RECORD 1996/3

SEISMIC STREAMER POSITIONING AND WATER DEPTH COMPUTATION AT RECEIVER LOCATIONS - AGSO MARINE DATA PROCESSING

BY

R. PARUMS

FEBRUARY 1996

AGSO
AUSTRALIAN GEOLOGICAL SURVEY ORGANISATION

RECORD 1996/3

SEISMIC STREAMER POSITIONING AND WATER DEPTH COMPUTATION AT RECEIVER LOCATIONS - AGSO MARINE DATA PROCESSING

BY

R. PARUMS

FEBRUARY 1996

ISSN: 1039-0073
ISBN: 0 642 22394 7

Acknowledgements

I would like to acknowledge Peter Petkovic, Mike Sexton, Roy Whitworth and Norm Johnston for ideas and contributions during this project. I would also like to thank Peter Petkovic and Keith Lockwood for reviewing the text.

Contents

Introduction

During April and May 1995, the Australian Geological Survey Organisation (AGSO) acquired multichannel marine seismic data for MIM Petroleum in the WA-235P (Kununga) and WA-238P (Cognac) Permit Blocks in the Timor Sea and North Western Australia (AGSO, 1995a,b). The survey vessel used was the R.V. Rig Seismic on which seismic data were acquired and water depths, gravity and magnetics also collected. In the Kununga survey, 2043 kms of data were acquired in a grid over 129 seismic lines with parallel line spacings of between 400 metres and 1 km. The Cognac survey acquired 611 kms of data over 24 lines with variable line spacings of between 500 metres and 2 kms in the predominant shooting direction. The AGSO survey numbers for these two surveys were S160 and S161 respectively.

The location of the seismic streamer was required as a check of the acquisition parameters and to assess the likely effect of feathering on the final seismic data in the processing and interpretation phases of the project. The final processed streamer data were displayed in map form and supplied as a digital file containing the location of each receiver in the seismic streamer and its associated water depth value for each shot in the survey.

This record describes the processes undertaken to produce the final streamer positioning file, including water depths at each shot and receiver position, from raw field data. It concludes by outlining a qualitative scheme for the integration of the receiver positioning and water depth data into the seismic processing stream, to aid in the correction for the effect of irregular water bottom topography and near-surface velocity inhomogoneities on the timing of deep seismic reflections.

Navigation Processing

The ship navigation was controlled by a differential Global Positioning System (DGPS) with an independent DGPS as backup. Navigation data and some geophysical data were captured by the Data Acquisition System (DAS) where the data were used for real-time navigation and then written to tape. The navigation was also used to control the seismic shot interval. Data from the differential GPS and other instruments were sampled every second and written to tape in 100-channel binary format. The DAS was hosted on a MicroVax 2 computer with the software running under the VMS Version 5.5-1 operating system. Processing of non-seismic data was conducted at AGSO in Canberra mainly on VAX computers using FORTRAN programs running under the VMS operating system.

The following section describes the major processing steps routinely undertaken to obtain navigation at each shot location in a seismic survey. All processes produce a time-based binary file as output.

Navigation Data

- Navigation and other data were resampled to every 10 seconds.
- Both sets of navigation data were interpolated to the 10 second mark after accounting for an instrument-induced time lag between time of satellite fix and the time of record as written to the DAS input buffer.
- Data were despiked with a median filter after choosing a suitable window length and threshold value.
- Data were interpolated over small time gaps (up to 3 minutes).
- An interactive PC-based editor was used to patch the navigation data between the two DGPS navigation systems to give a final data set.
- Navigation data were smoothed using a sinc function[1] filter.

At this stage of the processing, the file contained final processed navigation in binary format. Appendix 1 describes the full processing sequence of the Non-Seismic Processing (NSP) programs which were used. Figure 1 is an example of raw and final processed navigation data.

Shot-Time Data

- A shot-times file was edited to first and last good chargeable shots, missing shots were interpolated and duplicated shot numbers omitted.
- The shot-times file was delayed to the time at which the ship's antenna was at the seismic stack point (normally the source or the point midway between the source and first channel). The delay was calculated by determining the speed of the ship at the time of the shot from the navigation file and using it to compute the time taken to travel the fixed distance from the stack point to the antenna position. Shotpoint positions were required at the source for the Mimpex surveys.
- Merge lineparts, rename lines and resequence shotpoints as required.

