          sum1 = sum1 + (Cmu(p)**2)/gamma(p)
10    continue
      Cmin = C0 - 0.5*sum1

c  Find Caim [Eqn 29].
c  Skilling and Bryans choice of 0.667 and 0.333.
      Caim = 0.667*Cmin + 0.333*C0
      if (chi0 .gt. Caim)  Caim = chi0
      if(Caim.gt.C0) Caim=C0

c  Find L0 [Paragraph above Eqn 28].
c  *** I changed this from using Itot to using nmax, which ***
c  *** I feel is a more useful limit. Peter W 13/03/91     ***
      L0 = (2.0*nmax)/dlimit

c  Find alpha min [Paragraph below Eqn 32].
      almin = -gamma(1)
      do 20 p = 2, ne
          sum1 = -gamma(p)
          if (sum1 .gt. almin)  almin = sum1
20    continue
      if (almin .lt. 0.0)  almin = 0.0
      return
      end
@PROCESS DC(GAUS),OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS DIRECTIVE('*VD:')
c
c ---------------------------------------------------------------------
c
      subroutine CONTROL(x, S0, C0, Smu, Cmu, gamma, Caim, almin,ne)

      integer   ne, q, asuc, psuc, afinished, pfinished, dne, dneq
      integer   nxx,mxx,nmax,mmax,max,ndat
      parameter (dneq = 4)
      real*4 length(dneq)
      real*4    nint,mint
      real*4    x(ne), S0, C0, Smu(ne), Cmu(ne), gamma(ne), Caim, L0,
     &          almin, alpha, p, C, Cp, L, xs(dneq), alow, plow, ahigh,
     &          phigh, Caimh
      real*4    Lsuc,Csuc

      common /penalt/length,L,L0
      common /conparm/nxx,mxx,nint,mint,nmax,mmax,max,ndat,dne

c  Implements the control procedure charted on Page 122, Figure 3 and
c  described in Sections 3.7.2 and 3.7.3.

      if (C0 .lt. Caim*1.01) then
          Caimh = Caim*1.01
      else
          Caimh = Caim
      endif

      p = 0.0
      plow = 0.0
      phigh = -1.0
      psuc = 0
      pfinished = 0

c  Start of the P chop.
   10 continue
```

```
          alow = almin
          ahigh = -1.0
          alpha = almin + 1.0
          asuc = 0
          afinished = 0

c  Start of the alpha chop
   20     continue

c  Calculate C [Eqn 23b], Cp [Eqn 35], L [Eqn 27c] and x [Eqn 34].
               C  = C0
               Cp = C0
               L  = 0.0
               do 1 q = 1, ne
                   x(q) = (alpha*Smu(q) - Cmu(q)) / (P+gamma(q)+alpha)
                   C    = C + Cmu(q)*x(q) + 0.5*gamma(q)*(x(q)**2)
                   Cp   = Cp + Cmu(q)*x(q) + 0.5*(p+gamma(q))*(x(q)**2)
                   L    = L + length(q)*x(q)**2
1              continue

               if ((Cp .gt. C0) .and. (Cp .gt. Caimh)) then
                   ahigh = alpha
                   alpha = 0.5*(alpha + alow)
               elseif(L.gt.L0) then
                   if(Cp.le.C0) then
                       alow=alpha
                       if(ahigh.gt.0.0) then
                           alpha=0.5*(ahigh+alpha)
                       else
                           alpha=2.0*alpha
                           endif
                   else
                       ahigh=alpha
                       alpha=0.5*(alpha+alow)
                       endif
               elseif(C.lt.Caim) then
                   if(Caimh.ge.C0) then
                       do 2 q = 1, ne
                           xs(q) = x(q)
2                          continue
                       asuc=1
                       Lsuc=L
                       Csuc=C
                       endif
                   alow=alpha
                   if(ahigh.gt.0.0) then
                       alpha=0.5*(ahigh+alpha)
                   else
                       alpha=2.0*alpha
                       endif
               else
                   do 4 q = 1, ne
                       xs(q) = x(q)
4                      continue
                   asuc=1
                   Lsuc=L
                   Csuc=C
                   ahigh=alpha
                   alpha=0.5*(alow+alpha)
                   endif

               if ((ahigh.gt.0.0) .and. ((ahigh-alow).lt.
     &                (1.0E-3*ahigh+1.0E-10)))  afinished = 1
               if (alpha .gt. 1.0E20) afinished = 1

               if (afinished .ne. 1) goto 20
c  End of alpha chop

          if (asuc .eq. 1) then
              psuc = 1
              phigh = p
```

```
            p = 0.5*(plow + p)
         else
            plow = p
            if (phigh .gt. 0.0) then
               p = 0.5*(phigh + p)
            else
               p = 2.0*(p + 1.0)
            endif
         endif

         if ((psuc.eq.1) .and. ((p.eq.0.0) .or. (phigh-plow).lt.
     &               (1.0E-3*phigh+1.0E-10)))  pfinished = 1
c check for occasional bad topology (rare)
         if((psuc.eq.0).and.(p.gt.1.0e20)) then
            pfinished=1
            write(2,'(2x,a27)') 'P chop blow-up encountered'
            endif

         if (pfinished .ne. 1) goto 10
c  End of P chop

c  Topology correction
        if(psuc.eq.0) then
           do 5 q=1,ne
              x(q)=1.0e-2*L0*(Smu(q)-Cmu(q))
5             continue
           endif

c  Copy the most recent successful x (array xs) to the output increment
c  array x.
3333  do 3 q = 1, ne
         x(q) = xs(q)
3     continue

      WRITE(2,'(2X,A36,G16.8)')'MEM_NL$: expected Chi in CONTROL - ',Cp
      WRITE(2,'(2X,A36,G16.8)')'MEM_NL$: expected ENT in CONTROL - ',S
      WRITE(2,'(2X,A36,G16.8)')'MEM_NL$: calculated distance     - ',L
      WRITE(2,'(2X,A36,G16.8)')'MEM_NL$: maximum distance        - ',L0
      WRITE(2,'(2X,A26,G16.8)')'MEM_NL$: minimum alpha - ',almin
      WRITE(2,'(2X,A26,G16.8)')'MEM_NL$: final alpha   - ',alpha

      return
      end
@PROCESS DC(GAUS),OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS DIRECTIVE('*VD:')
c
c ---------------------------------------------------------------------
c
      subroutine UPDATE(x, e, ne, I, n)

      integer   ne, n, dne, nmax, j, p, mmax
      integer   nmaxq,mmaxq,maxq,dneq
      integer   nxx,mxx,max,ndat
      real*4    I(n),sum
      real*4    nint,mint
       parameter(nmaxq=17000,mmaxq=4200,maxq=17000)
      parameter (dneq = 4)
      real*4    e(nmaxq, dneq), x(ne)

       common /conparm/nxx,mxx,nint,mint,nmax,mmax,max,ndat,dne

c  Update the image vector [Eqn 25].
      sum=0.0
      do 10 p = 1, ne
         do 20 j = 1, n
            I(j) = I(j) + x(p)*e(j,p)
            sum=sum+(x(p)*e(j,p))**2
 20       continue
 10    continue

       write(6,'(2x,a25,g16.8)') 'actual distance of update',sqrt(sum)
```

```
c  Protect against stray negative image values.
      do 30 j = 1, n
         if (I(j) .le. 0.0)  I(j) = 1.e-6
30    continue

      return
      end
@PROCESS DC(GAUS),OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS DIRECTIVE('*VD:')
c
c ---------------------------------------------------------------------
c
      subroutine CALCTEST(n, test )

      integer   n, nmax, j ,mmax
      integer   nmaxq,mmaxq,maxq
      integer   nxx,mxx,max,ndat,dne
      real*4    test
      real*4    nint,mint
       parameter(nmaxq=17000,mmaxq=4200,maxq=17000)
      real*4    gS(nmaxq), gC(nmaxq), S0, C0, Itot, sum1, sum2, sum3

      common /mem5cm/ gS, gC, C0, S0, Itot
      common /conparm/nxx,mxx,nint,mint,nmax,mmax,max,ndat,dne

c  Finds Skilling and Bryans test parameter [Eqn 37].
      sum1 = 0.0
      sum2 = 0.0
      do 10 j = 1, n
          sum1 = sum1 + gS(j)**2
          sum2 = sum2 + gC(j)**2
10    continue
      sum1 = sqrt(sum1)
      sum2 = sqrt(sum2)
      sum3 = 0.0
      if ((sum1 .ne. 0.0) .and. (sum2 .ne. 0.0)) then
          do 20 j = 1, n
              sum3 = sum3 + (gS(j)/sum1 - gC(j)/sum2)**2
20        continue
      endif
      test = 0.5 * sum3

      return
      end
@PROCESS DC(GAUS),OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS DIRECTIVE('*VD:')
c
c ---------------------------------------------------------------------
c
      SUBROUTINE JACOBI(A,N,NP,D,V,NROT)
      implicit real*4    (a-h,o-z)
      implicit integer (i-n)
      PARAMETER (NMAX=4)
      DIMENSION A(NP,NP),D(NP),V(NP,NP),B(NMAX),Z(NMAX)
      DO 12 IP=1,N
        DO 11 IQ=1,N
          V(IP,IQ)=0.
11      CONTINUE
        V(IP,IP)=1.
12    CONTINUE
      DO 13 IP=1,N
        B(IP)=A(IP,IP)
        D(IP)=B(IP)
        Z(IP)=0.
13    CONTINUE
      NROT=0
      DO 24 I=1,50
        SM=0.
        DO 15 IP=1,N-1
          DO 14 IQ=IP+1,N
            SM=SM+ABS(A(IP,IQ))
```

```
14          CONTINUE
15        CONTINUE
          IF(SM.EQ.0.)RETURN
          IF(I.LT.4)THEN
            TRESH=0.2*SM/N**2
          ELSE
            TRESH=0.
          ENDIF
          DO 22 IP=1,N-1
            DO 21 IQ=IP+1,N
              G=100.*ABS(A(IP,IQ))
              IF((I.GT.4).AND.(ABS(D(IP))+G.EQ.ABS(D(IP)))
     *           .AND.(ABS(D(IQ))+G.EQ.ABS(D(IQ))))THEN
                A(IP,IQ)=0.
              ELSE IF(ABS(A(IP,IQ)).GT.TRESH)THEN
                H=D(IQ)-D(IP)
                IF(ABS(H)+G.EQ.ABS(H))THEN
                  T=A(IP,IQ)/H
                ELSE
                  THETA=0.5*H/A(IP,IQ)
                  T=1./(ABS(THETA)+SQRT(1.+THETA**2))
                  IF(THETA.LT.0.)T=-T
                ENDIF
                C=1./SQRT(1+T**2)
                S=T*C
                TAU=S/(1.+C)
                H=T*A(IP,IQ)
                Z(IP)=Z(IP)-H
                Z(IQ)=Z(IQ)+H
                D(IP)=D(IP)-H
                D(IQ)=D(IQ)+H
                A(IP,IQ)=0.
                DO 16 J=1,IP-1
                  G=A(J,IP)
                  H=A(J,IQ)
                  A(J,IP)=G-S*(H+G*TAU)
                  A(J,IQ)=H+S*(G-H*TAU)
16              CONTINUE
                DO 17 J=IP+1,IQ-1
                  G=A(IP,J)
                  H=A(J,IQ)
                  A(IP,J)=G-S*(H+G*TAU)
                  A(J,IQ)=H+S*(G-H*TAU)
17              CONTINUE
                DO 18 J=IQ+1,N
                  G=A(IP,J)
                  H=A(IQ,J)
                  A(IP,J)=G-S*(H+G*TAU)
                  A(IQ,J)=H+S*(G-H*TAU)
18              CONTINUE
                DO 19 J=1,N
                  G=V(J,IP)
                  H=V(J,IQ)
                  V(J,IP)=G-S*(H+G*TAU)
                  V(J,IQ)=H+S*(G-H*TAU)
19              CONTINUE
                NROT=NROT+1
              ENDIF
21          CONTINUE
22        CONTINUE
          DO 23 IP=1,N
            B(IP)=B(IP)+Z(IP)
            D(IP)=B(IP)
            Z(IP)=0.
23        CONTINUE
24      CONTINUE
        PAUSE '50 iterations should never happen'
        RETURN
        END
```

```
@PROCESS DC(GAUS),OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS DIRECTIVE('*VD:')
c-------------------------------------------------------------------------------
c-------------------------------------------------------------------------------
c----- program to perform maximum entropy reflection tomographic imaging  -----
c----- utilises the 'directional reception' method of ray tracing         -----
c----- uses Chuck Sword's XERR statistic                                  -----
c----- uses Steve Brown's MEM_NL subroutine for the ME optimisation       -----
c----- updates to MEM_NL for smoothing                                    -----
c----- all real calculations in double-precision                          -----
c----- modified to use a tipex scratch file for the frechet matrix        -----
c-------------------------------------------------------------------------------
c      program metomos
c-------------------------------------------------------------------------------
c----- ndatq = maximum number of data rays used in inversion
c----- ndat  = actual number of data rays
c-----
c----- nq    = maximum number of rows in velocity grid
c----- mq    = maximum number of columns in velocity grid
c----- n     = actual number of rows in velocity grid
c----- m     = actual number of columns in velocity grid
c-----
c----- nmaxq = maximum number of velocity nodes
c----- mmaxq = maximum number of data rays
c----- maxq  = maximum of nmaxq and mmaxq
c----- nmax  = actual number of velocity nodes
c----- mmax  = actual number of data rays
c----- max   = maximum of nmax and mmax
c-----
c----- nint = spatial increment between grid rows
c----- mint = spatial increment between grid columns
c-----
c----- chi0         = desired chi value
c-----
c----- LL    = gaussian half-width (in metres)
c-----
c----- vel(nq,mq)= values of velocity at grid points
c----- Am(nmaxq)= the image vector storing the prior information
c----- Rray(nq*mq)= temporary vector to hold Frechet derivative values
c-----
c----- sloc1 = source location of first ray
c----- tang1 = take-off angle in radians (positive angle implies
c-----         in direction of increasing distance) of first ray
c----- sloc2 = source location of second ray
c----- tang2 = take-off angle in radians (positive angle implies
c-----         in direction of increasing distance) of second ray
c----- xerr  = Chuck Sword's statistic
c-----
c----- i,j,k= temporary integer variables
c----- t1,t2,t3,t4,t5= temporary double-precision variables
c-----
c----- chimem(5) = vector for remembering latest iterations for
c-----             automatic stage changes, chi0 determination, and
c-----             termination
c-----
c----- Image(nq*mq+5)= vector of velocity values
c----- D(ndatq)      = data travel times
c----- zero(ndatq)   = array storing zero
c----- F(ndatq)      = current model travel times
c----- chi2          = chi value at start of iteration
c----- entropy       = value of entropy at start of iteration
c----- test          = Skilling and Bryan's test statistic at start of iteration
c-----
c----- raybad  = logical vector to remember the rays that had Xerr's
c-----           greater than the threshold.
c----- sig(ndatq)= holds the ray weights, computed by VARIAN
c----- ai(ndatq), bi(ndatq)  =  assist in the application of the weights
c-----
c----- numnull = records the number of nulled rays ie xerr>threshold in XERRRT
c----- numturn = records the number of turned rays nulled in XERRRT
c-----
```

```
c----- set(n*m) = logical array set to .true. is vel node not to change
c-----
c----- ucount  = 'unstability count'. incremented when dlimit increased
c-----
c-------------------------------------------------------------------------
      real*4 sloc1,tang1,xerr
      real*4 sloc2,tang2
      real*4 nint,mint
      real*4 ray1travel,ray2travel
      real*4 Rray,D,zero,F,Fold
      real*4 Image,Am
      real*4 chi0,chi2,entropy,test,testold
      real*4 dlimit,pctd
      real*4 length,L,L0
      real*4 vel
      real*4 lc
      real*4 sig,ai,bi,XXerr,XXerr2
      real*4 t11,t12,t13,t14,t15
      real*8 t1,t2,t3,t4,t5
      real*8 tol,pi,pion2,twopi
      logical raybad
      logical set
      integer*4 set1,set2,set3,set4,dskip,stop,maxit,form
      integer*4 stopc
      integer*4 i,j,k,x,dne,dneq
      integer*4 n,nq,m,mq,nbym
      integer*4 num,num2
      integer*4 numnull,numturn,ucount,iter,numtot
      integer*4 ndat,ndatq
      integer*4 LL,LLn,LLm
      integer*4 nmax,nmaxq,mmax,mmaxq,max,maxq
      integer*4 total
      integer*4 rem,mult,row,col,block,nc
      integer*4 tImage
      integer*4 acnm,fcb,rcb

      parameter(ndatq=4200)
      parameter(nq=90,mq=210)
      parameter(nmaxq=17000,mmaxq=4200,maxq=17000)
      parameter(dneq=4)

      real*4 gS(nmaxq),gC(nmaxq),S0,C0,Itot
      dimension vel(nq,mq)
      dimension chimem(5)
      dimension Image(nq*mq+5)
      dimension raybad(ndatq)
      dimension set(nq*mq)
      dimension length(dneq)
      dimension Am(nmaxq)
      dimension tImage(nq*mq+5)
      dimension ray1travel(nq*mq,6),ray2travel(nq*mq,6)
      dimension Rray(nq*mq)
      dimension D(ndatq),zero(ndatq),F(ndatq),Fold(ndatq)
      dimension sig(ndatq)
      dimension ai(ndatq),bi(ndatq),XXerr(ndatq),XXerr2(ndatq)
      dimension sloc1(ndatq),sloc2(ndatq),tang1(ndatq),tang2(ndatq)
      dimension C(0:nq-1,0:mq-1)
      dimension acnm(2),fcb(20),rcb(20)
      common /velocity/vel
      common /iterat/iter
      common /gausiz/LL,LLn,LLm
      common /backg/Am
      common /constant/tol,pi,pion2,twopi
      common /dist/dlimit
      common /lgxerr/raybad
      common /var/sig,ai,bi,XXerr,XXerr2
      common /nums/numnull,numturn
      common /setvel/set
      common /gaus/C
      common /frechet/fcb,rcb
      common /penalt/length,L,L0
```

```
      common /conparm/n,m,nint,mint,nmax,mmax,max,ndat,dne
      common /mem5cm/gS,gC,C0,S0,Itot
      common /temp/nc
      data acnm/'SCRA','TCH '/
c
c----- read variables from FT21F001
c----- set1-set2 and set3-set4 and 'set' velocity nodes (inclusive)
c----- dskip is the number of data cards to skip before each read
c----- submit -1 for LL and 0 for sets and dskip to turn off
c----- submit 1 for 'stop' to stop inversion after one iteration
c----- 'maxit' is the max iterations in a single stage
c----- 'pctd' defines the percentage drop for chi control
c----- 'form' =0 gives 5g12.6 format for data, =1 gives 5g15.6
c
      read(21,1021) n,m,nint,mint,nmax,mmax,max,ndat,dne
      read(21,1022) LL,set1,set2,set3,set4,dskip,stop,maxit,pctd,form
      write(6,1023) 'n    = ',n
      write(6,1023) 'm    = ',m
      write(6,1024) 'nint = ',nint
      write(6,1024) 'mint = ',mint
      write(6,1023) 'nmax = ',nmax
      write(6,1023) 'mmax = ',mmax
      write(6,1023) 'max  = ',max
      write(6,1023) 'ndat = ',ndat
      write(6,1023) 'dne  = ',dne
      write(6,1023) 'LL   = ',LL
      write(6,1023) 'set1 = ',set1
      write(6,1023) 'set2 = ',set2
      write(6,1023) 'set3 = ',set3
      write(6,1023) 'set4 = ',set4
      write(6,1023) 'dskip= ',dskip
      write(6,1023) 'stop = ',stop
      write(6,1023) 'maxit= ',maxit
      write(6,1024) 'pctd = ',pctd
      write(6,1023) 'form = ',form
      if(n.gt.nq) stop 'n .gt. default maximum'
      if(m.gt.mq) stop 'm .gt. default maximum'
      if(nmax.gt.nmaxq) stop 'nmax .gt. default maximum'
      if(mmax.gt.mmaxq) stop 'mmax .gt. default maximum'
      if(max.gt.maxq) stop 'max .gt. default maximum'
      if(ndat.gt.ndatq) stop 'ndat .gt. default maximum'
      if(dne.gt.dneq) stop 'dne .gt. default maximum'
      if(LL.eq.-1) LL=int(mint)*(m-1)
1021  format(2i5,2f10.2,5i5)
1022  format(8i5,f10.2,i5)
1023  format(2x,a7,i5)
1024  format(2x,a7,f12.2)
c
c----- set constants and initial variables
c
      numtot=0
      nc=0
      stopc=0
      LLn=10*int(LL/nint)+1
      LLm=10*int(LL/mint)+1
      numnull=0
      numturn=0
      iter=0
      ucount=0
      dlimit=1.0
      lc=1.0
      chi0=0.0
      nbym=n*m
      tol=1.0d-10
      pi=4.0d0*datan(1.0d0)
      pion2=2.0d0*datan(1.0d0)
      twopi=2.0d0*pi
      t1=0.0d0
      t2=0.0d0
      t3=0.0d0
      t4=0.0d0
```

```
      t5=0.0d0
      chi2=0.0e0
      entropy=0.0e0
      test=0.0e0
      testold=0.0e0
      total=0
      do 6 i=1,ndat
         raybad(i)=.false.
         sig(i)=1.0
         F(i)=9999.0
6        continue
      do 7 i=1,nbym
         if(((i.ge.set1).and.(i.le.set2)).or.
     +      ((i.ge.set3).and.(i.le.set4))) then
            set(i)=.true.
         else
            set(i)=.false.
            endif
7        continue
c
c----- read the velocity cards
c
      do 610 i=1,(int(n*m/10)-1)*10+1,10
         read(10,'(10i6)') (tImage(k),k=i,i+9)
         write(6,'(10i6)') (tImage(k),k=i,i+9)
610      continue
      read(10,'(10i6)') (tImage(j),j=int(n*m/10)*10+1,n*m,1)
      do 611 i=1,nbym
         Image(i)=float(tImage(i))
611      continue
      do 612 i=1,(int(n*m/10)-1)*10+1,10
         read(12,'(10(1x,i5))') (tImage(k),k=i,i+9)
         write(6,'(10(1x,i5))') (tImage(k),k=i,i+9)
612      continue
      read(12,'(10i6)') (tImage(j),j=int(n*m/10)*10+1,n*m,1)
      do 613 i=1,nbym
         Am(i)=float(tImage(i))
613      continue
      do 100 i=1,n
         do 110 j=1,m
            vel(i,j)=Image((i-1)*m+j)
110         continue
100      continue
c
c----- open the Frechet derivative scratch file
c
      call opn013(fcb,1,acnm,3,0,0,0,4096,i)
      if(i.ne.0) stop 'error opening scratch'
c
c----- open the log files
c
      open(unit=2,file='metlog')
      open(unit=5,file='metfil')
c
c----- read the data times into an array
c----- this file contains the two initial angles, the two surface locations
c----- and the data time
c
      do 50 i=1,ndat
         if(dskip.gt.0) then
            do 51 j=1,dskip
               if(form.eq.0) then
                  read(9,'(g12.6)') t15
               else
                  read(9,'(g15.6)') t15
                  endif
51             continue
            endif
         if(form.eq.0) then
         read(9,'(5g12.6)') sloc1(i),sloc2(i),tang1(i),tang2(i),D(i)
         write(6,'(5g12.6)') sloc1(i),sloc2(i),tang1(i),tang2(i),D(i)
```

```
         else
         read(9,'(5g15.6)') sloc1(i),sloc2(i),tang1(i),tang2(i),D(i)
         write(6,'(5g15.6)') sloc1(i),sloc2(i),tang1(i),tang2(i),D(i)
         endif
         zero(i)=0.0e0
50       continue
      rewind(9)
c
c----- calculate the gaussian smoothing through the model covariance matrix
c----- STAGE LOOPING POINT
c
10101 call gauscalc(LL)
      write(2,'(2x,a25,i6)') 'current half-width (LL) = ',LL
      do 55 i=1,5
         chimem(i)=0.0
55       continue
c
c----- ITERATION LOOPING POINT
c
1     continue
      numnull=0
      numturn=0
      do 3 i=1,ndat
         Fold(i)=F(i)
3        continue
c
c----- begin looping to ray trace for all entries in geom file
c
      do 1000 i=1,ndat
         if(raybad(i)) then
            goto 1000
            endif
         xerr=0.0e0
c
c----- zero the frechet derivative temporary vector Rray
c----- this vector must be zero when each ray is traced
c
         do 61 j=1,nbym
            Rray(j)=0.0e0
61          continue
c
c----- call raytracing subroutine
c
         call xerrrt(sloc1(i),tang1(i),sloc2(i),tang2(i),D(i),xerr)
c
c----- call subroutine to calculate frechet derivatives
c
         if(xerr.ne.0.0) then
            call sade2(i,sloc1(i),tang1(i),sloc2(i),tang2(i),D(i),
     +                  xerr,Rray)
            endif
c
c----- update model xerr in F array
c
         F(i)=xerr
c
c----- zero the Frechet derivatives of any set nodes for this ray
c
      do 305 j=1,nbym
         if(set(j)) Rray(j)=0.0
305     continue
c
c----- write Rray temporary vector out to direct access disc file
c
      j=(i-1)*nbym
      call dmo015(Rray(1),nbym,j,rcb,fcb,1,k)
      if(k.ne.0) stop 'error writing frechet'
c
c----- loop back to do rest of rays
c
1000     continue
```

```
      write(2,'(2x,a14,i10)') 'nulled rays = ',numnull
      write(2,'(2x,a21,i10)') 'nulled turned rays = ',numturn
      write(2,'(2x,a21,i10)') 'number count SADE2 = ',nc
      nc=0
c
c----- compute the weights based on the rays Xerr's
c
      call varian(F)
c
c----- call the ME optimisation subroutine
c
      call MEMNL(nbym,ndat,Image,zero,F,ndat,
     +            nbym,0.0,chi0,1,entropy,chi2,test,1)
c
c----- convert the the new Image vector into the vel matrix for output
c
      do 320 i=1,n
         do 330 j=1,m
            vel(i,j)=Image((i-1)*m+j)
330         continue
320      continue
c
c----- write results out to log file
c
      write(2,'(2x,a16,g16.8)') 'Entropy       = ',entropy
      write(2,'(2x,a16,g16.8)') 'Resulting chi = ',chi2
      write(2,'(2x,a16,g16.8)') 'Test value    = ',test
      write(2,'(2x,a16)') '****************'
      write(5,'(i10,3g16.8)') total,chi2,entropy,test
c
c----- write velocities into log file
c
      do 333 i=1,nbym,10
         write(2,'(10(1x,i5))') (idnint(dble(Image(k))),k=i,i+9)
333      continue
c
c----- stop after one iteration if required
      if(stop.eq.1) goto 2
      stopc=stopc+1
      if(stop.eq.stopc) goto 2
c
c----- update chi memory vector
c
      do 7000 i=1,4
         chimem(i)=chimem(i+1)
7000     continue
      numtot=numtot+numnull
      chimem(5)=chi2
      write(2,'(2x,a9,i5)') 'numtot = ',numtot
c-----
      total=total+1
      if(total.gt.maxit) then
         write(2,'(2x,a37)') '-> more than MAXIT iterations in stage'
         if(test.lt.0.2) then
            goto 77
         else
            goto 2
            endif
         endif
c
c----- check to see if chi0 can be set and if stage is finished
c----- check for oscillations and reduce distance limit if necessary
c----- check for stability and increase distance limit if necessary
c
      j=0
      k=0
      t1=pctd*chimem(1)
      t2=pctd*chimem(1)
      do 7010 i=2,5
         if(chimem(i).lt.t1) j=1
         if(chimem(i).lt.t2) k=1
```

```
7010      continue
      x=0
      do 7015 i=2,5
         if((chimem(i).gt.chimem(i-1)).and.(chimem(i-1).ne.0.0)) x=x+1
7015      continue
      if(chimem(1).eq.0.0) then
         j=1
         k=1
         endif
      if(((test.ge.0.2).and.(test.gt.testold)).or.(x.ge.2).or.
     +    (chimem(5).gt.chimem(4))) then
         if((dlimit.gt.1000.0).and.(test.lt.0.2)) goto 77
         dlimit=dlimit*2.0
         ucount=ucount+1
         chi0=0.0
         do 7016 i=1,5
            chimem(i)=chimem(5)
7016        continue
         chimem(1)=0.0
         testold=test
         iter=iter+1
         goto 1
         endif
      if((k.eq.0).and.(test.lt.0.2)) then
         if((L.gt.0.9*L0).and.(ucount.le.10)) then
            dlimit=dlimit*0.667
            testold=test
            iter=iter+1
            goto 1
         else
            goto 77
            endif
         endif
      if((x.eq.0).and.(L.gt.0.9*L0).and.(ucount.le.10)) then
         dlimit=dlimit*0.8
         testold=test
         iter=iter+1
         goto 1
         endif
c
c----- begin next iteration
c
      testold=test
      iter=iter+1
      goto 1
c
c----- reset smoothing stage
c
77    if(LL.eq.10) goto 2
      total=0
      chi0=0.0
      dlimit=1.0
      testold=0.0
      ucount=0
      LL=LL/2
      if(LL.lt.int(2*mint)) LL=10
      do 7020 i=1,nmax
         Am(i)=Image(i)
7020     continue
      iter=iter+1
      goto 10101
c
c----- else close files and exit
c
2     continue
      close(2)
      close(5)
      close(9)
      close(11)
      stop
      end
```

```
@PROCESS OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS FLAG(E),
@PROCESS DIRECTIVE('*VD:')
      SUBROUTINE CMPICK(MEM,LMEM)
      IMPLICIT INTEGER(A-Y)
      DIMENSION MEM(1)
      CALL PICKER(MEM,MEM,LMEM)
      STOP
      END
@PROCESS OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS FLAG(E),
@PROCESS DIRECTIVE('*VD:')
      SUBROUTINE PICKER (MEM,ZMEM,LMEM)
      IMPLICIT INTEGER (A-Y)
      COMMON /SCRPTR/ IBSCR,NXTSCR,MAXSCR
      DIMENSION ZPICKS(2000,5)
      DIMENSION ZOFF(150)
      DIMENSION FCB(20),RCB(20),TRACES(2)
      DIMENSION FCB2(20),RCB2(20)
      DIMENSION ACNM(2),ACNM2(2),ZARRAY(200*150),ARRAY(2)
      DIMENSION MEM(1),ZMEM(1)
      DATA ATD /'ATD '/,TRACES/'0001','0001'/
      DATA ACNM/'CRPI','N   '/
      DATA ACNM2/'PICK','S1  '/
      DATA GOOD/'GOOD'/,BLNK/'    '/
      NXTSCR=1
      MAXSCR=LMEM
      ZTWOPI=8.0*ATAN(1.0)
      ZPI=4.0*ATAN(1.0)
      ZPION2=2.0*ATAN(1.0)
C
      WRITE(6,80000)
80000 FORMAT('1',///' NPCKCRP2 FOR LINE D DATA      - 09 JAN 1993',/,
     *            ' ------ WITH TRACE HEADER UTILISATION ------',
     *            ' ----------------------------------------')
C
C *** OPEN INPUT/OUTPUT FILE
C
      CALL OPN013(FCB,1,ACNM,2,0,0,0,4096,IST)
      IF(IST.NE.0)GOTO 90000
C
C *** OPEN INPUT PICKS FILE
C
      CALL OPN013(FCB2,1,ACNM2,1,0,0,0,4096,IST)
      IF(IST.NE.0)GOTO 90001
      CALL DMI014(ARRAY(1),2,1,RCB2,FCB2,1,IST)
      MINCRP=ARRAY(1)
      MAXCRP=ARRAY(2)
      WRITE(6,'(1X,A13,2I10)') 'MIN/MAXCRP =',MINCRP,MAXCRP
C
C *** READ OUTPUT FILE AND MAKE SOME CHECKS
C
      IF((NXTSCR+320).GT.MAXSCR)GOTO 90100
      CALL DMI014(MEM(NXTSCR),320,4096,RCB,FCB,1,IST)
      NXTSCR=NXTSCR-1
      IF(IST.NE.0)GOTO 90010
C
C *** CHECK THAT FILE IS ATD/TRACES
C
      IF(MEM(NXTSCR+1).NE.TRACES(1))GOTO 90020
      IF(MEM(NXTSCR+2).NE.TRACES(2))GOTO 90020
C
C *** SET SDMIN, SDMAX, SDINC,
C ***     EDMIN, EDMAX, EDINC, TZERO , TMAX , DELT
C
      SDMIN=MEM(NXTSCR+132)
      NSDMS=MEM(NXTSCR+133)
      SDINC=MEM(NXTSCR+134)
      EDMIN=MEM(NXTSCR+136)
      NEDMS=MEM(NXTSCR+137)
      EDINC=MEM(NXTSCR+138)
```

```
      TZERO=MEM(NXTSCR+152)
      NAMPS=MEM(NXTSCR+153)
      DELT=MEM(NXTSCR+149)
      LTH=MEM(NXTSCR+150)*64
      LTA=MEM(NXTSCR+151)*64
      TRLEN=(LTH+LTA)
      SDMAX=(NSDMS-1)*SDINC+SDMIN
      TMAX=(NAMPS-1)*DELT+TZERO
      LINTV=NSDMS*NEDMS
      SDBHDR=NXTSCR
      NXTSCR=NXTSCR+320
      WRITE(6,'(2X,A10,I10)') 'SDMIN    =',SDMIN
      WRITE(6,'(2X,A10,I10)') 'SDMAX    =',SDMAX
      WRITE(6,'(2X,A10,I10)') 'SDINC    =',SDINC
      WRITE(6,'(2X,A10,I10)') 'EDMIN    =',EDMIN
      WRITE(6,'(2X,A10,I10)') 'NEDMS    =',NEDMS
      WRITE(6,'(2X,A10,I10)') 'EDINC    =',EDINC
      WRITE(6,'(2X,A10,I10)') 'TZERO    =',TZERO
      WRITE(6,'(2X,A10,I10)') 'TMAX     =',TMAX
      WRITE(6,'(2X,A10,I10)') 'DELT     =',DELT
      WRITE(6,'(2X,A10,I10)') 'LTH      =',LTH
      WRITE(6,'(2X,A10,I10)') 'LTA      =',LTA
      ZDELTT=FLOAT(DELT)
C
C *** GET IV'S
C
      IF((NXTSCR+LINTV).GT.MAXSCR) GOTO 90100
      CALL DMI014(MEM(NXTSCR),LINTV,8192,RCB,FCB,1,IST)
      IF(IST.NE.0)GOTO 90030
C
C *** CHECK ENOUGH ROOM
C
      WRITE(6,'(2X,A10)') '          '
      WRITE(6,'(2X,A10,I10)') 'MAXSCR   =',MAXSCR
      WRITE(6,'(2X,A10,I10)') 'NXTSCR   =',NXTSCR
      WRITE(6,'(2X,A10,I10)') 'LINTV    =',LINTV
      WRITE(6,'(2X,A10,I10)') 'NAMPS    =',NAMPS
      IF((NXTSCR+LINTV+3*NAMPS*NEDMS).GT.MAXSCR)
     +     GOTO 90100
      TRSCR=NXTSCR+LINTV+LTH
      IF(MOD(FLOAT(TRSCR),2.0).EQ.0.0) TRSCR=TRSCR+1
      WRITE(6,'(2X,A10,I10)') 'TRSCR    =',TRSCR
C *** MAIN LOOP
C
      SHTEND=TRSCR+NAMPS*NEDMS
      DO 900 I=1,NSDMS,1
         WRITE(6,'(2X,A18,I5)') 'PROCESSING CRP NO.',SDMIN+(I-1)
         DO 905 J=1,NAMPS*NEDMS
            MEM(TRSCR+J-1)=0
905         CONTINUE
C *** READ IN COMPLETE SDOM
         DO 950 J=1,NEDMS,1
C *** GET IV OF NEXT TRACE
            K=NXTSCR+(I-1)*NEDMS+(J-1)
            IV=MEM(K)
            IF(IV.LE.0) THEN
               DO 951 K=1,NAMPS
                  MEM(TRSCR+(J-1)*NAMPS+(K-1))=0.0
951               CONTINUE
                  GOTO 950
               ENDIF
            IV=IV+4096
C *** GET TRACE HEADER ONLY
            CALL DMI014(MEM(TRSCR-LTH),LTH,IV,RCB,FCB,1,IST)
            ZOFF(J)=NINT(ZMEM(TRSCR-LTH+16))
C *** GET TRACE DATA ONLY
            CALL DMI014(MEM(TRSCR+(J-1)*NAMPS),NAMPS,IV+LTH,
     +                  RCB,FCB,1,IST)
            IF(IST.NE.0)GOTO 90050
950         CONTINUE
C
```

```
C----- READ ALL THE PICKS FOR THIS CRP
C
         TMP=200+(SDMIN+(I-1)-MINCRP)*150*200
         CALL DMI014(ZARRAY(1),150*200,TMP,RCB2,FCB2,1,IST)
         IF(IST.NE.0)GOTO 90150
C
C----- LOOP THROUGH EACH OF THE PICKS
C
         NUMPCK=0
         DO 100 L=1,(NEDMS-1)*200,4
            ZXSRC=ZARRAY(L)
            ZXRCV=ZARRAY(L+1)
            ZTIME=ZARRAY(L+2)
            ZANGRC=ZARRAY(L+3)
            IF(ZXSRC.LT.0) GOTO 100
            IF(ABS(ZXSRC-ZXRCV).LT.50.0) GOTO 100
            IF((ZTIME.LE.0.0).OR.(ZTIME.GT.3.0)) GOTO 100
            CRPTR=INT(FLOAT(L-1)/200.0)+1
            NUMPCK=NUMPCK+1
            IF((CRPTR.EQ.1).OR.(CRPTR.EQ.NEDMS)) GOTO 100
c***** COMPUTE PHASE VELOCITY
            K=(CRPTR-1)*NAMPS
            KN=((CRPTR-1)-1)*NAMPS
            KP=((CRPTR+1)-1)*NAMPS
            J=NINT((ZTIME*1000.0)/ZDELTT)
            ZA=ZMEM(TRSCR+KN+J-1)
            ZAP=ZMEM(TRSCR+KN+J-2)
            ZB=ZMEM(TRSCR+KN+J)
            ZC=ZMEM(TRSCR+KN+J+1)
            ZCP=ZMEM(TRSCR+KN+J+2)
            ZD=ZMEM(TRSCR+K+J-1)
            ZE=ZMEM(TRSCR+K+J)
            ZF=ZMEM(TRSCR+K+J+1)
            ZH=ZMEM(TRSCR+KP+J-1)
            ZHP=ZMEM(TRSCR+KP+J-2)
            ZI=ZMEM(TRSCR+KP+J)
            ZJ=ZMEM(TRSCR+KP+J+1)
            ZJP=ZMEM(TRSCR+KP+J+2)
C
C----- CALCULATE THE TIMES OF THE MAXIMA (QUADRATIC INTERP.)
C
            ZT2=ZDELTT*((ZD-ZF)/(2*(ZD+ZF-2*ZE)))+(J-1)*ZDELTT
C
C----- COMPUTE FIRST TRIPLET TIME (TRACE LESS THAN TARGET TRACE)
C----- IF THE ADJACENT TRIPLET DOES NOT SURROUND A MAXIMUM
C----- CHECK THE BEST TRIPLET ONE UP OR ONE DOWN
C
          IF((ZA.GT.ZB).OR.(ZC.GT.ZB)) THEN
             IF(ZA.GT.ZC) THEN
                ZT1=ZDELTT*((ZAP-ZB)/(2*(ZAP+ZB-2*ZA)))+(J-2)*ZDELTT
             ELSE
                ZT1=ZDELTT*((ZB-ZCP)/(2*(ZB+ZCP-2*ZC)))+J*ZDELTT
                ENDIF
          ELSE
             ZT1=ZDELTT*((ZA-ZC)/(2*(ZA+ZC-2*ZB)))+(J-1)*ZDELTT
             ENDIF
C
C----- COMPUTE FIRST TRIPLET TIME (TRACE GREATER THAN TARGET TRACE)
C----- IF THE ADJACENT TRIPLET DOES NOT SURROUND A MAXIMUM
C----- CHECK THE BEST TRIPLET ONE UP OR ONE DOWN
C
          IF((ZH.GT.ZI).OR.(ZJ.GT.ZI)) THEN
             IF(ZH.GT.ZJ) THEN
                ZT3=ZDELTT*((ZHP-ZI)/(2*(ZHP+ZI-2*ZH)))+(J-2)*ZDELTT
             ELSE
                ZT3=ZDELTT*((ZI-ZJP)/(2*(ZI+ZJP-2*ZJ)))+J*ZDELTT
                ENDIF
          ELSE
             ZT3=ZDELTT*((ZH-ZJ)/(2*(ZH+ZJ-2*ZI)))+(J-1)*ZDELTT
             ENDIF
C
```