[1] A spatial filter of the form sinx/x multiplied with a Hanning taper.

Source positions

- The processed shot-times file was combined with final time-based navigation to produce position for each shot (at the seismic stack point) in modified UKOOA format (UKOOA, 1990). The time of each shotpoint was matched with the nearest time in the navigation file and a position for the stack point was determined by interpolating between the positions of the nearest two times. Appendix 7 describes the modified UKOOA format adopted by AGSO Non-Seismic Processing.

The final navigation UKOOA file contained linename, shot, latitude and longitude. Latitude and longitude values were produced to 2 decimal places of an arc second, which was a precision of 0.3 of a metre.

Water Depth Processing

Water depth data were recorded on 2 separate instruments - a 3.5 KHz and a 12 KHz Raytheon CESP Echo Sounder. The data were recorded on analogue charts and digitally at 1 second sample rate for both instruments. Processing began by first choosing digital data from the instrument which gave the clearest response and using this as the base data set. Digital or analogue data from the second instrument were used to supplement the primary water depth data where they were poor. The processing stream that was adopted is as follows (for a more detailed summary see Appendix 2).

- Water depth values with error flags were omitted.
- Gaps in the data were interpolated (up to 2 minutes).
- The data were resampled to 10 second sample interval.
- The data were interactively edited on a PC. The water depths were inspected and obvious errors were corrected. Common sources of error in water depth data during the survey were (i) the generation of spurious values when the instrument lost tracking of the water bottom due to rough or steeply dipping sea bottom topography and (ii) sub-bottom profiling where the system preferentially tracked an event below the true water bottom. Other phenomena which were manifest were dropouts in the signal and 'cloudy' water bottoms, both of which may be related to poor weather conditions.
- The analogue charts were digitised over areas where the digital data were poor. The digitised values were then used to control the processing of the digital data.
- A correction was applied for transducer stem depth.

Figure 2 shows an example of the difference between raw water depth data and final processed water depth. Most of the noise has been eliminated in an interactive editor after inspection of the analogue chart records.

At this stage, water depths were added to the binary navigation file and a UKOOA file was regenerated which contained final water depths.