```
C----- COMPUTE THE TWO DIPS AND COMPARE (COMPUTE AS VELOCITIES)
C
            IF(ABS((ZT3-ZT2)-(ZT2-ZT1)).GT.4.0) THEN
               GOTO 100
               ENDIF
            ZV1=(ZOFF(CRPTR+1)-ZOFF(CRPTR))/(ZT3-ZT2)
            ZV2=(ZOFF(CRPTR)-ZOFF(CRPTR-1))/(ZT2-ZT1)
C
C----- AVERAGE VELOCITIES, CONVERT TO ANGLES AND SAVE
C
            ZANGLE=(ZV1+ZV2)/2.0
            IF(ABS(ZANGLE).LT.1.50) THEN
               GOTO 100
               ENDIF
            ZANGLE=ASIN(1.50/ZANGLE)
            WRITE(5,6000) ZXSRC,ZXRCV,ZANGLE,ZANGRC,ZTIME
100       CONTINUE
      WRITE(6,'(2X,A25,I10)') 'TOTAL PICKS CONSIDERED -',NUMPCK
900   CONTINUE
5000  FORMAT(1X,2G16.8,2I6,2G16.8)
6000  FORMAT(5G15.6)

C
C *** CLOSE INPUT FILE
C
      CALL CLO074(FCB,0,1,IST)
      IF(IST.NE.0)GOTO 90070
      CLOSE(4)
      CLOSE(5)
C
C
      WRITE(6,80010)
80010 FORMAT(' ',//,' NORMAL COMPLETION')
      STOP
C
C
90000 WRITE(6,90005)
90005 FORMAT(' ',//,'    PICKER - ERROR OPENING TRACES FILE',/,
     *            '             ENSURE ACNM OF INPUT FILE IS CRPIN')
      GOTO 99999
C
90001 WRITE(6,90006)
90006 FORMAT(' ',//,'    PICKER - ERROR OPENING PICKS1 FILE',/,
     *            '             ENSURE ACNM OF PICKS FILE IS PICKS1')
      GOTO 99999
C
90010 WRITE(6,90015)
90015 FORMAT(' ',//,'    PICKER -ERROR READING SUBFILE HEADER')
      GOTO 99999
C
90020 WRITE(6,90025)
90025 FORMAT(' ',//,'    PICKER - ERROR INPUT FILE NOT ATD/TRACES')
      GOTO 99999
C
90030 WRITE(6,90035)
90035 FORMAT(' ',//,'    PICKER - ERROR READING INTERVAL VECTORS')
      GOTO 99999
C
90040 WRITE(6,90045)ERRCNT
90045 FORMAT(' ',//,'    PICKER - ERROR OUTPUTTING CRP NO. ',I8)
      GOTO 99999
C
90050 WRITE(6,90055)
90055 FORMAT(' ',//,'    PICKER - ERROR READING TRACE DATA')
      GOTO 99999
C
90070 WRITE(6,90075)
90075 FORMAT(' ',//,'    PICKER - ERROR CLOSING OUTPUT FILE')
      GOTO 99999
C
90100 WRITE(6,90105)
```

```
90105 FORMAT(' ',//,'    PICKER - ERROR - LTP PARAMETER TOO SMALL')
      GOTO 99999
C
90150 WRITE(6,90155)
90155 FORMAT(' ',//,'    PICKER - ERROR READING PICKS1 DATA')
      GOTO 99999
99999 STOP 15
      END
```

```
@PROCESS OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS FLAG(E),
@PROCESS DIRECTIVE('*VD:')
      SUBROUTINE CMPICK(MEM,LMEM)
      IMPLICIT INTEGER(A-Y)
      DIMENSION MEM(1)
      CALL PICKER(MEM,MEM,LMEM)
      STOP
      END
@PROCESS OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS FLAG(E),
@PROCESS DIRECTIVE('*VD:')
      SUBROUTINE PICKER (MEM,ZMEM,LMEM)
      IMPLICIT INTEGER (A-Y)
      COMMON /SCRPTR/ IBSCR,NXTSCR,MAXSCR
      COMMON /CMPPTR/ TRSCR,RADIX,SHTEND,NEDMS,NAMPS,AMPSTR,SPC,ZDELTT,
     +                CRP1,STRT,INC,RECNUM
      COMMON /CRPOFF/ CRP,CRPTR,ZOFF
      COMMON /PICKS1/ FCB2,RCB2,NEXT,MINCRP,MAXCRP,NUMPCK
      DIMENSION FCB(20),RCB(20),TRACES(2)
      DIMENSION FCB2(20),RCB2(20)
      DIMENSION ACNM(2),ACNM2(2)
      DIMENSION MEM(1),ZMEM(1)
      DIMENSION CRP(150),CRPTR(150),ZOFF(150)
      DIMENSION NEXT(2000,150)
      DATA ATD /'ATD '/,TRACES/'0001','0001'/
      DATA ACNM/'INPU','T   '/
      DATA ACNM2/'PICK','S1  '/
      DATA GOOD/'GOOD'/,BLNK/'    '/
      NXTSCR=1
      MAXSCR=LMEM
      MINCRP=0
      MAXCRP=0
      NUMPCK=0
      DO 16 I=1,2000
         DO 17 J=1,150
            NEXT(I,J)=0
17          CONTINUE
16       CONTINUE
C
      WRITE(6,80000)
80000 FORMAT('1',///' NPCKSPC2 FOR LINE D DATA  - 11 DEC 1992',/,
     *             ' ---- WITH TRACE HEADER UTILISATION ----',
     *             ' ---------------------------------------')
C
C *** OPEN INPUT/OUTPUT FILE
C
      CALL OPN013(FCB,1,ACNM,2,0,0,0,4096,IST)
      IF(IST.NE.0)GOTO 90000
C
C *** OPEN FILE TO STORE THE PICKS
C
      CALL OPN013(FCB2,1,ACNM2,4,0,0,0,4096,IST)
      IF(IST.NE.0)GOTO 90000
C
C *** READ OUTPUT FILE AND MAKE SOME CHECKS
C
      IF((NXTSCR+320).GT.MAXSCR)GOTO 90100
      CALL DMI014(MEM(NXTSCR),320,4096,RCB,FCB,1,IST)
      NXTSCR=NXTSCR-1
      IF(IST.NE.0)GOTO 90010
C
C *** CHECK THAT FILE IS ATD/TRACES
C
      IF(MEM(NXTSCR+1).NE.TRACES(1))GOTO 90020
      IF(MEM(NXTSCR+2).NE.TRACES(2))GOTO 90020
C
C *** SET SDMIN, SDMAX, SDINC,
C ***     EDMIN, EDMAX, EDINC, TZERO , TMAX , DELT
C
      SDMIN=MEM(NXTSCR+132)
```

```
      NSDMS=MEM(NXTSCR+133)
      SDINC=MEM(NXTSCR+134)
      EDMIN=MEM(NXTSCR+136)
      NEDMS=MEM(NXTSCR+137)
      EDINC=MEM(NXTSCR+138)
      TZERO=MEM(NXTSCR+152)
      NAMPS=MEM(NXTSCR+153)
      DELT=MEM(NXTSCR+149)
      LTH=MEM(NXTSCR+150)*64
      LTA=MEM(NXTSCR+151)*64
      TRLEN=(LTH+LTA)
      SDMAX=(NSDMS-1)*SDINC+SDMIN
      TMAX=(NAMPS-1)*DELT+TZERO
      LINTV=NSDMS*NEDMS
      SDBHDR=NXTSCR
      NXTSCR=NXTSCR+320
      WRITE(6,'(2X,A10,I10)') 'SDMIN    =',SDMIN
      WRITE(6,'(2X,A10,I10)') 'SDMAX    =',SDMAX
      WRITE(6,'(2X,A10,I10)') 'SDINC    =',SDINC
      WRITE(6,'(2X,A10,I10)') 'EDMIN    =',EDMIN
      WRITE(6,'(2X,A10,I10)') 'NEDMS    =',NEDMS
      WRITE(6,'(2X,A10,I10)') 'EDINC    =',EDINC
      WRITE(6,'(2X,A10,I10)') 'TZERO    =',TZERO
      WRITE(6,'(2X,A10,I10)') 'TMAX     =',TMAX
      WRITE(6,'(2X,A10,I10)') 'DELT     =',DELT
      WRITE(6,'(2X,A10,I10)') 'LTH      =',LTH
      WRITE(6,'(2X,A10,I10)') 'LTA      =',LTA
      ZDELTT=FLOAT(DELT)
C
C *** WRITE ZERO'S TO THE PICKS FILE
C
      CALL ZDA060(FCB2,RCB2,0,(NSDMS*4)*150*200,1,IST)
C
C *** GET IV'S
C
      IF((NXTSCR+LINTV).GT.MAXSCR) GOTO 90100
      CALL DMI014(MEM(NXTSCR),LINTV,8192,RCB,FCB,1,IST)
      IF(IST.NE.0)GOTO 90030
c
c----- COMPUTE THE TEMPORAL RADIX TO USE
c
      RADIX=256
      IF(NAMPS.GT.250) RADIX=512
      IF(NAMPS.GT.500) RADIX=1024
      IF(NAMPS.GT.1000) RADIX=2048
      IF(NAMPS.GT.2000) RADIX=4096
      IF(NAMPS.GT.4000) GOTO 90200
C
C *** CHECK ENOUGH ROOM
C
      WRITE(6,'(2X,A10)') '          '
      WRITE(6,'(2X,A10,I10)') 'MAXSCR   =',MAXSCR
      WRITE(6,'(2X,A10,I10)') 'RADIX    =',RADIX
      WRITE(6,'(2X,A10,I10)') 'NXTSCR   =',NXTSCR
      WRITE(6,'(2X,A10,I10)') 'LINTV    =',LINTV
      WRITE(6,'(2X,A10,I10)') 'NAMPS    =',NAMPS
      IF((NXTSCR+LINTV+2*RADIX*NEDMS+3*NAMPS*NEDMS+80000).GT.MAXSCR)
     +     GOTO 90100
      TRSCR=NXTSCR+LINTV+LTH
      IF(MOD(FLOAT(TRSCR),2.0).EQ.0.0) TRSCR=TRSCR+1
      WRITE(6,'(2X,A10,I10)') 'TRSCR    =',TRSCR
C *** MAIN LOOP
C
      SHTEND=TRSCR+2*RADIX*NEDMS
      DO 900 I=1,NSDMS,1
         WRITE(6,'(2X,A18,I5)') 'PROCESSING SPC NO.',SDMIN+(I-1)
         DO 905 J=1,2*RADIX*NEDMS
            MEM(TRSCR+J-1)=0
905         CONTINUE
C *** READ IN COMPLETE SDOM
         DO 950 J=1,NEDMS,1
```

```
C *** GET IV OF NEXT TRACE
           K=NXTSCR+(I-1)*NEDMS+(J-1)
           IV=MEM(K)
           IF(IV.LE.0) GOTO 950
           IV=IV+4096
C *** GET TRACE HEADER ONLY
           CALL DMI014(MEM(TRSCR-LTH),LTH,IV,RCB,FCB,1,IST)
           CRP(J)=MEM(TRSCR-LTH+40)
           CRPTR(J)=MEM(TRSCR-LTH+41)
           ZOFF(J)=NINT(ZMEM(TRSCR-LTH+16))
c     WRITE(6,'(4I10,G16.8)') I,J,CRP(J),CRPTR(J),ZOFF(J)
C *** GET TRACE DATA ONLY
           CALL DMI014(MEM(TRSCR+RADIX+(J-1)*2*RADIX),LTA,IV+LTH,
     +                RCB,FCB,1,IST)
           IF(IST.NE.0)GOTO 90050
950        CONTINUE
C
C----- IF THIS IS THE FIRST SHOT, DETERMINE MIN CRP
C
       IF(MINCRP.EQ.0) THEN
          IF(CRP(1).LT.CRP(NEDMS)) THEN
             MINCRP=CRP(1)
          ELSE
             MINCRP=CRP(NEDMS)
             ENDIF
          ENDIF
C
C----- ADJUST THE TRACE AMLITUDES TO SIMULATE COMPLEX DATA
C
         DO 960 J=1,NEDMS,1
            DO 970 K=1,RADIX,1
               MEM(TRSCR+(2*(K-1))+(J-1)*2*RADIX)=
     +                     MEM(TRSCR+RADIX+(J-1)*2*RADIX+(K-1))
               ZMEM(TRSCR+RADIX+(J-1)*2*RADIX+(K-1))=0.0
               ZMEM(TRSCR+(2*(K-1)+1)+(J-1)*RADIX)=0.0
970            CONTINUE
960         CONTINUE
C
C----- CALL SUBROUTINE TO DO COMPLEX WORK
C
         SPC=SDMIN+I-1
         CALL CMPLEX(MEM,MEM,MEM,LMEM)
C
C *** WRITE OUT THE INSTANTANEOUS PHASE SHOT RECORD
         ERRCNT=SDMIN+(I-1)*SDINC
         DO 980 J=1,NEDMS,1
C *** GET IV OF NEXT TRACE
           K=NXTSCR+(I-1)*NEDMS+(J-1)
           IV=MEM(K)
           IF(IV.LE.0) GOTO 980
           IV=IV+4096
C *** WRITE TRACE DATA ONLY
           CALL DMO015(MEM(AMPSTR+(J-1)*RADIX),NAMPS,IV+LTH,
     +                RCB,FCB,1,IST)
           IF(IST.NE.0)GOTO 90040
980        CONTINUE
C
900   CONTINUE
C
C *** CLOSE OUTPUT FILE
C
      CALL CLO074(FCB,0,1,IST)
      IF(IST.NE.0)GOTO 90070
      CLOSE(4)
C
C
      WRITE(6,80010)
80010 FORMAT(' ',//,' NORMAL COMPLETION')
      STOP
C
C
```

```
90000 WRITE(6,90005)
90005 FORMAT(' ',//,'    PICKER - ERROR OPENING TRACES FILE',/,
     *             '             ENSURE ACNM OF INPUT FILE IS INPUT')
      GOTO 99999
C
90001 WRITE(6,90006)
90006 FORMAT(' ',//,'    PICKER - ERROR OPENING PICKS1 FILE',/,
     *             '             ENSURE ACNM OF SCRATCH FILE IS PICKS1')
      GOTO 99999
C
90010 WRITE(6,90015)
90015 FORMAT(' ',//,'    PICKER -ERROR READING SUBFILE HEADER')
      GOTO 99999
C
90020 WRITE(6,90025)
90025 FORMAT(' ',//,'    PICKER - ERROR INPUT FILE NOT ATD/TRACES')
      GOTO 99999
C
90030 WRITE(6,90035)
90035 FORMAT(' ',//,'    PICKER - ERROR READING INTERVAL VECTORS')
      GOTO 99999
C
90040 WRITE(6,90045)ERRCNT
90045 FORMAT(' ',//,'    PICKER - ERROR OUTPUTTING SPC NO. ',I8)
      GOTO 99999
C
90050 WRITE(6,90055)
90055 FORMAT(' ',//,'    PICKER - ERROR READING TRACE DATA')
      GOTO 99999
C
90070 WRITE(6,90075)
90075 FORMAT(' ',//,'    PICKER - ERROR CLOSING OUTPUT FILE')
      GOTO 99999
C
90100 WRITE(6,90105)
90105 FORMAT(' ',//,'    PICKER - ERROR - LTP PARAMETER TOO SMALL')
      GOTO 99999
C
90200 WRITE(6,90205)
90205 FORMAT(' ',//,'    PICKER - MORE THAN 4000 SAMPLES/TRACE   ')
      GOTO 99999
C
90300 WRITE(6,90305)
90305 FORMAT(' ',//,'    PICKER - MORE THAN 490 TRACES/SDOM      ')
      GOTO 99999
C
C
99999 STOP 15
      END
@PROCESS OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS FLAG(E),
@PROCESS DIRECTIVE('*VD:')
      SUBROUTINE CMPLEX (MEM,YMEM,ZMEM,LMEM)
      IMPLICIT INTEGER (A-X)
      IMPLICIT COMPLEX*8 (Y)
      COMMON /SCRPTR/ IBSCR,NXTSCR,MAXSCR
      COMMON /CMPPTR/ TRSCR,RADIX,SHTEND,NEDMS,NAMPS,AMPSTR,SPC,ZDELTT,
     +                CRP1,STRT,INC,RECNUM
      COMMON /CRPOFF/ CRP,CRPTR,ZOFF
      COMMON /PICKS1/ FCB2,RCB2,NEXT,MINCRP,MAXCRP,NUMPCK
      DIMENSION FCB2(20),RCB2(20)
      DIMENSION MEM(1),YMEM(1),ZMEM(1)
      DIMENSION ZPICKS(150,50,6)
      DIMENSION CRP(150),CRPTR(150),ZOFF(150)
      DIMENSION NEXT(2000,150),ZARRAY(4),ARRAY(2)

      NUMPCK=0
      ZTWOPI=8.0*ATAN(1.0)
      ZPI=4.0*ATAN(1.0)
      ZPION2=2.0*ATAN(1.0)
      ZPION8=0.5*ATAN(1.0)
```

```
c
c----- PERFORM FORWARD FOURIER TRANSFORMS
c
      TRSCR2=(TRSCR+1)/2
      SHTEND2=SHTEND/2+2
      IF((SHTEND+120000).GT.MAXSCR) GOTO 90100
      CALL SCFT(1,YMEM(TRSCR2),1,RADIX,YMEM(TRSCR2),1,RADIX,
     +          RADIX,NEDMS,1,1.0,YMEM(SHTEND2),20000,
     +          YMEM(SHTEND2+20000),20000)
      CALL SCFT(0,YMEM(TRSCR2),1,RADIX,YMEM(TRSCR2),1,RADIX,
     +          RADIX,NEDMS,1,1.0,YMEM(SHTEND2),20000,
     +          YMEM(SHTEND2+20000),20000)
C
C----- ZERO THE NEGATIVE FREQUENCIES
C
      DO 100 I=1,NEDMS,1
         ZA=TRSCR2+(I-1)*RADIX
         DO 200 J=RADIX/2+1,RADIX
            YMEM(ZA+J)=CMPLX(0.0,0.0)
200         CONTINUE
         YMEM(ZA)=YMEM(ZA)/2.0
100      CONTINUE
c
c----- PERFORM INVERSE 2D FOURIER TRANSFORM
c
      CALL SCFT(1,YMEM(TRSCR2),1,RADIX,YMEM(TRSCR2),1,RADIX,
     +     RADIX,NEDMS,-1,1.0/FLOAT(RADIX),YMEM(SHTEND2),20000,
     +          YMEM(SHTEND2+20000),20000)
      CALL SCFT(0,YMEM(TRSCR2),1,RADIX,YMEM(TRSCR2),1,RADIX,
     +     RADIX,NEDMS,-1,1.0/FLOAT(RADIX),YMEM(SHTEND2),20000,
     +          YMEM(SHTEND2+20000),20000)
c
c----- COMPUTE INSTANTANEOUS AMPLITUDE
c
      AMPSTR=SHTEND+80100
      IF((AMPSTR+NEDMS*RADIX).GT.MAXSCR) GOTO 90100
      Z1=0.0
      DO 300 I=1,NEDMS
         K=(I-1)*RADIX
         DO 400 J=0,NAMPS-1
            ZMEM(AMPSTR+K+J)=SQRT(REAL(YMEM(TRSCR2+K+J))**2+
     +                            IMAG(YMEM(TRSCR2+K+J))**2)
            IF(ZMEM(AMPSTR+K+J).GT.Z1) Z1=ZMEM(AMPSTR+K+J)
400         CONTINUE
300      CONTINUE
c
c----- FIND MAXIMUM AMPLITUDE POINTS
c
      VELSTR=AMPSTR+NEDMS*RADIX
      IF((VELSTR+NEDMS*RADIX).GT.MAXSCR) GOTO 90100
      DO 700 I=2,NEDMS-1,1
         K=(I-1)*RADIX
         KN=((I-1)-1)*RADIX
         KP=((I+1)-1)*RADIX
         DO 800 J=1,NAMPS-2,1
            IF(ZMEM(AMPSTR+K+J).LT.0.01*Z1) THEN
               ZMEM(VELSTR+K+J)=0.0
               GOTO 800
               ENDIF
            IF((ZMEM(AMPSTR+K+J).LT.ZMEM(AMPSTR+K+J-1)).OR.
     +          (ZMEM(AMPSTR+K+J).LT.ZMEM(AMPSTR+K+J+1))) THEN
               ZMEM(VELSTR+K+J)=0.0
               GOTO 800
               ENDIF
C----- PICK LARGEST AMPLITUDE WITHIN 200 MSEC ONLY
            DO 850 JJ=-12,-1,1
               IF((J+JJ).LT.0) GOTO 850
               IF((ZMEM(VELSTR+K+J+JJ).NE.0.0).AND.
     +             (ZMEM(AMPSTR+K+J+JJ).LE.ZMEM(AMPSTR+K+J))) THEN
                  ZMEM(VELSTR+K+J+JJ)=0.0
               ELSEIF((ZMEM(VELSTR+K+J+JJ).NE.0.0).AND.
```

```
     +            (ZMEM(AMPSTR+K+J+JJ).GT.ZMEM(AMPSTR+K+J))) THEN
                  ZMEM(VELSTR+K+J)=0.0
                  GOTO 800
                  ENDIF
850               CONTINUE
               ZA=ZMEM(AMPSTR+KN+J-1)
               ZAP=ZMEM(AMPSTR+KN+J-2)
               ZB=ZMEM(AMPSTR+KN+J)
               ZC=ZMEM(AMPSTR+KN+J+1)
               ZCP=ZMEM(AMPSTR+KN+J+2)
               ZD=ZMEM(AMPSTR+K+J-1)
               ZE=ZMEM(AMPSTR+K+J)
               ZF=ZMEM(AMPSTR+K+J+1)
               ZH=ZMEM(AMPSTR+KP+J-1)
               ZHP=ZMEM(AMPSTR+KP+J-2)
               ZI=ZMEM(AMPSTR+KP+J)
               ZJ=ZMEM(AMPSTR+KP+J+1)
               ZJP=ZMEM(AMPSTR+KP+J+2)
C
C----- CALCULATE THE TIMES OF THE MAXIMA (QUADRATIC INTERP.)
C----- FOR THE CENTRAL TARGET TRACE
C
               ZT2=ZDELTT*((ZD-ZF)/(2*(ZD+ZF-2*ZE)))+(J-1)*ZDELTT
C
C----- COMPUTE FIRST TRIPLET TIME (TRACE LESS THAN TARGET TRACE)
C----- IF THE ADJACENT TRIPLET DOES NOT SURROUND A MAXIMUM
C----- CHECK THE BEST TRIPLET ONE UP OR ONE DOWN
C
              IF((ZA.GT.ZB).OR.(ZC.GT.ZB)) THEN
                 IF(ZA.GT.ZC) THEN
                    ZT1=ZDELTT*((ZAP-ZB)/(2*(ZAP+ZB-2*ZA)))+(J-2)*ZDELTT
                 ELSE
                    ZT1=ZDELTT*((ZB-ZCP)/(2*(ZB+ZCP-2*ZC)))+J*ZDELTT
                    ENDIF
              ELSE
                 ZT1=ZDELTT*((ZA-ZC)/(2*(ZA+ZC-2*ZB)))+(J-1)*ZDELTT
                 ENDIF
C
C----- COMPUTE FIRST TRIPLET TIME (TRACE GREATER THAN TARGET TRACE)
C----- IF THE ADJACENT TRIPLET DOES NOT SURROUND A MAXIMUM
C----- CHECK THE BEST TRIPLET ONE UP OR ONE DOWN
C
              IF((ZH.GT.ZI).OR.(ZJ.GT.ZI)) THEN
                 IF(ZH.GT.ZJ) THEN
                    ZT3=ZDELTT*((ZHP-ZI)/(2*(ZHP+ZI-2*ZH)))+(J-2)*ZDELTT
                 ELSE
                    ZT3=ZDELTT*((ZI-ZJP)/(2*(ZI+ZJP-2*ZJ)))+J*ZDELTT
                    ENDIF
              ELSE
                 ZT3=ZDELTT*((ZH-ZJ)/(2*(ZH+ZJ-2*ZI)))+(J-1)*ZDELTT
                 ENDIF
C
C----- COMPARE THE TIMES OF THE MAXIMA
C
               IF((ABS(ZT1-ZT2).GT.14.0).OR.(ABS(ZT2-ZT3).GT.14.0)) THEN
                  ZMEM(VELSTR+K+J)=0.0
                  GOTO 800
                  ENDIF
C
C----- COMPUTE THE TWO DIPS AND COMPARE (COMPUTE AS VELOCITIES)
C
               IF(ABS((ZT3-ZT2)-(ZT2-ZT1)).GT.4.0) THEN
                  ZMEM(VELSTR+K+J)=0.0
                  GOTO 800
                  ENDIF
               ZV1=(ZOFF(I+1)-ZOFF(I))/(ZT3-ZT2)
               ZV2=(ZOFF(I)-ZOFF(I-1))/(ZT2-ZT1)
C
C----- AVERAGE VELOCITIES, CONVERT TO ANGLES AND SAVE
C
               ZMEM(VELSTR+K+J)=(ZV1+ZV2)/2.0
```

```
            IF(ABS(ZMEM(VELSTR+K+J)).LT.1.50) THEN
               ZMEM(VELSTR+K+J)=0.0
               GOTO 800
               ENDIF
            ZARRAY(4)=ASIN(1.50/ZMEM(VELSTR+K+J))
            ZARRAY(1)=(401-SPC)*25.0
            ZARRAY(3)=J*0.004
            ZARRAY(2)=ZARRAY(1)+ZOFF(I)
            IF(ZXRCV.GT.10000.0) GOTO 800
C
            TMP=NEXT(CRP(I)-MINCRP,CRPTR(I))
            IF(TMP.GT.196) THEN
C              WRITE(6,'(1X,A22)') 'MAX PICKS/TR EXCEEDED'
               GOTO 800
               END IF
            TMP=TMP+200+(CRP(I)-MINCRP)*150*200+(CRPTR(I)-1)*200
            CALL DMO015(ZARRAY(1),4,TMP,RCB2,FCB2,1,IST)
            IF(IST.NE.0) GOTO 90110
           NEXT(CRP(I)-MINCRP,CRPTR(I))=NEXT(CRP(I)-MINCRP,CRPTR(I))+4
            IF(CRP(I).GT.MAXCRP) MAXCRP=CRP(I)
            NUMPCK=NUMPCK+1
800         CONTINUE
700       CONTINUE

      ARRAY(1)=MINCRP
      ARRAY(2)=MAXCRP
      WRITE(6,'(2X,A12,I10)') 'NUMPCK     =',NUMPCK
      WRITE(6,'(2X,A12,2I10)') 'MIN/MAXCRP =',MINCRP,MAXCRP
      CALL DMO015(ARRAY(1),2,1,RCB2,FCB2,1,IST)
      IF(IST.NE.0) GOTO 90120

      RETURN
90100 WRITE(6,90105)
90105 FORMAT(' ',//,'    CMPLEX - ERROR - LTP PARAMETER TOO SMALL')
      GOTO 99999
C
90101 WRITE(6,90106)
90106 FORMAT(' ',//,'    CMPLEX - ERROR - TOO MANY PICKS ON TRACE')
      GOTO 99999
C
90110 WRITE(6,90115)
90115 FORMAT(' ',//,'    CMPLEX - ERROR - WRITING TO PICKS1      ')
      GOTO 99999
C
90120 WRITE(6,90125)
90125 FORMAT(' ',//,'    CMPLEX - ERROR - WRITING PICKS1 HEADER ')
      GOTO 99999
C
C
99999 STOP 15
      END
```

```
@PROCESS OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS FLAG(E),
@PROCESS DIRECTIVE('*VD:')
      SUBROUTINE CMPICK(MEM,LMEM)
      IMPLICIT INTEGER(A-Y)
      DIMENSION MEM(1)
      CALL PICKER(MEM,MEM,LMEM)
      STOP
      END
@PROCESS OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS FLAG(E),
@PROCESS DIRECTIVE('*VD:')
      SUBROUTINE PICKER (MEM,ZMEM,LMEM)
      IMPLICIT INTEGER (A-Y)
      COMMON /SCRPTR/ IBSCR,NXTSCR,MAXSCR
      DIMENSION ZPICKS(2000,5)
      DIMENSION ZOFF(150)
      DIMENSION FCB(20),RCB(20),TRACES(2)
      DIMENSION ACNM(2)
      DIMENSION MEM(1),ZMEM(1)
      DATA ATD /'ATD '/,TRACES/'0001','0001'/
      DATA ACNM/'CRPI','N   '/
      DATA GOOD/'GOOD'/,BLNK/'    '/
      NXTSCR=1
      MAXSCR=LMEM
      ZTWOPI=8.0*ATAN(1.0)
      ZPI=4.0*ATAN(1.0)
      ZPION2=2.0*ATAN(1.0)
C
      WRITE(6,80000)
80000 FORMAT('1',//' PICKCRP5 FOR LINE C DATA      - 15 AUG 1992',/,
     *             ' ------ WITH TRACE HEADER UTILISATION ------',
     *             ' ----------------------------------------')
C
C *** OPEN INPUT/OUTPUT FILE
C
      CALL OPN013(FCB,1,ACNM,2,0,0,0,4096,IST)
      IF(IST.NE.0)GOTO 90000
C
C *** READ OUTPUT FILE AND MAKE SOME CHECKS
C
      IF((NXTSCR+320).GT.MAXSCR)GOTO 90100
      CALL DMI014(MEM(NXTSCR),320,4096,RCB,FCB,1,IST)
      NXTSCR=NXTSCR-1
      IF(IST.NE.0)GOTO 90010
C
C *** CHECK THAT FILE IS ATD/TRACES
C
      IF(MEM(NXTSCR+1).NE.TRACES(1))GOTO 90020
      IF(MEM(NXTSCR+2).NE.TRACES(2))GOTO 90020
C
C *** SET SDMIN, SDMAX, SDINC,
C ***     EDMIN, EDMAX, EDINC, TZERO , TMAX , DELT
C
      SDMIN=MEM(NXTSCR+132)
      NSDMS=MEM(NXTSCR+133)
      SDINC=MEM(NXTSCR+134)
      EDMIN=MEM(NXTSCR+136)
      NEDMS=MEM(NXTSCR+137)
      EDINC=MEM(NXTSCR+138)
      TZERO=MEM(NXTSCR+152)
      NAMPS=MEM(NXTSCR+153)
      DELT=MEM(NXTSCR+149)
      LTH=MEM(NXTSCR+150)*64
      LTA=MEM(NXTSCR+151)*64
      TRLEN=(LTH+LTA)
      SDMAX=(NSDMS-1)*SDINC+SDMIN
      TMAX=(NAMPS-1)*DELT+TZERO
      LINTV=NSDMS*NEDMS
      SDBHDR=NXTSCR
      NXTSCR=NXTSCR+320
```

```
      WRITE(6,'(2X,A10,I10)') 'SDMIN    =',SDMIN
      WRITE(6,'(2X,A10,I10)') 'SDMAX    =',SDMAX
      WRITE(6,'(2X,A10,I10)') 'SDINC    =',SDINC
      WRITE(6,'(2X,A10,I10)') 'EDMIN    =',EDMIN
      WRITE(6,'(2X,A10,I10)') 'NEDMS    =',NEDMS
      WRITE(6,'(2X,A10,I10)') 'EDINC    =',EDINC
      WRITE(6,'(2X,A10,I10)') 'TZERO    =',TZERO
      WRITE(6,'(2X,A10,I10)') 'TMAX     =',TMAX
      WRITE(6,'(2X,A10,I10)') 'DELT     =',DELT
      WRITE(6,'(2X,A10,I10)') 'LTH      =',LTH
      WRITE(6,'(2X,A10,I10)') 'LTA      =',LTA
      ZDELTT=FLOAT(DELT)
C
C *** GET IV'S
C
      IF((NXTSCR+LINTV).GT.MAXSCR) GOTO 90100
      CALL DMI014(MEM(NXTSCR),LINTV,8192,RCB,FCB,1,IST)
      IF(IST.NE.0)GOTO 90030
C
C *** CHECK ENOUGH ROOM
C
      WRITE(6,'(2X,A10)') '          '
      WRITE(6,'(2X,A10,I10)') 'MAXSCR   =',MAXSCR
      WRITE(6,'(2X,A10,I10)') 'NXTSCR   =',NXTSCR
      WRITE(6,'(2X,A10,I10)') 'LINTV    =',LINTV
      WRITE(6,'(2X,A10,I10)') 'NAMPS    =',NAMPS
      IF((NXTSCR+LINTV+3*NAMPS*NEDMS).GT.MAXSCR)
     +     GOTO 90100
      TRSCR=NXTSCR+LINTV+LTH
      IF(MOD(FLOAT(TRSCR),2.0).EQ.0.0) TRSCR=TRSCR+1
      WRITE(6,'(2X,A10,I10)') 'TRSCR    =',TRSCR
C *** MAIN LOOP
C
      SHTEND=TRSCR+NAMPS*NEDMS
      DO 900 I=1,NSDMS,1
          DO 1200 J=1,2000
             ZPICKS(J,5)=0.0
1200         CONTINUE
         WRITE(6,'(2X,A18,I5)') 'PROCESSING CRP NO.',SDMIN+(I-1)
         DO 905 J=1,NAMPS*NEDMS
            MEM(TRSCR+J-1)=0
905         CONTINUE
C *** READ IN COMPLETE SDOM
         DO 950 J=1,NEDMS,1
C *** GET IV OF NEXT TRACE
            K=NXTSCR+(I-1)*NEDMS+(J-1)
            IV=MEM(K)
            IF(IV.LE.0) THEN
               DO 951 K=1,NAMPS
                  MEM(TRSCR+(J-1)*NAMPS+(K-1))=0.0
951               CONTINUE
                  GOTO 950
               ENDIF
            IV=IV+4096
C *** GET TRACE HEADER ONLY
            CALL DMI014(MEM(TRSCR-LTH),LTH,IV,RCB,FCB,1,IST)
            ZOFF(J)=NINT(ZMEM(TRSCR-LTH+16))
C *** GET TRACE DATA ONLY
            CALL DMI014(MEM(TRSCR+(J-1)*NAMPS),NAMPS,IV+LTH,
     +                  RCB,FCB,1,IST)
            IF(IST.NE.0)GOTO 90050
950         CONTINUE
C
C----- LOOK THOUGH PICKSPC FILE FOR PICKS WITHIN THIS CRP
C
         COUNT=1
         REWIND(4)
100      READ(4,5000,END=1000) ZXSRC,ZTIME,CRP,CRPTR,ZXRCV,ZANGRC
         IF(CRP.EQ.SDMIN+(I-1)) THEN
            IF((CRPTR.EQ.1).OR.(CRPTR.EQ.NEDMS)) GOTO 100
c***** COMPUTE PHASE VELOCITY
```

```
            K=(CRPTR-1)*NAMPS
            KN=((CRPTR-1)-1)*NAMPS
            KP=((CRPTR+1)-1)*NAMPS
            J=INT((ZTIME*1000.0)/ZDELTT)+1
c         WRITE(6,'(2X,A3,1X,I10)') '-J-',J
            ZA=ZMEM(TRSTR+K+J+1)
            ZM=ZMEM(TRSCR+K+J)
            ZB=ZMEM(TRSCR+K+J-1)
            ZC=ZMEM(TRSCR+KP+J)
            ZD=ZMEM(TRSCR+KN+J)
C----- CALCULATE DIFFERENCES
            CALL DIFF(ZA,ZM,ZU1)
            CALL DIFF(ZM,ZB,ZU2)
            CALL DIFF(ZC,ZM,ZL1)
            CALL DIFF(ZM,ZD,ZL2)
C----- IF TWO HALF DIFFERENCES ARE VERY DIFFERENT, IGNORE POINT
            ZT=ZU1/ZU2
            IF((ZT.LT.0.25).OR.(ZT.GT.4.00)) GOTO 100
            ZT=ZL1/ZL2
            IF((ZT.LT.0.25).OR.(ZT.GT.4.00)) GOTO 100
C----- COMPUTE AVERAGE DIFFERENCES
            ZU=(ZU1+ZU2)/2.0
            ZL=(ZL1+ZL2)/2.0
C----- CHECK TO SEE IF DIP IS NEGATIVE AND CORRECT IF SO
            IF(ZL.GT.ZPI) ZL=ZL-ZTWOPI
            ZPVEL=ZU/ZL
            ZDELTX=ABS(ZOFF(CRPTR+1)-ZOFF(CRPTR-1))/2.0
            ZPVEL=-1.0*ZPVEL*ZDELTX/ZDELTT
C***** CONVERT PHASE VELOCITIES TO ANGLES AND WRITE TO OUTPUT FILE
            IF(ABS(ZPVEL).LT.1.48) THEN
               GOTO 100
               ENDIF
            ZANGSC=ASIN(1.48/ZPVEL)
            ZPICKS(COUNT,1)=ZXSRC
            ZPICKS(COUNT,2)=ZXRCV
            ZPICKS(COUNT,3)=ZANGSC
            ZPICKS(COUNT,4)=ZANGRC
            ZPICKS(COUNT,5)=ZTIME
            COUNT=COUNT+1
            ENDIF
            GOTO 100
C
C----- OUTPUT VALUES
C
1000      DO 1110 J=1,2000
            ZTIME=ZPICKS(J,5)
            IF(ZTIME.EQ.0.0) GOTO 900
            WRITE(5,6000) ZPICKS(J,1),ZPICKS(J,2),
     +                    ZPICKS(J,3),ZPICKS(J,4),ZPICKS(J,5)
1110        CONTINUE
C
900   CONTINUE
5000   FORMAT(1X,2G16.8,2I6,2G16.8)
6000   FORMAT(5G15.6)
C
C *** CLOSE INPUT FILE
C
      CALL CLO074(FCB,0,1,IST)
      IF(IST.NE.0)GOTO 90070
      CLOSE(4)
      CLOSE(5)
C
C
      WRITE(6,80010)
80010 FORMAT(' ',//,' NORMAL COMPLETION')
      STOP
C
C
90000 WRITE(6,90005)
90005 FORMAT(' ',//,'    PICKER - ERROR OPENING TRACES FILE',/,
     *             '             ENSURE ACNM OF INPUT FILE IS CRPIN')
```