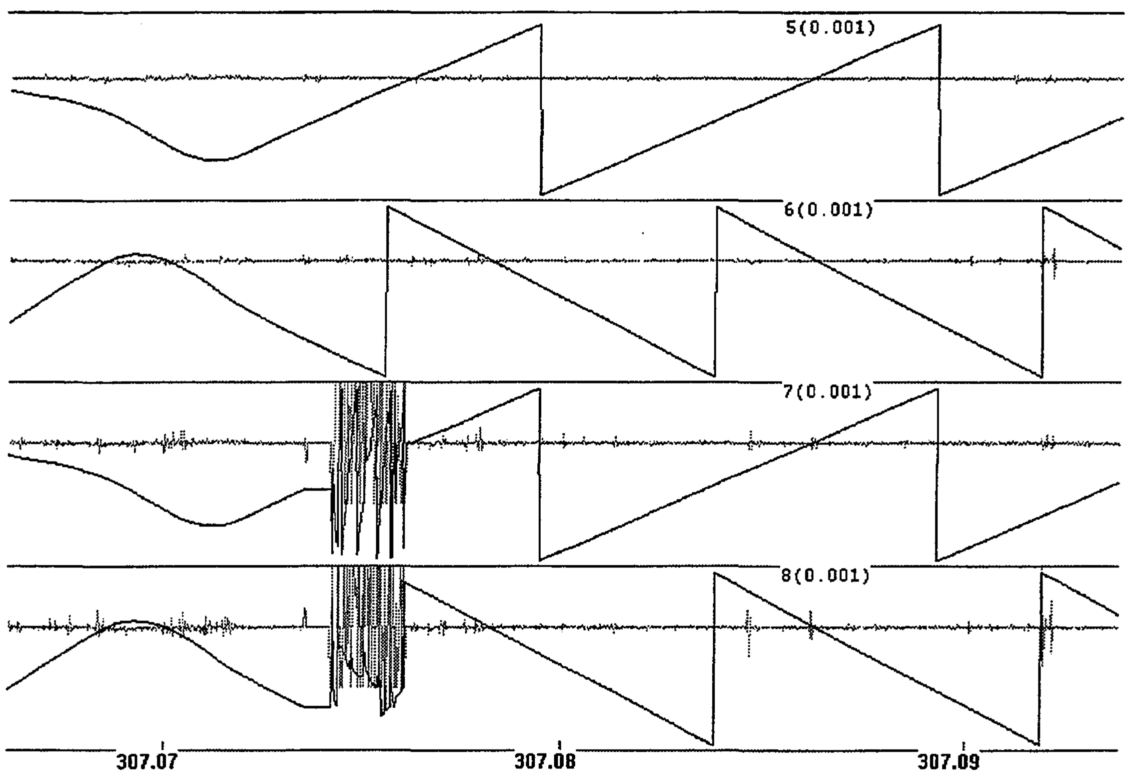

Figure 1: An example of final latitude and longitude data in radians (channels 5,6) after processing of the raw data (channels 7,8). Scale width is 1.0e-3. Data shown in grey are second difference plots (scale width 2.0e-6).

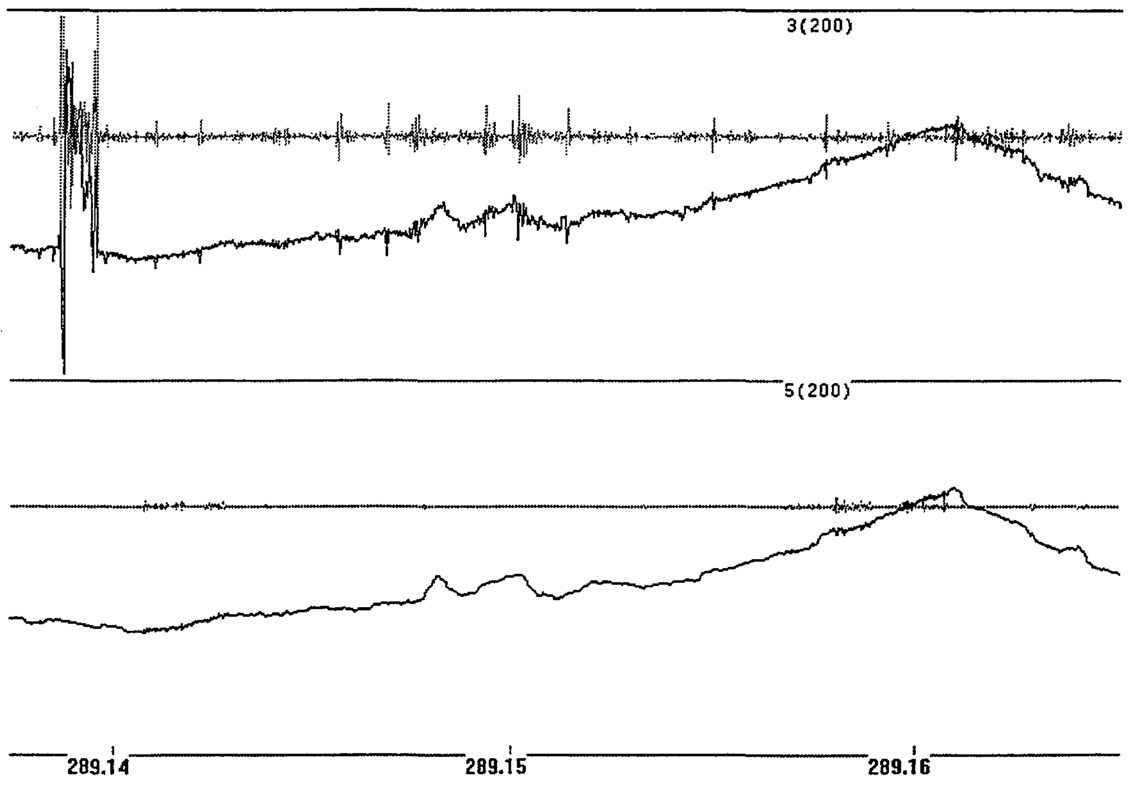

Figure 2: An example of raw water depths (channel 3) and processed water depths (channel 5). Scale width is 200m. Data shown in grey are second difference plots (scale width 100m).

Seismic Streamer Position Processing

The raw data used to process the seismic streamer position were the tailbuoy, compass, course and heading information. Tailbuoy data were derived from a GPS system mounted on the tailbuoy. These positions are always less accurate than the DGPS system, having an error of about 30-100m, whereas DGPS positioning error is quoted at 5m. Compass data were provided by 5 Fluxgate Vector Magnetometers located at points along the cable with heading information measured by a Sperry Mk 37 Gyrocompass. During acquisition, the course and ship speed were determined at each data sample from the position at that instant and the position 10 seconds earlier.