```
      GOTO 99999
C
90010 WRITE(6,90015)
90015 FORMAT(' ',//,'    PICKER -ERROR READING SUBFILE HEADER')
      GOTO 99999
C
90020 WRITE(6,90025)
90025 FORMAT(' ',//,'    PICKER - ERROR INPUT FILE NOT ATD/TRACES')
      GOTO 99999
C
90030 WRITE(6,90035)
90035 FORMAT(' ',//,'    PICKER - ERROR READING INTERVAL VECTORS')
      GOTO 99999
C
90040 WRITE(6,90045)ERRCNT
90045 FORMAT(' ',//,'    PICKER - ERROR OUTPUTTING CRP NO. ',I8)
      GOTO 99999
C
90050 WRITE(6,90055)
90055 FORMAT(' ',//,'    PICKER - ERROR READING TRACE DATA')
      GOTO 99999
C
90070 WRITE(6,90075)
90075 FORMAT(' ',//,'    PICKER - ERROR CLOSING OUTPUT FILE')
      GOTO 99999
C
90100 WRITE(6,90105)
90105 FORMAT(' ',//,'    PICKER - ERROR - LTP PARAMETER TOO SMALL')
      GOTO 99999
C
99999 STOP 15
      END
      SUBROUTINE DIFF(ZA,ZM,ZU1)
      REAL ZA,ZM,ZU1,ZTWOPI
      ZTWOPI=8.0*ATAN(1.0)
      IF(((ZA.GE.0.0).AND.(ZM.GE.0.0)).OR.
     +   ((ZA.LT.0.0).AND.(ZM.LT.0.0))) THEN
           IF(ZA.GE.ZM) THEN
              ZU1=ZA-ZM
           ELSE
              ZU1=ZTWOPI-(ZM-ZA)
              ENDIF
           ENDIF
      IF((ZA.GE.0.0).AND.(ZM.LT.0.0)) ZU1=ZA-ZM
      IF((ZA.LT.0.0).AND.(ZM.GE.0.0)) ZU1=ZTWOPI-(ZM-ZA)
      RETURN
      END
```

```
@PROCESS OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS FLAG(E),
@PROCESS DIRECTIVE('*VD:')
      SUBROUTINE CMPICK(MEM,LMEM)
      IMPLICIT INTEGER(A-Y)
      DIMENSION MEM(1)
      CALL PICKER(MEM,MEM,LMEM)
      STOP
      END
@PROCESS OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS FLAG(E),
@PROCESS DIRECTIVE('*VD:')
      SUBROUTINE PICKER (MEM,ZMEM,LMEM)
      IMPLICIT INTEGER (A-Y)
      COMMON /SCRPTR/ IBSCR,NXTSCR,MAXSCR
      COMMON /CMPPTR/ TRSCR,RADIX,SHTEND,NEDMS,NAMPS,PHSSTR,SPC,ZDELTT
      COMMON /CRPOFF/ CRP,CRPTR,ZOFF
      DIMENSION FCB(20),RCB(20),TRACES(2)
      DIMENSION ACNM(2)
      DIMENSION MEM(1),ZMEM(1)
      DIMENSION CRP(150),CRPTR(150),ZOFF(150)
      DATA ATD /'ATD '/,TRACES/'0001','0001'/
      DATA ACNM/'INPU','T   '/
      DATA GOOD/'GOOD'/,BLNK/'    '/
      NXTSCR=1
      MAXSCR=LMEM
C
      WRITE(6,80000)
80000 FORMAT('1',//' PICKSPC5 FOR LINE C DATA  - 15 AUG 1992',/,
     *            ' ---- WITH TRACE HEADER UTILISATION ----',
     *            ' ---------------------------------------')
C
C *** OPEN INPUT/OUTPUT FILE
C
      CALL OPN013(FCB,1,ACNM,2,0,0,0,4096,IST)
      IF(IST.NE.0)GOTO 90000
C
C *** READ OUTPUT FILE AND MAKE SOME CHECKS
C
      IF((NXTSCR+320).GT.MAXSCR)GOTO 90100
      CALL DMI014(MEM(NXTSCR),320,4096,RCB,FCB,1,IST)
      NXTSCR=NXTSCR-1
      IF(IST.NE.0)GOTO 90010
C
C *** CHECK THAT FILE IS ATD/TRACES
C
      IF(MEM(NXTSCR+1).NE.TRACES(1))GOTO 90020
      IF(MEM(NXTSCR+2).NE.TRACES(2))GOTO 90020
C
C *** SET SDMIN, SDMAX, SDINC,
C ***     EDMIN, EDMAX, EDINC, TZERO , TMAX , DELT
C
      SDMIN=MEM(NXTSCR+132)
      NSDMS=MEM(NXTSCR+133)
      SDINC=MEM(NXTSCR+134)
      EDMIN=MEM(NXTSCR+136)
      NEDMS=MEM(NXTSCR+137)
      EDINC=MEM(NXTSCR+138)
      TZERO=MEM(NXTSCR+152)
      NAMPS=MEM(NXTSCR+153)
      DELT=MEM(NXTSCR+149)
      LTH=MEM(NXTSCR+150)*64
      LTA=MEM(NXTSCR+151)*64
      TRLEN=(LTH+LTA)
      SDMAX=(NSDMS-1)*SDINC+SDMIN
      TMAX=(NAMPS-1)*DELT+TZERO
      LINTV=NSDMS*NEDMS
      SDBHDR=NXTSCR
      NXTSCR=NXTSCR+320
      WRITE(6,'(2X,A10,I10)') 'SDMIN    =',SDMIN
      WRITE(6,'(2X,A10,I10)') 'SDMAX    =',SDMAX
```

```
      WRITE(6,'(2X,A10,I10)') 'SDINC    =',SDINC
      WRITE(6,'(2X,A10,I10)') 'EDMIN    =',EDMIN
      WRITE(6,'(2X,A10,I10)') 'NEDMS    =',NEDMS
      WRITE(6,'(2X,A10,I10)') 'EDINC    =',EDINC
      WRITE(6,'(2X,A10,I10)') 'TZERO    =',TZERO
      WRITE(6,'(2X,A10,I10)') 'TMAX     =',TMAX
      WRITE(6,'(2X,A10,I10)') 'DELT     =',DELT
      WRITE(6,'(2X,A10,I10)') 'LTH      =',LTH
      WRITE(6,'(2X,A10,I10)') 'LTA      =',LTA
      ZDELTT=FLOAT(DELT)
C
C *** GET IV'S
C
      IF((NXTSCR+LINTV).GT.MAXSCR) GOTO 90100
      CALL DMI014(MEM(NXTSCR),LINTV,8192,RCB,FCB,1,IST)
      IF(IST.NE.0)GOTO 90030
c
c----- COMPUTE THE TEMPORAL RADIX TO USE
c
      RADIX=256
      IF(NAMPS.GT.250) RADIX=512
      IF(NAMPS.GT.500) RADIX=1024
      IF(NAMPS.GT.1000) RADIX=2048
      IF(NAMPS.GT.2000) RADIX=4096
      IF(NAMPS.GT.4000) GOTO 90200
C
C *** CHECK ENOUGH ROOM
C
      WRITE(6,'(2X,A10)') '          '
      WRITE(6,'(2X,A10,I10)') 'MAXSCR   =',MAXSCR
      WRITE(6,'(2X,A10,I10)') 'RADIX    =',RADIX
      WRITE(6,'(2X,A10,I10)') 'NXTSCR   =',NXTSCR
      WRITE(6,'(2X,A10,I10)') 'LINTV    =',LINTV
      WRITE(6,'(2X,A10,I10)') 'NAMPS    =',NAMPS
      IF((NXTSCR+LINTV+2*RADIX*NEDMS+3*NAMPS*NEDMS+80000).GT.MAXSCR)
     +     GOTO 90100
      TRSCR=NXTSCR+LINTV+LTH
      IF(MOD(FLOAT(TRSCR),2.0).EQ.0.0) TRSCR=TRSCR+1
      WRITE(6,'(2X,A10,I10)') 'TRSCR    =',TRSCR
C *** MAIN LOOP
C
      SHTEND=TRSCR+2*RADIX*NEDMS
      DO 900 I=1,NSDMS,1
         WRITE(6,'(2X,A18,I5)') 'PROCESSING SPC NO.',SDMIN+(I-1)
         DO 905 J=1,2*RADIX*NEDMS
            MEM(TRSCR+J-1)=0
905         CONTINUE
C *** READ IN COMPLETE SDOM
         DO 950 J=1,NEDMS,1
C *** GET IV OF NEXT TRACE
            K=NXTSCR+(I-1)*NEDMS+(J-1)
            IV=MEM(K)
            IF(IV.LE.0) GOTO 950
            IV=IV+4096
C *** GET TRACE HEADER ONLY
            CALL DMI014(MEM(TRSCR-LTH),LTH,IV,RCB,FCB,1,IST)
            CRP(J)=MEM(TRSCR-LTH+40)
            CRPTR(J)=MEM(TRSCR-LTH+41)
            ZOFF(J)=NINT(ZMEM(TRSCR-LTH+16))
C *** GET TRACE DATA ONLY
            CALL DMI014(MEM(TRSCR+RADIX+(J-1)*2*RADIX),LTA,IV+LTH,
     +                  RCB,FCB,1,IST)
            IF(IST.NE.0)GOTO 90050
950         CONTINUE
C
C----- ADJUST THE TRACE AMLITUDES TO SIMULATE COMPLEX DATA
C
         DO 960 J=1,NEDMS,1
            DO 970 K=1,RADIX,1
               MEM(TRSCR+(2*(K-1))+(J-1)*2*RADIX)=
     +                     MEM(TRSCR+RADIX+(J-1)*2*RADIX+(K-1))
```

```
                ZMEM(TRSCR+RADIX+(J-1)*2*RADIX+(K-1))=0.0
                ZMEM(TRSCR+(2*(K-1)+1)+(J-1)*RADIX)=0.0
970             CONTINUE
960           CONTINUE
C
C----- CALL SUBROUTINE TO DO COMPLEX WORK
C
          SPC=SDMIN+I-1
          CALL CMPLEX(MEM,MEM,MEM,LMEM)
C
C *** WRITE OUT THE INSTANTANEOUS PHASE SHOT RECORD
          ERRCNT=SDMIN+(I-1)*SDINC
          DO 980 J=1,NEDMS,1
C *** GET IV OF NEXT TRACE
             K=NXTSCR+(I-1)*NEDMS+(J-1)
             IV=MEM(K)
             IF(IV.LE.0) GOTO 980
             IV=IV+4096
C *** WRITE TRACE DATA ONLY
             CALL DMO015(MEM(PHSSTR+(J-1)*RADIX),NAMPS,IV+LTH,
     +                   RCB,FCB,1,IST)
             IF(IST.NE.0)GOTO 90040
980          CONTINUE
C
900   CONTINUE
C
C *** CLOSE OUTPUT FILE
C
      CALL CLO074(FCB,0,1,IST)
      IF(IST.NE.0)GOTO 90070
      CLOSE(4)
C
C
      WRITE(6,80010)
80010 FORMAT(' ',//,' NORMAL COMPLETION')
      STOP
C
C
90000 WRITE(6,90005)
90005 FORMAT(' ',//,'    PICKER - ERROR OPENING TRACES FILE',/,
     *            '             ENSURE ACNM OF INPUT FILE IS INPUT')
      GOTO 99999
C
90010 WRITE(6,90015)
90015 FORMAT(' ',//,'    PICKER -ERROR READING SUBFILE HEADER')
      GOTO 99999
C
90020 WRITE(6,90025)
90025 FORMAT(' ',//,'    PICKER - ERROR INPUT FILE NOT ATD/TRACES')
      GOTO 99999
C
90030 WRITE(6,90035)
90035 FORMAT(' ',//,'    PICKER - ERROR READING INTERVAL VECTORS')
      GOTO 99999
C
90040 WRITE(6,90045)ERRCNT
90045 FORMAT(' ',//,'    PICKER - ERROR OUTPUTTING SPC NO. ',I8)
      GOTO 99999
C
90050 WRITE(6,90055)
90055 FORMAT(' ',//,'    PICKER - ERROR READING TRACE DATA')
      GOTO 99999
C
90070 WRITE(6,90075)
90075 FORMAT(' ',//,'    PICKER - ERROR CLOSING OUTPUT FILE')
      GOTO 99999
C
90100 WRITE(6,90105)
90105 FORMAT(' ',//,'    PICKER - ERROR - LTP PARAMETER TOO SMALL')
      GOTO 99999
C
```

```
90200 WRITE(6,90205)
90205 FORMAT(' ',//,'    PICKER - MORE THAN 4000 SAMPLES/TRACE   ')
      GOTO 99999
C
90300 WRITE(6,90305)
90305 FORMAT(' ',//,'    PICKER - MORE THAN 490 TRACES/SDOM      ')
      GOTO 99999
C
C
99999 STOP 15
      END
@PROCESS OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS FLAG(E),
@PROCESS DIRECTIVE('*VD:')
      SUBROUTINE CMPLEX (MEM,YMEM,ZMEM,LMEM)
      IMPLICIT INTEGER (A-X)
      IMPLICIT COMPLEX*8 (Y)
      COMMON /SCRPTR/ IBSCR,NXTSCR,MAXSCR
      COMMON /CMPPTR/ TRSCR,RADIX,SHTEND,NEDMS,NAMPS,PHSSTR,SPC,ZDELTT
      COMMON /CRPOFF/ CRP,CRPTR,ZOFF
      DIMENSION FCB(20),RCB(20),TRACES(2)
      DIMENSION MEM(1),YMEM(1),ZMEM(1)
      DIMENSION ZPICKS(150,50,6)
      DIMENSION CRP(150),CRPTR(150),ZOFF(150)

      ZTWOPI=8.0*ATAN(1.0)
      ZPI=4.0*ATAN(1.0)
      ZPION2=2.0*ATAN(1.0)
c
c----- PERFORM FORWARD FOURIER TRANSFORMS
c
       TRSCR2=(TRSCR+1)/2
       SHTEND2=SHTEND/2+2
       IF((SHTEND+120000).GT.MAXSCR) GOTO 90100
       CALL SCFT(1,YMEM(TRSCR2),1,RADIX,YMEM(TRSCR2),1,RADIX,
     +           RADIX,NEDMS,1,1.0,YMEM(SHTEND2),20000,
     +           YMEM(SHTEND2+20000),20000)
       CALL SCFT(0,YMEM(TRSCR2),1,RADIX,YMEM(TRSCR2),1,RADIX,
     +           RADIX,NEDMS,1,1.0,YMEM(SHTEND2),20000,
     +           YMEM(SHTEND2+20000),20000)
C
C----- ZERO THE NEGATIVE FREQUENCIES
C
       DO 100 I=1,NEDMS,1
          ZA=TRSCR2+(I-1)*RADIX
          DO 200 J=RADIX/2+1,RADIX
             YMEM(ZA+J)=CMPLX(0.0,0.0)
200          CONTINUE
          YMEM(ZA)=YMEM(ZA)/2.0
100       CONTINUE
c
c----- PERFORM INVERSE 2D FOURIER TRANSFORM
c
       CALL SCFT(1,YMEM(TRSCR2),1,RADIX,YMEM(TRSCR2),1,RADIX,
     +      RADIX,NEDMS,-1,1.0/FLOAT(RADIX),YMEM(SHTEND2),20000,
     +           YMEM(SHTEND2+20000),20000)
       CALL SCFT(0,YMEM(TRSCR2),1,RADIX,YMEM(TRSCR2),1,RADIX,
     +      RADIX,NEDMS,-1,1.0/FLOAT(RADIX),YMEM(SHTEND2),20000,
     +           YMEM(SHTEND2+20000),20000)
c
c----- COMPUTE INSTANTANEOUS AMPLITUDE
c
       AMPSTR=SHTEND+80100
       IF((AMPSTR+NEDMS*RADIX).GT.MAXSCR) GOTO 90100
       Z1=0.0
       DO 300 I=1,NEDMS
          K=(I-1)*RADIX
          DO 400 J=0,NAMPS-1
             ZMEM(AMPSTR+K+J)=SQRT(REAL(YMEM(TRSCR2+K+J))**2+
     +                             IMAG(YMEM(TRSCR2+K+J))**2)
             IF(ZMEM(AMPSTR+K+J).GT.Z1) Z1=ZMEM(AMPSTR+K+J)
```

```
400          CONTINUE
300        CONTINUE
c
C----- COMPUTE INSTANTANEOUS PHASE
c
       PHSSTR=AMPSTR+NEDMS*RADIX
       IF((PHSSTR+NEDMS*RADIX).GT.MAXSCR) GOTO 90100
       DO 500 I=1,NEDMS,1
          K=(I-1)*RADIX
          DO 600 J=1,NAMPS-2,1
             IF(ABS(REAL(YMEM(TRSCR2+K+J))).LT.1.E-10) THEN
                IF(IMAG(YMEM(TRSCR2+K+J)).LT.0.0) THEN
                   ZMEM(PHSSTR+K+J)=-1.0*ZPION2
                ELSE
                   ZMEM(PHSSTR+K+J)=ZPION2
                   ENDIF
                GOTO 600
                ENDIF
             ZA=ATAN(IMAG(YMEM(TRSCR2+K+J))/REAL(YMEM(TRSCR2+K+J)))
             IF(REAL(YMEM(TRSCR2+K+J)).LT.0.0) THEN
                IF(ZA.LT.0.0) THEN
                   ZA=ZPI+ZA
                ELSE
                   ZA=-1.0*ZPI+ZA
                   ENDIF
                ENDIF
             ZMEM(PHSSTR+K+J)=ZA
600       CONTINUE
500    CONTINUE
c
c----- COMPUTE PHASE VELOCITY
c
       VELSTR=PHSSTR+NEDMS*RADIX
       IF((VELSTR+NEDMS*RADIX).GT.MAXSCR) GOTO 90100
       DO 700 I=2,NEDMS-1,1
          K=(I-1)*RADIX
          KN=((I-1)-1)*RADIX
          KP=((I+1)-1)*RADIX
          DO 800 J=1,NAMPS-2,1
             IF(ZMEM(AMPSTR+K+J).LT.0.05*Z1) THEN
                ZMEM(VELSTR+K+J)=0.0
                GOTO 800
                ENDIF
             IF((ZMEM(AMPSTR+K+J).LT.ZMEM(AMPSTR+K+J-1)).OR.
     +          (ZMEM(AMPSTR+K+J).LT.ZMEM(AMPSTR+K+J+1))) THEN
                ZMEM(VELSTR+K+J)=0.0
                GOTO 800
                ENDIF
 C----- PICK LARGEST AMPLITUDE WITHIN 200 MSEC ONLY
             DO 850 JJ=-12,-1,1
                IF((J+JJ).LT.0) GOTO 850
                IF((ZMEM(VELSTR+K+J+JJ).NE.0.0).AND.
     +             (ZMEM(AMPSTR+K+J+JJ).LE.ZMEM(AMPSTR+K+J))) THEN
                   ZMEM(VELSTR+K+J+JJ)=0.0
                ELSEIF((ZMEM(VELSTR+K+J+JJ).NE.0.0).AND.
     +             (ZMEM(AMPSTR+K+J+JJ).GT.ZMEM(AMPSTR+K+J))) THEN
                   ZMEM(VELSTR+K+J)=0.0
                   GOTO 800
                   ENDIF
 850            CONTINUE
             ZA=ZMEM(PHSSTR+K+J+1)
             ZM=ZMEM(PHSSTR+K+J)
             ZB=ZMEM(PHSSTR+K+J-1)
             ZC=ZMEM(PHSSTR+KP+J)
             ZD=ZMEM(PHSSTR+KN+J)
 C----- CALCULATE DIFFERENCES
             CALL DIFF(ZA,ZM,ZU1)
             CALL DIFF(ZM,ZB,ZU2)
             CALL DIFF(ZC,ZM,ZL1)
             CALL DIFF(ZM,ZD,ZL2)
 C----- IF TWO HALF DIFFERENCES ARE VERY DIFFERENT, IGNORE POINT
```

```
            ZT=ZU1/ZU2
            IF((ZT.LT.0.50).OR.(ZT.GT.1.50)) GOTO 800
            ZT=ZL1/ZL2
            IF((ZT.LT.0.50).OR.(ZT.GT.1.50)) GOTO 800
C----- COMPUTE AVERAGE DIFFERENCES
            ZU=(ZU1+ZU2)/2.0
            ZL=(ZL1+ZL2)/2.0
C----- CHECK TO SEE IF DIP IS NEGATIVE AND CORRECT IF SO
            IF(ZL.GT.ZPI) ZL=ZL-ZTWOPI
            ZMEM(VELSTR+K+J)=ZU/ZL
            ZDELTX=ABS(ZOFF(I+1)-ZOFF(I-1))/2.0
            ZMEM(VELSTR+K+J)=-1.0*ZMEM(VELSTR+K+J)*ZDELTX/ZDELTT
800       CONTINUE
700     CONTINUE
C
C----- CONVERT PHASE VELOCITIES TO ANGLES AND SAVE IN ZPICKS
C
        DO 900 I=2,NEDMS-1,1
           K=(I-1)*RADIX
           COUNT=1
           DO 901 J=1,25
              ZPICKS(I,J,2)=0.0
901           CONTINUE
           DO 1000 J=1,NAMPS-2,1
              IF(ABS(ZMEM(VELSTR+K+J)).LT.1.48) THEN
                 ZMEM(VELSTR+K+J)=0.0
                 GOTO 1000
                 ENDIF
              ZMEM(VELSTR+K+J)=ASIN(1.48/ZMEM(VELSTR+K+J))
              ZXSRC=(472-SPC)*25.0
              ZTIME=J*0.004
              ZXRCV=ZXSRC+ZOFF(I)
              ZANGRC=ZMEM(VELSTR+K+J)
              IF(ZXRCV.GT.11780.0) GOTO 1000
              ZPICKS(I,COUNT,1)=ZXSRC
              ZPICKS(I,COUNT,2)=ZTIME
              ZPICKS(I,COUNT,3)=FLOAT(CRP(I))
              ZPICKS(I,COUNT,4)=FLOAT(CRPTR(I))
              ZPICKS(I,COUNT,5)=ZXRCV
              ZPICKS(I,COUNT,6)=ZANGRC
              COUNT=COUNT+1
1000          CONTINUE
900       CONTINUE
C
C----- CHECK ZPICKS AND OUTPUT ONLY CONSISTENT PICKS
C
        DO 1100 I=3,NEDMS-2
           DO 1110 J=1,25
              ZTIME=ZPICKS(I,J,2)
              IF(ZTIME.EQ.0.0) GOTO 1100
              DO 1120 K=1,25
                 IF(ABS(ZPICKS(I-1,K,2)-ZTIME).LT.5.000) THEN
                 KK=K
                 GOTO 1130
                 ENDIF
1120             CONTINUE
              GOTO 1110
1130          DO 1140 K=1,25
                 IF(ABS(ZPICKS(I+1,K,2)-ZTIME).LT.5.000) THEN
                 KKK=K
                 GOTO 1150
                 ENDIF
1140             CONTINUE
              GOTO 1110
1150          ZK=ZPICKS(I,K,6)/((ZPICKS(I-1,KK,6)+ZPICKS(I+1,KKK,6))/2.0)
              WRITE(4,5000) ZPICKS(I,J,1),ZPICKS(I,J,2),
     +                      INT(ZPICKS(I,J,3)),INT(ZPICKS(I,J,4)),
     +                      ZPICKS(I,J,5),ZPICKS(I,J,6)
1110          CONTINUE
1100       CONTINUE
5000    FORMAT(1X,2G16.8,2I6,2G16.8)
```

```
       RETURN
90100 WRITE(6,90105)
90105 FORMAT(' ',//,'    CMPLEX - ERROR - LTP PARAMETER TOO SMALL')
      GOTO 99999
C
C
99999 STOP 15
      END
      SUBROUTINE DIFF(ZA,ZM,ZU1)
      REAL ZA,ZM,ZU1,ZTWOPI
      ZTWOPI=8.0*ATAN(1.0)
      IF(((ZA.GE.0.0).AND.(ZM.GE.0.0)).OR.
     +   ((ZA.LT.0.0).AND.(ZM.LT.0.0))) THEN
           IF(ZA.GE.ZM) THEN
              ZU1=ZA-ZM
           ELSE
              ZU1=ZTWOPI-(ZM-ZA)
              ENDIF
           ENDIF
      IF((ZA.GE.0.0).AND.(ZM.LT.0.0)) ZU1=ZA-ZM
      IF((ZA.LT.0.0).AND.(ZM.GE.0.0)) ZU1=ZTWOPI-(ZM-ZA)
      RETURN
      END
```

```
@PROCESS DC(GAUS),OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS DIRECTIVE('*VD:')
c=======================================================================
c----- this subroutine traces a ray through a single cell
c=======================================================================
      subroutine raytrace(x,theta0,z,t,triflag,tri,triloc)
c-----------------------------------------------------------------------
c----- nq   = maximum number of rows in velocity grid
c----- mq   = maximlm number of columns in velocity grid
c----- n    = actual number of rows in velocity grid
c----- m    = actual number of columns in velocity grid
c----- nint = spatial increment between grid rows
c----- mint = spatial increment between grid columns
c-----
c----- vel(nq,mq)= values of velocity at grid points
c-----
c----- tol  = 1.0d-10
c----- pi,pion2,twopi = value of pi, pi divided by 2 and pi times 2
c----- i,j,k= temporary integer variables
c----- t1,t2,t3,t4,t5= temporary double-precision variables
c----- triflag = .true. if triangle flat on top, .false. otherwise
c----- tri(3,2)= grid points of current triangle
c-----         - 1st dimension for the three corners
c-----         - 2nd dimension for the column/row
c----- triloc(3,2)= grid point locations of current triangle (in metres)
c-----            - 1st dimension for the three corners
c-----            - 2nd dimension for the column/row
c----- v0   = initial velocity in cell
c----- x,z  = coordinates of the ray (in metres)
c----- t    = cumulative travel time of the ray
c----- vx,vz= lateral and vertical components of the velocity gradient
c-----       for the current cell
c----- velang= angle the velocity gradient makes with the reference axis
c----- gradient= magnitude of the velocity gradient
c----- radius= radius of circular arc within current cell
c----- radius2= the square of the radius
c----- straight= .true. if ray is determined to be straight
c----- s,e  = parameters of straight ray, z=s.x+e
c-----
c-----
c----- a,b  = parameters of the local circular arc
c----- a2,b2= the squares of the above quantities
c----- slope,c  = parameters of sloping interface of triangle, z=slope.x+c
c----- slope2   = the square of slope
c----- intpoints= array of possible ray intersection points with cell boundary
c-----           - first entry is the x location, second is the z location
c----- rslope,d = parameters of local reflector line segment, z=rslope.x+d
c----- rslope2  = the square of rslope
c----- clockwise= 1.0 if ray travelling clockwise on arc, -1.0 otherwise
c----- outangle = array holding exit angles at the intersection points
c----- pathlength = array holding path lengths of arcs to exit points
c----- exitpoint = flagged index of the exit point from cell
c-----
c-----------------------------------------------------------------------
      real*4 nint,mint
      real*4 vel
      real*8 pi,pion2,twopi
      real*8 tol
      real*8 x,z,t
      real*8 triloc
      real*8 v0,vx,vz
      real*8 theta0,velang,gradient
      real*8 radius,radius2
      real*8 a,b,a2,b2
      real*8 slope,c,slope2
      real*8 s,e
      real*8 intpoints
      real*8 rslope,d
      real*8 t1,t2,t3,t4,t5
      real*8 val1,val2
      real*8 clockwise
      real*8 outangle,pathlength
```

```
      logical triflag,straight
      integer*4 loop
      integer*4 i,j,k
      integer*4 n,m,nq,mq,nmax,mmax,max,ndat,dne
      integer*4 tri
      integer*4 exitpoint

      parameter(nq=90,mq=210)

      dimension vel(nq,mq)
      dimension tri(3,2),triloc(3,2)
      dimension intpoints(8,2)
      dimension outangle(8),pathlength(8)
      common /velocity/vel
      common /constant/tol,pi,pion2,twopi
      common /conparm/n,m,nint,mint,nmax,mmax,max,ndat,dne
c
c----- set constants and initial variables
c
      t1=0.0d0
      t2=0.0d0
      t3=0.0d0
      t4=0.0d0
      t5=0.0d0
      straight=.false.
c
c----- check to see if ray is passing through a grid point and if so,
c----- shift it by one tenth of a millimetre
c
      if((dmod(x,dble(mint)).eq.0.0d0).and.
     +   (dmod(z,dble(nint)).eq.0.0d0)) then
         x=x+.0001d0
         endif
c
c----- compute x and z components of the velocity gradient
c
      vx=(dble(vel(tri(2,2),tri(2,1))-vel(tri(1,2),tri(1,1))))
     +   /dble(mint)
      if(triflag) then
         vz=(dble(vel(tri(3,2),tri(3,1))-vel(tri(2,2),tri(2,1))))
     +      /dble(nint)
         else
         vz=(dble(vel(tri(1,2),tri(1,1))-vel(tri(3,2),tri(3,1))))
     +      /dble(nint)
         endif
c
c----- compute initial velocity in cell
c
      v0=dble(vel(tri(1,2),tri(1,1)))+(x-dble(tri(1,1)-1)*dble(mint))*vx
     +                          +(z-dble(tri(1,2)-1)*dble(nint))*vz
c
c----- compute angle velocity gradient makes with reference axes and
c----- the magnitude of the velocity gradient
c
      if(dabs(vz).lt.1.0d-10) then
         velang=dsign(pion2,vx)
      elseif(dabs(vx).lt.1.0d-10) then
         velang=pion2-dsign(pion2,vz)
      elseif(vz.gt.0.0d0) then
         velang=datan(vx/vz)
      else
         velang=dsign(1.0d0,vx)*(pi-datan(dabs(vx/vz)))
         endif
2     gradient=dsqrt(vz**2+vx**2)
c
c----- if the gradient is zero then flag the ray for straight-ray tracing
c
      if(gradient.lt.tol) then
         straight=.true.
         goto 11
         endif
c
```

```
c----- compute radius of ray in cell
c
      if(dabs(theta0-velang).gt.tol) then
         radius=v0/dabs(gradient*dsin(theta0-velang))
      else
         radius=1.0d10
         endif
3     radius2=radius*radius
c
c----- if the radius of the ray is very large, flag it as straight
c----- currently, larger than 1.0d6 means double precision cannot perform
c----- the required calulations for curved rays
c
      if(radius.lt.1.0d6) goto 12
      straight=.true.
c
c----- jump to here if ray flagged straight
c----- compute straight ray parameters, z=s.x+e
c
11    if(theta0.eq.0.0d0) goto 12
      s=1.0d0/dtan(theta0)
      e=z-s*x
12    continue
c
c----- here only if ray is curved
c----- compute parameters of the circle (x-a)**2+(z-b)**2=rad**2
c
      if(straight) goto 13
c---------- convert theta0 from range -pi to pi to range -pion2 to pion2
c---------- this removes sense of direction of ray
      t2=theta0
      if(theta0.gt.pion2) t2=theta0-pi
      if(theta0.lt.-1.0d0*pion2) t2=theta0+pi
c---------- need to be careful of location of centre of circle
      val1=dcos(t2)
      val2=dsin(t2)
c
c----- check first of the two possible ray circle centres. Is vel zero?
c
      t1=1.0d0
      a=x+t1*val1*radius
      b=z-t1*val2*radius
      t3=(a-x)*vx+(b-z)*vz+v0
      if(dabs(t3).lt.1.0d0) goto 4
c
c----- check other possibility if the first was no good
c
      t1=-1.0d0
      a=x+t1*val1*radius
      b=z-t1*val2*radius
      t3=(a-x)*vx+(b-z)*vz+v0
      if(dabs(t3).ge.1.0d0) stop 'no suitable centre for ray'
c----- continue on
4     a2=a*a
      b2=b*b
13    continue
c-------------------------------------------------------------------
c----- find the exit point of the ray
c-------------------------------------------------------------------
c
c----- check the horizontal side of triangle for intersections
c
      if(straight) then
         if(theta0.eq.0.0d0) then
            t3=x
            t4=-1.0d0
         else
            t3=(triloc(1,2)-e)/s
            t4=-1.0d0
            endif
         goto 14
         endif
```

```
      t1=radius2-(triloc(1,2)-b)**2
      if(t1) 20,10,10
10    t2=dsqrt(t1)
      t3=a+t2
      t4=a-t2
14    if((t3.lt.triloc(1,1)).or.(t3.gt.triloc(2,1))) then
         intpoints(1,1)=-1.0d0
         intpoints(1,2)=-1.0d0
         else
         intpoints(1,1)=t3
         intpoints(1,2)=triloc(1,2)
         endif
      if((t4.lt.triloc(1,1)).or.(t4.gt.triloc(2,1))) then
         intpoints(2,1)=-1.0d0
         intpoints(2,2)=-1.0d0
         else
         intpoints(2,1)=t4
         intpoints(2,2)=triloc(1,2)
         endif
      goto 30
20    intpoints(1,1)=-1.0d0
      intpoints(1,2)=-1.0d0
      intpoints(2,1)=-1.0d0
      intpoints(2,2)=-1.0d0
30    continue
c
c----- check the vertical side of triangle for intersections
c
      if(straight) then
         if(theta0.eq.0.0d0) then
            t3=-1.0d0
            t4=-1.0d0
            goto 15
            endif
         t3=s*triloc(3,1)+e
         t4=-1.0d0
         goto 15
         endif
      if(triflag) then
         t1=radius2-(triloc(2,1)-a)**2
      else
         t1=radius2-(triloc(1,1)-a)**2
         endif
      if(t1) 60,40,40
40    t2=dsqrt(t1)
      t3=b+t2
      t4=b-t2
15    if(.not.(triflag)) goto 50
      if((t3.lt.triloc(2,2)).or.(t3.gt.triloc(3,2))) then
         intpoints(3,1)=-1.0d0
         intpoints(3,2)=-1.0d0
         else
         intpoints(3,1)=triloc(2,1)
         intpoints(3,2)=t3
         endif
      if((t4.lt.triloc(2,2)).or.(t4.gt.triloc(3,2))) then
         intpoints(4,1)=-1.0d0
         intpoints(4,2)=-1.0d0
         else
         intpoints(4,1)=triloc(2,1)
         intpoints(4,2)=t4
         endif
      goto 70
50    if((t3.gt.triloc(2,2)).or.(t3.lt.triloc(3,2))) then
         intpoints(3,1)=-1.0d0
         intpoints(3,1)=-1.0d0
         else
         intpoints(3,1)=triloc(1,1)
         intpoints(3,2)=t3
         endif
      if((t4.gt.triloc(2,2)).or.(t4.lt.triloc(3,2))) then
         intpoints(4,1)=-1.0d0
```

```
            intpoints(4,2)=-1.0d0
            else
            intpoints(4,1)=triloc(1,1)
            intpoints(4,2)=t4
            endif
         goto 70
60       intpoints(3,1)=-1.0d0
         intpoints(3,2)=-1.0d0
         intpoints(4,1)=-1.0d0
         intpoints(4,2)=-1.0d0
70       continue
c
c----- check the sloping side of the triangle. First find the parameters of the
c----- of the line z=slope.x+c, then solve quadratic similarly to above.
c
         if(triflag) then
            slope=(triloc(1,2)-triloc(3,2))/(triloc(1,1)-triloc(3,1))
            else
            slope=(triloc(2,2)-triloc(3,2))/(triloc(2,1)-triloc(3,1))
            endif
         c=triloc(3,2)-slope*triloc(3,1)
         if(straight) then
            if(theta0.eq.0.0d0) then
               t3=x
               t4=-1.0d0
               goto 16
               endif
            t3=(c-e)/(s-slope)
            t4=-1.0d0
            goto 16
            endif
         slope2=slope**2
         t1=(slope*(c-b)-a)**2-(slope2+1.0d0)*(a2+(c-b)**2-radius2)
         if(t1) 90,80,80
80       t2=dsqrt(t1)
         t3=((a-slope*(c-b))+t2)/(slope2+1.0d0)
         t4=((a-slope*(c-b))-t2)/(slope2+1.0d0)
16       if((t3.lt.triloc(1,1)).or.(t3.gt.triloc(2,1))) then
            intpoints(5,1)=-1.0d0
            intpoints(5,2)=-1.0d0
            else
            intpoints(5,1)=t3
            intpoints(5,2)=slope*t3+c
            endif
         if((t4.lt.triloc(1,1)).or.(t4.gt.triloc(2,1))) then
            intpoints(6,1)=-1.0d0
            intpoints(6,2)=-1.0d0
            else
            intpoints(6,1)=t4
            intpoints(6,2)=slope*t4+c
            endif
         goto 100
90       intpoints(5,1)=-1.0d0
         intpoints(5,2)=-1.0d0
         intpoints(6,1)=-1.0d0
         intpoints(6,2)=-1.0d0
100      continue
c
c----- compute exit angles for all the possible intersection points
c----- need to determine sense of rotation of the ray (clockwise or not)
c----- flag anticlockwise rays with a negative sign
c
         if(straight) goto 18
         if(theta0.eq.0.0d0) then
            if(a.lt.x) then
               clockwise=1.0d0
            else
               clockwise=-1.0d0
               endif
            goto 145
            endif
         if(dabs(theta0).eq.pi) then
```

```
            if(a.lt.x) then
               clockwise=-1.0d0
            else
               clockwise=1.0d0
               endif
            goto 145
            endif
         if(dabs(theta0).eq.pion2) then
            if(b.lt.z) then
               clockwise=dsign(1.0d0,-1.0d0*theta0)
            else
               clockwise=dsign(1.0d0,theta0)
               endif
            goto 145
            endif
         if((a.ge.x).and.(b.ge.z)) then
           clockwise=dsign(1.0d0,theta0)
           goto 145
           endif
         if((a.ge.x).and.(b.lt.z)) then
           clockwise=dsign(1.0d0,-1.0d0*theta0)
           goto 145
           endif
         if((a.lt.x).and.(b.ge.z)) then
           clockwise=dsign(1.0d0,theta0)
           goto 145
           endif
         if((a.lt.x).and.(b.lt.z)) then
           clockwise=dsign(1.0d0,-1.0d0*theta0)
           endif
145      do 140 i=1,6
            if(intpoints(i,1).eq.-1.0d0) goto 140
            if(a-intpoints(i,1)) 170,160,150
150         outangle(i)=dasin((intpoints(i,2)-b)/radius)
            goto 146
160         outangle(i)=pion2
            goto 146
170         outangle(i)=dasin((b-intpoints(i,2))/radius)
146         if((a.gt.intpoints(i,1)).and.(b.gt.intpoints(i,2))) then
              if(clockwise.gt.0.0d0) outangle(i)=outangle(i)+pi
            elseif((a.gt.intpoints(i,1)).and.(b.lt.intpoints(i,2))) then
              if(clockwise.gt.0.0d0) outangle(i)=outangle(i)-pi
            elseif((a.lt.intpoints(i,1)).and.(b.gt.intpoints(i,2))) then
              if(clockwise.lt.0.0d0) outangle(i)=outangle(i)-pi
            elseif((a.lt.intpoints(i,1)).and.(b.lt.intpoints(i,2))) then
              if(clockwise.lt.0.0d0) outangle(i)=outangle(i)+pi
            elseif(a.eq.intpoints(i,1)) then
               outangle(i)=dsign(pion2,-1.0d0*clockwise)
            elseif(b.eq.intpoints(i,2)) then
               if(clockwise.lt.0.0d0) outangle(i)=outangle(i)-pi
               endif
139         if(outangle(i).lt.0.0d0) outangle(i)=twopi+outangle(i)
            outangle(i)=clockwise*outangle(i)
140         continue
18       continue
c
c----- compute path lengths to each of the intersection points
c----- remembering to compute length in direction of travel
c----- also find nearest intpoint and set its pathlength to 1.0d20
c
         t1=theta0
         t3=1.0d70
         k=0
         if(theta0.lt.0.0d0) t1=twopi+theta0
         do 180 i=1,6
            if(intpoints(i,1).eq.-1.0d0) then
               pathlength(i)=1.0d20
            elseif(straight) then
               pathlength(i)=dsqrt((intpoints(i,1)-x)**2+
     +                     (intpoints(i,2)-z)**2)
            else
               if(outangle(i).lt.0.0d0) then
```