The following major processing steps were carried out to produce final positions for each receiver in the seismic streamer for each shot. Appendix 3 describes the streamer processing stream while Appendix 4 discusses errors in receiver locations and presents a review of recent industry techniques to determine the streamer positioning.

Tailbuoy data

- Data were despiked using a median filter.
- Time gaps of up to 10 minutes in the data were interpolated.
- A PC-based editor was used to remove spurious values and noise bursts.
- The data were smoothed with a sinc function filter.

Compass, Heading and Course data

- Compass, heading and course data were converted from modulus 360[2] values to a continuous function.
- The data were despiked.
- Gaps of up to 10 minutes were interpolated.
- Data were interactively edited to remove obvious errors.
- Data were converted back to modulus 360 values.

At this stage, tailbuoy, compass, course and heading were merged into one binary file along with final navigation in channels 3 and 4. Figure 3(a) shows an example of the compass data before and after processing.

Compass location data

- The location of each compass was computed by assuming that the streamer sections between compasses were rigid. Output was in dx and dy values in metres, with the origin defined at the rear of the ship, the y-axis in the sail-line direction (course) and the x-axis orthogonal to the y-axis. Dx values were positive when the cable drifted to starboard.
- The compass data were despiked and smoothed (See Figure 3(b)).

[2]Angle values were represented in the DAS as numbers between 0 and 359 degrees by convention. These data were in a discontinuous functional form which were unsuitable as input into processes such as despiking and smoothing.

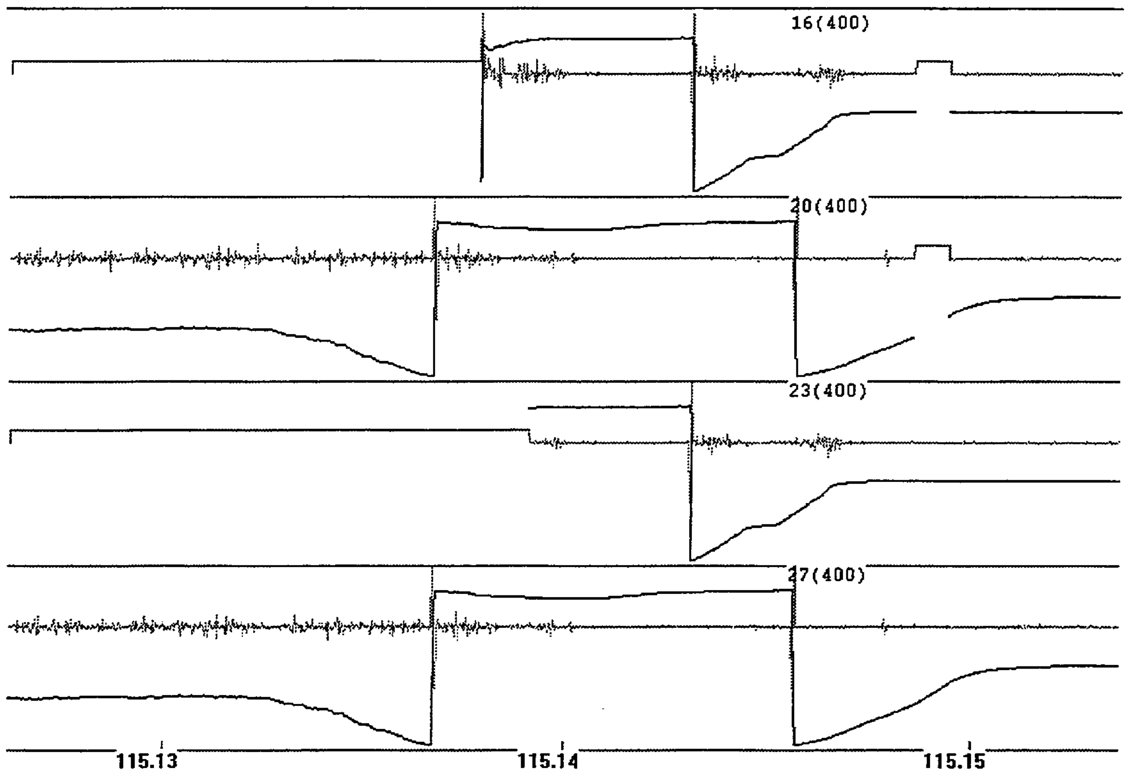

Figure 3(a): Example of raw compass data (channels 16,20) and after despiking and interactive editing (channels 23,27). Scale width is 400°. Data in grey are second difference plots (scale width 10°).