```
               t2=dabs(outangle(i))-t1
            else
               t2=t1-outangle(i)
               endif
            if(t2.lt.0.0d0) t2=twopi+t2
            pathlength(i)=radius*t2
            endif
          t4=(x-intpoints(i,1))**2+(z-intpoints(i,2))**2
          if(t4.lt.t3) then
             t3=t4
             k=i
             endif
180       continue
       pathlength(k)=1.0d20
c
c----- choose shortest pathlength left and flag that exit point
c
       exitpoint=0
       t1=1.0d20
       do 195 i=6,1,-1
          if(pathlength(i).lt.t1) then
             t1=pathlength(i)
             exitpoint=i
             endif
195       continue
c
c----- compute time through this cell and add to ray's total time
c
       if(straight) then
          t1=((intpoints(exitpoint,1)-x)*vx)+
     +       ((intpoints(exitpoint,2)-z)*vz)+v0
c----- nb: harmonic mean
          t1=(1/t1+1/v0)/2
          t1=pathlength(exitpoint)*t1
          t=t+t1
       else
          t2=dabs(outangle(exitpoint))
          if(t2.gt.pi) t2=t2-twopi
          t4=dabs(velang-t2)
          t5=dabs(velang-theta0)
          if(t4.gt.pi) t4=dabs(t4-twopi)
          if(t5.gt.pi) t5=dabs(t5-twopi)
          t1=(1/dabs(gradient))*dlog(dtan(t4/2.0d0)/dtan(t5/2.0d0))
          t=t+t1
          endif
c
c----- reset necessary parameters for completion of this triangle
c
       x=intpoints(exitpoint,1)
       z=intpoints(exitpoint,2)
       if(.not.(straight)) then
          theta0=dabs(outangle(exitpoint))
          if(theta0.gt.pi) theta0=theta0-twopi
          endif
       straight=.false.
c
c----- check to see if ray is passing through a grid point and if so,
c----- shift it by one tenth of a millimetre and compute new cell
c
       if((dmod(x,dble(mint)).lt.tol).and.
     +    (dmod(z,dble(nint)).lt.tol)) then
       if(triflag) then
          if(((x-triloc(1,1)).lt.tol).and.((z-triloc(1,2)).lt.tol)) then
             x=triloc(1,1)
             z=triloc(1,2)+0.0001d0
             tri(1,1)=tri(3,1)-1
             tri(1,2)=tri(3,2)
             tri(2,1)=tri(1,1)+1
             tri(2,2)=tri(1,2)
             tri(3,1)=tri(1,1)
             tri(3,2)=tri(1,2)-1
             triloc(1,1)=triloc(3,1)-dble(mint)
```

```
       triloc(1,2)=triloc(3,2)
       triloc(2,1)=triloc(1,1)+dble(mint)
       triloc(2,2)=triloc(1,2)
       triloc(3,1)=triloc(1,1)
       triloc(3,2)=triloc(1,2)-dble(nint)
elseif(((x-triloc(2,1)).lt.tol).and.((z-triloc(2,2)).lt.tol)) then
       x=triloc(2,1)
       z=triloc(2,2)+0.0001d0
       tri(1,1)=tri(3,1)
       tri(1,2)=tri(3,2)
       tri(2,1)=tri(1,1)+1
       tri(2,2)=tri(1,2)
       tri(3,1)=tri(1,1)
       tri(3,2)=tri(1,2)-1
       triloc(1,1)=triloc(3,1)
       triloc(1,2)=triloc(3,2)
       triloc(2,1)=triloc(1,1)+dble(mint)
       triloc(2,2)=triloc(1,2)
       triloc(3,1)=triloc(1,1)
       triloc(3,2)=triloc(1,2)-dble(nint)
elseif(((x-triloc(3,1)).lt.tol).and.((z-triloc(3,2)).lt.tol)) then
       x=triloc(3,1)
       z=triloc(3,2)-0.0001d0
       tri(1,1)=tri(3,1)
       tri(1,2)=tri(3,2)
       tri(2,1)=tri(1,1)+1
       tri(2,2)=tri(1,2)
       tri(3,1)=tri(1,1)
       tri(3,2)=tri(1,2)-1
       triloc(1,1)=triloc(3,1)
       triloc(1,2)=triloc(3,2)
       triloc(2,1)=triloc(1,1)+dble(mint)
       triloc(2,2)=triloc(1,2)
       triloc(3,1)=triloc(1,1)
       triloc(3,2)=triloc(1,2)-dble(nint)
       endif
 else
    if(((x-triloc(1,1)).lt.tol).and.((z-triloc(1,2)).lt.tol)) then
       x=triloc(1,1)+0.0001d0
       z=triloc(1,2)
       tri(3,1)=tri(2,1)
       tri(3,2)=tri(2,2)+1
       triloc(3,1)=triloc(2,1)
       triloc(3,2)=triloc(2,2)+dble(nint)
elseif(((x-triloc(2,1)).lt.tol).and.((z-triloc(2,2)).lt.tol)) then
       x=triloc(2,1)-0.0001d0
       z=triloc(2,2)
       tri(3,1)=tri(2,1)
       tri(3,2)=tri(2,2)+1
       triloc(3,1)=triloc(2,1)
       triloc(3,2)=triloc(2,2)+dble(nint)
elseif(((x-triloc(3,1)).lt.tol).and.((z-triloc(3,2)).lt.tol)) then
       x=triloc(3,1)
       z=triloc(3,2)+0.0001d0
       tri(1,1)=tri(3,1)-1
       tri(1,2)=tri(3,2)
       tri(2,1)=tri(1,1)+1
       tri(2,2)=tri(1,2)
       tri(3,1)=tri(2,1)
       tri(3,2)=tri(2,2)+1
       triloc(1,1)=triloc(3,1)-dble(mint)
       triloc(1,2)=triloc(3,2)
       triloc(2,1)=triloc(1,1)+dble(mint)
       triloc(2,2)=triloc(1,2)
       triloc(3,1)=triloc(2,1)
       triloc(3,2)=triloc(2,2)+dble(nint)
       endif
    endif
 triflag=.not.(triflag)
 goto 1000
 endif
 if((exitpoint.eq.1).or.(exitpoint.eq.2)) then
```

```
      if(triflag) then
         tri(3,1)=tri(3,1)-1
         tri(3,2)=tri(3,2)-2
         triloc(3,1)=triloc(3,1)-dble(mint)
         triloc(3,2)=triloc(3,2)-2.0d0*dble(nint)
      else
         tri(3,1)=tri(3,1)+1
         tri(3,2)=tri(3,2)+2
         triloc(3,1)=triloc(3,1)+dble(mint)
         triloc(3,2)=triloc(3,2)+2.0d0*dble(nint)
         endif
      triflag=.not.(triflag)
      endif
   if((exitpoint.eq.3).or.(exitpoint.eq.4)) then
      if(triflag) then
         tri(1,1)=tri(3,1)
         tri(1,2)=tri(3,2)
         tri(2,1)=tri(1,1)+1
         tri(2,2)=tri(1,2)
         tri(3,1)=tri(1,1)
         tri(3,2)=tri(1,2)-1
         triloc(1,1)=triloc(3,1)
         triloc(1,2)=triloc(3,2)
         triloc(2,1)=triloc(1,1)+dble(mint)
         triloc(2,2)=triloc(1,2)
         triloc(3,1)=triloc(1,1)
         triloc(3,2)=triloc(1,2)-dble(nint)
      else
         tri(1,1)=tri(3,1)-1
         tri(1,2)=tri(3,2)
         tri(2,1)=tri(1,1)+1
         tri(2,2)=tri(1,2)
         tri(3,1)=tri(1,1)+1
         tri(3,2)=tri(1,2)+1
         triloc(1,1)=triloc(3,1)-dble(mint)
         triloc(1,2)=triloc(3,2)
         triloc(2,1)=triloc(1,1)+dble(mint)
         triloc(2,2)=triloc(1,2)
         triloc(3,1)=triloc(1,1)+dble(mint)
         triloc(3,2)=triloc(1,2)+dble(nint)
         endif
      triflag=.not.(triflag)
      endif
   if((exitpoint.eq.5).or.(exitpoint.eq.6)) then
      if(triflag) then
         tri(1,1)=tri(3,1)-1
         tri(1,2)=tri(3,2)
         tri(2,1)=tri(1,1)+1
         tri(2,2)=tri(1,2)
         tri(3,1)=tri(1,1)
         tri(3,2)=tri(1,2)-1
         triloc(1,1)=triloc(3,1)-dble(mint)
         triloc(1,2)=triloc(3,2)
         triloc(2,1)=triloc(1,1)+dble(mint)
         triloc(2,2)=triloc(1,2)
         triloc(3,1)=triloc(1,1)
         triloc(3,2)=triloc(1,2)-dble(nint)
      else
         tri(1,1)=tri(3,1)
         tri(1,2)=tri(3,2)
         tri(2,1)=tri(1,1)+1
         tri(2,2)=tri(1,2)
         tri(3,1)=tri(1,1)+1
         tri(3,2)=tri(1,2)+1
         triloc(1,1)=triloc(3,1)
         triloc(1,2)=triloc(3,2)
         triloc(2,1)=triloc(1,1)+dble(mint)
         triloc(2,2)=triloc(1,2)
         triloc(3,1)=triloc(1,1)+dble(mint)
         triloc(3,2)=triloc(1,2)+dble(nint)
         endif
      triflag=.not.(triflag)
```

```
            endif

1000    return
        end
```

```
@PROCESSqDC(GAUS),OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS DIRECTIVE('*VD:')
c-------------------------------------------------------------------------------
c----- subroutine to compute the frechet derivatives for submission as     -----
c----- the response matrix to MEM_NL                                       -----
c----- this subroutine uses Semi-Analytic Derivative Estimate no 1         -----
c----- uses Chuck Sword's XERR statistic                                   -----
c----- this routine is called for each ray involved in the data times      -----
c-------------------------------------------------------------------------------
      subroutine sade2(index,sloc1,tang1,sloc2,tang2,datatime,xerr,R)
c-------------------------------------------------------------------------------
c----- index  =  index number of ray as stored in geom file
c----- sloc1  =  surface location of ray1
c----- tang1  =  take off angle of ray1
c----- sloc2  =  surface location of ray2
c----- tang2  =  take off angle of ray2
c----- datatime=  actual measured time of ray
c----- xerr   =  Chuck Sword's statistic
c-----
c----- nq   = maximum number of rows in velocity grid
c----- mq   = maximum number of columns in velocity grid
c----- n    = actual number of rows in velocity grid
c----- m    = actual number of columns in velocity grid
c----- nint = spatial increment between grid rows
c----- mint = spatial increment between grid columns
c-----
c----- nmaxq = maximum number of velocity nodes
c----- mmaxq = maximum number of data rays
c----- maxq  = maximum of nmaxq and mmaxq
c----- nmax  = actual number of velocity nodes
c----- mmax  = actual number of data rays
c----- max   = maximum of nmax and mmax
c-----
c----- vel(nq,mq)= values of velocity at grid points
c-----
c----- ray1travel(nq*mq,6) = array that stores cell,theta0,x,z,t for ray1's path
c----- num                 = number of points in ray1travel
c----- ray2travel(nq*mq,6) = array that stores cell,theta0,x,z,t for ray2's path
c----- num2                = number of points in ray2travel
c-----
c----- i,j,k= temporary integer variables
c----- t1,t2,t3,t4,t5= temporary double-precision variables
c-----
c----- tri   = coordinate indices of the apex of the current triangle
c-----
c----- row   = the row of the current cell
c----- col   = the column of the current cell
c----- trinode1,2,3 = velocity node indices of apices of current triangle
c----- rec1  = integer variable for record numbers
c-----
c----- R(nq*mq) = temporary frechet derivative array
c-----
c-------------------------------------------------------------------------
      real*4 nint,mint
      real*4 ray1travel,ray2travel
      real*4 sloc1,tang1,sloc2,tang2
      real*4 vel
      real*4 datatime,xerr,newxerr
      real*4 R
      real*8 triloc,tmptriloc
      real*8 thetaV,lenV
      real*8 xout,zout,thetaout,timeout
      real*8 finalx1,finalx2,finalz,dtdz1,dxdz1,dtdz2,dxdz2
      real*8 finalv1,finalv2
      real*8 newthV,newx1,newx2,newz,newdxdz1,newdxdz2
      real*8 newdtdz1,newdtdz2
      real*8 t1,t2,t3,t4,t5
      logical triflag,tmptriflag
      integer*4 i,j,k,ndat,ndatq
      integer*4 n,m,dne,nq,mq
      integer*4 index
```

```
      integer*4 nmax,mmax,max,nmaxq,mmaxq,maxq
      integer*4 num,num2
      integer*4 row,col
      integer*4 tri,tmptri
      integer*4 trinode
      integer*4 rec1
      integer*4 p,mult,rem,block,nc

      parameter(nq=90,mq=210)
      parameter(nmaxq=17000,mmaxq=4200,maxq=17000)
      parameter(ndatq=4200)

      dimension vel(nq,mq)
      dimension ray1travel(nq*mq,6),ray2travel(nq*mq,6)
      dimension R(nq*mq)
      dimension tri(3,2),triloc(3,2)
      dimension tmptri(3,2),tmptriloc(3,2)
      dimension trinode(3)
      common /temp/nc
      common /velocity/vel
      common /travel/ray1travel,num,ray2travel,num2
      common /sade/finalx1,finalx2,finalz,dtdz1,dxdz1,dtdz2,dxdz2,
     +             finalv1,finalv2
      common /conparm/n,m,nint,mint,nmax,mmax,max,ndat,dne
c
c----- set constants and initial variables
c
      t1=0.0d0
      t2=0.0d0
      t3=0.0d0
      t4=0.0d0
      t5=0.0d0
c
c----- compute all velocity nodes affected by the rays
c----- begin working along path of ray1
c
      if(num.lt.2) goto 22222
      do 1000 i=1,num-1
         if(ray1travel(i+1,4).ge.finalz) goto 1000
c
c----- first compute the 'tri' nodes affecting this cell
c
         p=int(ray1travel(i,1))
         rem=mod(p,(2*(m-1)))
         mult=(p-rem)/(2*(m-1))
         row=mult+1
         if(rem.eq.0) row=row-1
         if(rem.gt.0) then
            col=int((rem-1)/2.0)+1
         else
            col=m-1
            endif
         if(mod(ray1travel(i,1),2.0e0).eq.0.0e0) then
            tri(1,1)=col
            tri(1,2)=row
            tri(2,1)=col+1
            tri(2,2)=row
            tri(3,1)=col+1
            tri(3,2)=row+1
         else
            tri(1,1)=col
            tri(1,2)=row+1
            tri(2,1)=col+1
            tri(2,2)=row+1
            tri(3,1)=col
            tri(3,2)=row
            endif
         triloc(1,1)=dble((tri(1,1)-1)*mint)
         triloc(1,2)=dble((tri(1,2)-1)*nint)
         triloc(2,1)=dble((tri(2,1)-1)*mint)
         triloc(2,2)=dble((tri(2,2)-1)*nint)
```

```
            triloc(3,1)=dble((tri(3,1)-1)*mint)
            triloc(3,2)=dble((tri(3,2)-1)*nint)
            if(tri(1,2).lt.tri(3,2)) then
               triflag=.true.
            else
               triflag=.false.
               endif
c
c----- compute the corresponding velocity node indices
c
            trinode(1)=(tri(1,2)-1)*m+tri(1,1)
            trinode(2)=(tri(2,2)-1)*m+tri(2,1)
            trinode(3)=(tri(3,2)-1)*m+tri(3,1)
c
c----- compute derivatives for these nodes and add to R matrix
c
c----- first re-trace through cell with perturbed velocity to find
c----- the effect on exitangle and time
c----- do this for each of the 3 trinodes
c
c----- compute parameters of the vector V
            t1=dble(ray1travel(i+1,3))
            t3=dble(ray1travel(i+1,4))
            thetaV=datan((finalx1-t1)/(finalz-t3))
            lenV=dsqrt((finalx1-t1)**2+(finalz-t3)**2)
            do 1010 k=1,3
               tmptri(1,1)=tri(1,1)
               tmptri(1,2)=tri(1,2)
               tmptri(2,1)=tri(2,1)
               tmptri(2,2)=tri(2,2)
               tmptri(3,1)=tri(3,1)
               tmptri(3,2)=tri(3,2)
               tmptriloc(1,1)=triloc(1,1)
               tmptriloc(1,2)=triloc(1,2)
               tmptriloc(2,1)=triloc(2,1)
               tmptriloc(2,2)=triloc(2,2)
               tmptriloc(3,1)=triloc(3,1)
               tmptriloc(3,2)=triloc(3,2)
               tmptriflag=triflag
               t1=dble(ray1travel(i,3))
               t2=dble(ray1travel(i,2))
               t3=dble(ray1travel(i,4))
               t4=dble(ray1travel(i,5))
               t5=0.0d0
               vel(tri(k,2),tri(k,1))=vel(tri(k,2),tri(k,1))*1.0025
               call raytrace(t1,t2,t3,t4,tmptriflag,tmptri,tmptriloc)
               vel(tri(k,2),tri(k,1))=vel(tri(k,2),tri(k,1))/1.0025
               xout=t1
               zout=t3
               thetaout=t2
               timeout=t4
c----- adjust thetaV and compute new end point
               t1=(thetaout-dble(ray1travel(i+1,2)))
               newthV=thetaV+t1
               newx1=xout+lenV*dsin(newthV)
               newz=zout+lenV*dcos(newthV)
c----- compute new deriv's and allow for time/depth correction and
c----- then compute partial frechet deriv for this node
               newdxdz1=dtan(datan(dxdz1)+t1)
               if(dabs(1/(finalv1*dtdz1)).gt.1.0d0) then
                  newdtdz1=1/(finalv1*dcos(t1))
               else
                  newdtdz1=1/(finalv1*dcos(dacos(1/(finalv1*dtdz1))+t1))
                  endif
               t5=dsqrt((xout-dble(ray1travel(i+1,3)))**2+(zout-
     +                        dble(ray1travel(i+1,4)))**2)
               if(t5.gt.1.0d0) then
                  newz=newz+(timeout-ray1travel(i+1,5))/newdtdz1
                  endif
               t2=(newz-finalz)*dtdz2+(timeout-ray1travel(i+1,5))
               newx2=(newz-finalz)*dxdz2+finalx2
```

```
            t3=-1.0d0*t2/(newdtdz1+dtdz2)
            t4=((t3*dxdz2)+newx2)-((t3*newdxdz1)+newx1)
            if(dabs(t4-dble(xerr)).lt.10.0d0)
     +      R(trinode(k))=R(trinode(k))+sngl(t4-dble(xerr))/
     +                        (vel(tri(k,2),tri(k,1))*0.0025)
            if(dabs(t4-dble(xerr)).ge.10.0d0) nc=nc+1
1010        continue
1000      continue
c
c----- now do the same for ray2
c
22222  if(num2.lt.2) goto 99999
       do 2000 i=1,num2-1
          if(ray2travel(i+1,4).ge.finalz) goto 2000
c
c----- first compute the 'tri' nodes affecting this cell
c
          p=int(ray2travel(i,1))
          rem=mod(p,(2*(m-1)))
          mult=(p-rem)/(2*(m-1))
          row=mult+1
          if(rem.eq.0) row=row-1
          if(rem.gt.0) then
             col=int((rem-1)/2.0)+1
          else
             col=m-1
             endif
          if(mod(ray2travel(i,1),2.0e0).eq.0.0e0) then
             tri(1,1)=col
             tri(1,2)=row
             tri(2,1)=col+1
             tri(2,2)=row
             tri(3,1)=col+1
             tri(3,2)=row+1
           else
             tri(1,1)=col
             tri(1,2)=row+1
             tri(2,1)=col+1
             tri(2,2)=row+1
             tri(3,1)=col
             tri(3,2)=row
             endif
          triloc(1,1)=dble((tri(1,1)-1)*mint)
          triloc(1,2)=dble((tri(1,2)-1)*nint)
          triloc(2,1)=dble((tri(2,1)-1)*mint)
          triloc(2,2)=dble((tri(2,2)-1)*nint)
          triloc(3,1)=dble((tri(3,1)-1)*mint)
          triloc(3,2)=dble((tri(3,2)-1)*nint)
          if(tri(1,2).lt.tri(3,2)) then
             triflag=.true.
          else
             triflag=.false.
             endif
c
c----- compute the corresponding velocity node indices
c
          trinode(1)=(tri(1,2)-1)*m+tri(1,1)
          trinode(2)=(tri(2,2)-1)*m+tri(2,1)
          trinode(3)=(tri(3,2)-1)*m+tri(3,1)
c
c----- compute derivatives for these nodes and add to R matrix
c
c----- first re-trace through cell with perturbed velocity to find
c----- the effect on exitangle and time
c----- do this for each of the 3 trinodes
c
c----- compute parameters of the vector V
          t1=dble(ray2travel(i+1,3))
          t3=dble(ray2travel(i+1,4))
          thetaV=datan((finalx2-t1)/(finalz-t3))
          lenV=dsqrt((finalx2-t1)**2+(finalz-t3)**2)
```

```
            do 2010 k=1,3
               tmptri(1,1)=tri(1,1)
               tmptri(1,2)=tri(1,2)
               tmptri(2,1)=tri(2,1)
               tmptri(2,2)=tri(2,2)
               tmptri(3,1)=tri(3,1)
               tmptri(3,2)=tri(3,2)
               tmptriloc(1,1)=triloc(1,1)
               tmptriloc(1,2)=triloc(1,2)
               tmptriloc(2,1)=triloc(2,1)
               tmptriloc(2,2)=triloc(2,2)
               tmptriloc(3,1)=triloc(3,1)
               tmptriloc(3,2)=triloc(3,2)
               tmptriflag=triflag
               t1=dble(ray2travel(i,3))
               t2=dble(ray2travel(i,2))
               t3=dble(ray2travel(i,4))
               t4=dble(ray2travel(i,5))
               t5=0.0d0
               vel(tri(k,2),tri(k,1))=vel(tri(k,2),tri(k,1))*1.0025
               call raytrace(t1,t2,t3,t4,tmptriflag,tmptri,tmptriloc)
               vel(tri(k,2),tri(k,1))=vel(tri(k,2),tri(k,1))/1.0025
               xout=t1
               zout=t3
               thetaout=t2
               timeout=t4
c----- adjust thetaV and compute new end point
               t1=(thetaout-dble(ray2travel(i+1,2)))
               newthV=thetaV+t1
               newx2=xout+lenV*dsin(newthV)
               newz=zout+lenV*dcos(newthV)
c----- compute new deriv's and allow for time/depth correction and
c----- then compute partial frechet deriv for this node
               newdxdz2=dtan(datan(dxdz2)+t1)
               if(dabs(1/(finalv2*dtdz2)).gt.1.0d0) then
                  newdtdz2=1/(finalv2*dcos(t1))
               else
                  newdtdz2=1/(finalv2*dcos(dacos(1/(finalv2*dtdz2))+t1))
                  endif
               t5=dsqrt((xout-dble(ray2travel(i+1,3)))**2+(zout-
     +                        dble(ray2travel(i+1,4)))**2)
               if(t5.gt.1.0d0) then
                  newz=newz+(timeout-ray2travel(i+1,5))/newdtdz2
                  endif
               t2=(newz-finalz)*dtdz1+(timeout-ray2travel(i+1,5))
               newx1=(newz-finalz)*dxdz1+finalx1
               t3=-1.0d0*t2/(newdtdz2+dtdz1)
               t4=((t3*newdxdz2)+newx2)-((t3*dxdz1)+newx1)
               if(dabs(t4-dble(xerr)).lt.10.0d0)
     +         R(trinode(k))=R(trinode(k))+sngl(t4-dble(xerr))/
     +                              (vel(tri(k,2),tri(k,1))*0.0025)
               if(dabs(t4-dble(xerr)).ge.10.0d0) nc=nc+1
2010           continue
2000        continue

99999    return
         end
```

```
@PROCESS DC(GAUS),OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS DIRECTIVE('*VD:')
c-------------------------------------------------------------------------------
c----- subroutine to compute the weights based on the Xerr's for each ray -----
c----- currently the Xerr threshold is 300 metres                          -----
c-------------------------------------------------------------------------------
      subroutine varian(F)
c-------------------------------------------------------------------------------
c-----
c----- the preliminary comments in METOMOS6 describe the variables in this
c----- subroutine
c-----
c-------------------------------------------------------------------------------
      real*4 F,sig,ai,bi,XXerr,XXerr2
      real*4 pi,pi6,pi18,t1,ts,tc
      real*4 nint,mint
      integer*4 ndat,ndatq
      integer*4 i,j,k,n,m,nmax,mmax,max,dne

      parameter(ndatq=4200)

      dimension F(ndatq),sig(ndatq)
      dimension ai(ndatq),bi(ndatq),XXerr(ndatq),XXerr2(ndatq)
      common /var/sig,ai,bi,XXerr,XXerr2
      common /conparm/n,m,nint,mint,nmax,mmax,max,ndat,dne
c
c----- set constants and initial variables
c
      pi=4.0*atan(1.0)
      pi6=-1.0*pi/600.0
      pi18=-1.0*(pi**2)/180000.0
c
c-----compute the sig(k) values using a modified Hanning window function
c-----also compute the ai's, bi's, and set Xerr and Xerr2 (see T.N.26)
c
      do 100 k=1,ndat
         t1=pi*F(k)/300.0
         ts=sin(t1)
         tc=cos(t1)
         ai(k)=pi6*ts*(0.75+0.25*tc)
         bi(k)=pi18*(0.75*tc+0.25*tc**2-0.25*ts**2)
         XXerr(k)=F(k)
         XXerr2(k)=F(k)**2
         if(F(k).ge.300.0) then
            sig(k)=0.25
            ai(k)=0.0
            bi(k)=0.0
            goto 100
            endif
         sig(k)=0.75+0.25*tc
         sig(k)=sig(k)**2
100      continue

      return
      end
```

```
@PROCESS DC(GAUS),OPT(3),VECTOR(LEVEL(2),REPORT(XLIST)),
@PROCESS DIRECTIVE('*VD:')
c-------------------------------------------------------------------------
c----- subroutine to trace two rays through medium divided into triangles -----
c----- both traces have known starting locations and initial angles       -----
c----- traces both rays through depth levels until data time is exceeded  -----
c----- so that Chuck Sword's 'xerr' statistic can be calculated           -----
c----- all real calculations in double-precision                          -----
c-------------------------------------------------------------------------
      subroutine xerrrt(sloc1,tang1,sloc2,tang2,datatime,xerr,ray)
c-------------------------------------------------------------------------
c----- sloc1 = source location of first ray
c----- tang1 = take-off angle in radians (positive angle implies
c-----         in direction of increasing distance) of first ray
c----- sloc2 = source location of second ray
c----- tang2 = take-off angle in radians (positive angle implies
c-----         in direction of increasing distance) of second ray
c----- datatime  = actual travel time of configured ray
c----- xerr  = Chuck Sword's statistic
c-----
c----- nq   = maximum number of rows in velocity grid
c----- mq   = maximum number of columns in velocity grid
c----- n    = actual number of rows in velocity grid
c----- m    = actual number of columns in velocity grid
c----- nint = spatial increment between grid rows
c----- mint = spatial increment between grid columns
c-----
c----- vel(nq,mq)= values of velocity at grid points
c-----
c----- ray1travel(n*m,6) = array that stores cell,theta0,x,z,t for ray1's path
c----- num               = number of points in ray1travel
c----- ray2travel(n*m,6) = array that stores cell,theta0,x,z,t for ray2's path
c----- num2              = number of points in ray2travel
c-----
c----- raylocs = array containing locations of first ray for graphing
c----- raylocs2= array containing locations of second ray for graphing
c-----
c----- i,j,k= temporary integer variables
c----- t1,t2,t3,t4,t5= temporary double-precision variables
c-----
c----- storage parameters:
c----- ray1x,ray2x = x values
c----- ray1z,ray2z = z values
c----- ray1time,ray2time = time values
c----- ray1theta0,ray2theta0 = theta0 values
c----- ray1tri(3,2),ray2tri(3,2) = tri values
c----- ray1triloc(3,2),ray2triloc(3,2) = triloc values
c----- ray1triflag,ray2triflag = triflag values
c-----
c----- ray1exit  = .true. if ray1 has exited the model area
c-----
c----- oldtime   = total time at last level
c----- oldxerr   = xerr at last level
c----- oldray1x  }
c----- oldray2x  }
c----- oldz      }----- parameters of last level; used in computations
c----- oldtime1  }                                 for sade2
c----- oldtime2  }
c-----
c-------------------------------------------------------------------------
      real*4 sloc1,tang1
      real*4 sloc2,tang2
      real*4 datatime,xerr
      real*4 nint,mint
      real*4 vel
      real*8 ray1x,ray2x,ray1z,ray2z
      real*8 ray1time,ray2time
      real*8 ray1triloc,ray2triloc
      real*8 ray1theta0,ray2theta0
      real*4 ray1travel,ray2travel
      real*8 oldtime,oldxerr
```

```
      real*8 oldray1x,oldray2x,oldz,oldtime1,oldtime2
      real*8 finalx1,finalx2,finalz,dtdz1,dxdz1,dtdz2,dxdz2
      real*8 finalv1,finalv2
      real*8 t1,t2,t3,t4,t5
      logical ray1triflag,ray2triflag
      logical ray1exit
      logical raybad
      integer*4 i,j,k
      integer*4 ray,ndat,nmax,mmax,max,dne
      integer*4 n,m,nq,mq,ndatq
      integer*4 ray1tri,ray2tri
      integer*4 num,num2
      integer*4 numnull,numturn,iterat

      parameter(nq=90,mq=210)
      parameter(ndatq=4200)

      dimension vel(nq,mq)
      dimension raybad(ndatq)
      dimension raylocs(2,nq*mq)
      dimension raylocs2(2,nq*mq)
      dimension ray1tri(3,2),ray2tri(3,2)
      dimension ray1triloc(3,2),ray2triloc(3,2)
      dimension ray1travel(nq*mq,6),ray2travel(nq*mq,6)
      common /velocity/vel
      common /iterat/iter
      common /travel/ray1travel,num,ray2travel,num2
      common /sade/finalx1,finalx2,finalz,dtdz1,dxdz1,dtdz2,dxdz2,
     +             finalv1,finalv2
      common /lgxerr/raybad
      common /nums/numnull,numturn
      common /conparm/n,m,nint,mint,nmax,mmax,max,ndat,dne
c
c----- set constants and initial variables
c
      t1=0.0d0
      t2=0.0d0
      t3=0.0d0
      t4=0.0d0
      t5=0.0d0
      ray1exit=.false.
      num=0
      num2=0
      do 10 i=1,n*m
         ray1travel(i,1)=0.0e0
         ray2travel(i,1)=0.0e0
10       continue
c
c----- put parameters for first ray into storage
c
      ray1x=dble(sloc1)
      ray1theta0=dble(tang1)
      ray1z=0.0d0
      ray1time=0.0d0
      ray1triflag=.true.
      ray1tri(1,1)=dint(dble(sloc1/mint))+1
      ray1tri(1,2)=1
      ray1tri(2,1)=ray1tri(1,1)+1
      ray1tri(2,2)=1
      ray1tri(3,1)=ray1tri(2,1)
      ray1tri(3,2)=2
      ray1triloc(1,1)=dble(ray1tri(1,1)-1)*dble(mint)
      ray1triloc(1,2)=0.0d0
      ray1triloc(2,1)=ray1triloc(1,1)+dble(mint)
      ray1triloc(2,2)=0.0d0
      ray1triloc(3,1)=ray1triloc(2,1)
      ray1triloc(3,2)=dble(nint)
c
c----- put parameters for second ray into storage
c
      ray2x=dble(sloc2)
```

```
      ray2theta0=dble(tang2)
      ray2z=0.0d0
      ray2time=0.0d0
      ray2triflag=.true.
      ray2tri(1,1)=dint(dble(sloc2/mint))+1
      ray2tri(1,2)=1
      ray2tri(2,1)=ray2tri(1,1)+1
      ray2tri(2,2)=1
      ray2tri(3,1)=ray2tri(2,1)
      ray2tri(3,2)=2
      ray2triloc(1,1)=dble(ray2tri(1,1)-1)*dble(mint)
      ray2triloc(1,2)=0.0d0
      ray2triloc(2,1)=ray2triloc(1,1)+dble(mint)
      ray2triloc(2,2)=0.0d0
      ray2triloc(3,1)=ray2triloc(2,1)
      ray2triloc(3,2)=dble(nint)
c
c----- load initial points into graphing array and "raytravel's"
c
      num=1
      ray1travel(1,1)=float(ray1tri(1,1)*2+(ray1tri(1,2)-1)*(m-1)*2)
      ray1travel(1,2)=tang1
      ray1travel(1,3)=sloc1
      ray1travel(1,4)=0.0e0
      ray1travel(1,5)=0.0e0
      ray1travel(1,6)=0.0e0
      num2=1
      ray2travel(1,1)=float(ray2tri(1,1)*2+(ray2tri(1,2)-1)*(m-1)*2)
      ray2travel(1,2)=tang2
      ray2travel(1,3)=sloc2
      ray2travel(1,4)=0.0e0
      ray2travel(1,5)=0.0e0
      ray2travel(1,6)=0.0e0
c
c----- trace ray1 through to the next level
c
1000  oldtime=ray1time+ray2time
      oldxerr=ray2x-ray1x
      oldray1x=ray1x
      oldray2x=ray2x
      oldz=ray1z
      oldtime1=ray1time
      oldtime2=ray2time
11111 call raytrace(ray1x,ray1theta0,ray1z,ray1time,ray1triflag,
     + ray1tri,ray1triloc)
c
c----- load results into 'ray1travel' memory array and graphing array
c
      num=num+1
      if(num.gt.n*m) goto 88888
      i=0
      if(.not.ray1triflag) i=2*(m-1)+1
      ray1travel(num,1)=float(ray1tri(1,1)*2+(ray1tri(1,2)-1)*
     +                                              (m-1)*2-i)
      ray1travel(num,2)=sngl(ray1theta0)
      ray1travel(num,3)=sngl(ray1x)
      ray1travel(num,4)=sngl(ray1z)
      ray1travel(num,5)=sngl(ray1time)
      ray1travel(num,6)=0.0d0
c
c----- check to see if ray1 has left the model
c----- if not check to see if the next level has been reached
c
      if((ray1tri(1,1).lt.1).or.(ray1tri(2,1).gt.m).or.
     +   (ray1tri(3,2).lt.1)) then
         ray1exit=.true.
         xerr=0.0e0
         return
         endif
      if(ray1z.le.ray1travel(num-1,4)) then
         numturn=numturn+1
```

```
            xerr=0.0e0
            return
            endif
         if(dmod(ray1z,dble(nint)).ne.0.0d0) goto 11111
c
c----- trace ray2 through to the next level
c
22222    call raytrace(ray2x,ray2theta0,ray2z,ray2time,ray2triflag,
     + ray2tri,ray2triloc)
c
c----- load results into 'ray2travel' memory array and graphing array
c
         num2=num2+1
         if(num2.gt.n*m) goto 88888
         i=0
         if(.not.ray2triflag) i=2*(m-1)+1
         ray2travel(num2,1)=float(ray2tri(1,1)*2+(ray2tri(1,2)-1)*
     +                                                  (m-1)*2-i)
         ray2travel(num2,2)=sngl(ray2theta0)
         ray2travel(num2,3)=sngl(ray2x)
         ray2travel(num2,4)=sngl(ray2z)
         ray2travel(num2,5)=sngl(ray2time)
         ray2travel(num2,6)=0.0d0
c
c----- check to see if ray2 has left the model
c----- if not check to see if the next level has been reached
c
         if((ray2tri(1,1).lt.1).or.(ray2tri(2,1).gt.m).or.
     +    (ray2tri(3,2).lt.1)) then
            xerr=0.0e0
            return
            endif
         if(ray2z.le.ray2travel(num2-1,4)) then
            numturn=numturn+1
            xerr=0.0e0
            return
            endif
         if(dmod(ray2z,dble(nint)).ne.0.0d0) goto 22222
         if(ray1exit) goto 99999
c
c----- check to see if the data time has been surpassed
c
         if((ray1time+ray2time).ge.dble(datatime)) goto 99999
c
c----- check to see if the last level has been reached
c----- otherwise loop back to trace to next level
c
         if(idnint(ray1z/dble(nint)).eq.(n-1)) then
            xerr=0.0e0
            return
            endif
         goto 1000
c
c----- if n*m intersection points exists, write to screen and output zero
c
88888    write(2,'(2x,a40)') '******* number of int points exceeded  '
         xerr=0.0
         return
c
c----- calculate 'xerr' (linearly interpolating between levels)
c
99999    t1=ray2x-ray1x
         t2=ray1time+ray2time
         t3=((dble(datatime)-oldtime)/(t2-oldtime))
         t4=t3*(t1-oldxerr)
         xerr=sngl(oldxerr+t4)
         if(abs(xerr).gt.350.0) then
            xerr=0.0
            raybad(ray)=.true.
            numnull=numnull+1
            return
```

```
          endif
c
c----- compute z, final x's and the four derivatives
c
       finalx1=oldray1x+(ray1x-oldray1x)*t3
       finalx2=oldray2x+(ray2x-oldray2x)*t3
       finalz=oldz+t3*dble(nint)
       dtdz1=(ray1time-oldtime1)/nint
       dxdz1=(finalx1-oldray1x)/(t3*nint)
       dtdz2=(ray2time-oldtime2)/nint
       dxdz2=(finalx2-oldray2x)/(t3*nint)
       finalv1=dmax1(dble(vel(ray1tri(1,2),ray1tri(1,1))),
     +               dble(vel(ray1tri(2,2),ray1tri(2,1))),
     +               dble(vel(ray1tri(3,2),ray1tri(3,1))))
       finalv2=dmax1(dble(vel(ray2tri(1,2),ray2tri(1,1))),
     +               dble(vel(ray2tri(2,2),ray2tri(2,1))),
     +               dble(vel(ray2tri(3,2),ray2tri(3,1))))

       return
       end
```