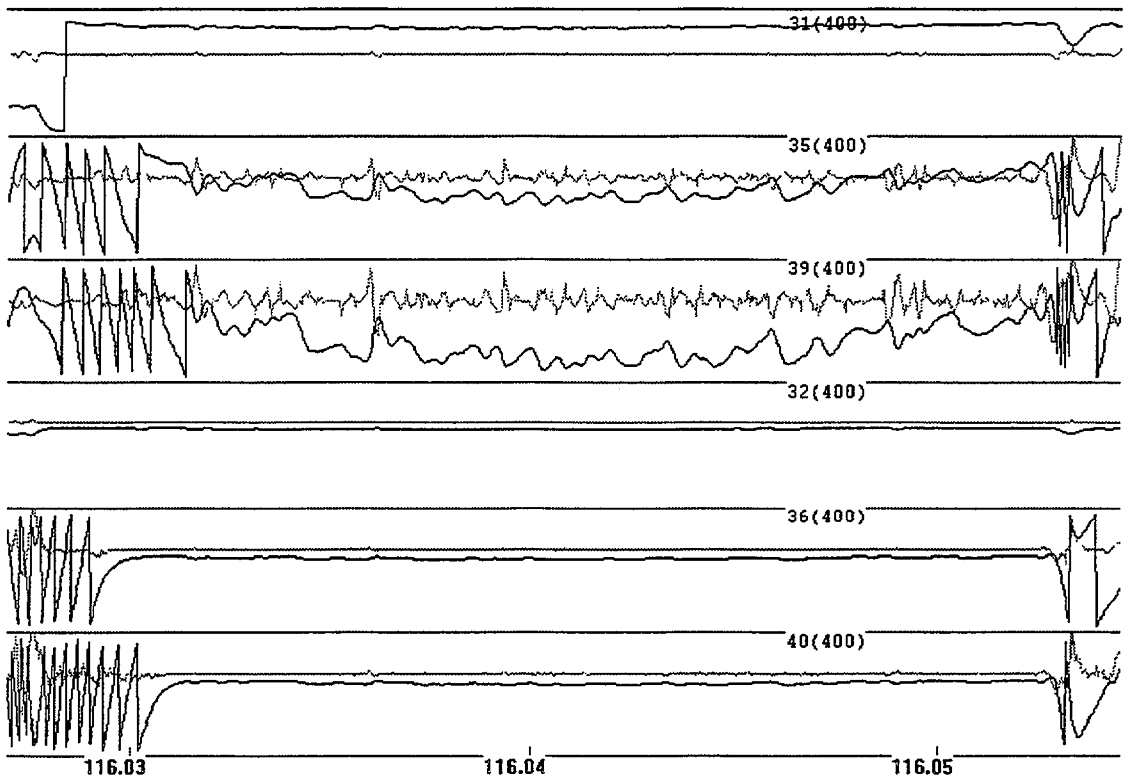

Figure 3(b): Example of processed compass locations after conversion to dx values (chans 31,35,39) and dy values (chans 32,36,40). Scale width is 400m. The wild variations before time 116.0312 and after 116.0515 were due to the cable turning on to and off a seismic line. The dx data showed rapid fluctuations as would be expected whenever the ship's course altered slightly.

- The dx, dy values were converted to latitude and longitude coordinates. An attempt was made to correct for the limited accuracy of the GPS receiver on the tailbuoy by subtracting the DGPS - GPS distance vector[3] from the tailbuoy measurement. This was unsuccessful because the two GPS systems quite often had different instantaneous satellite configurations and thus different measurement errors.

At this point in the processing, a 'non-delayed' shot-times file was combined with the compass, tailbuoy and ship position data to produce an ASCII file containing the locations of ship's antenna, compasses and tailbuoy.

Source, Compass and Tailbuoy Position data

- Data were output into an ASCII file containing the locations of ship's antenna, compasses and tailbuoy for each shot. Appendix 8 lists the format of the compass location file.
- Latitude and longitude values at the seismic stack point were inserted (from the final modified UKOOA file) into the compass location file.
- The compasses were adjusted to the tailbuoy location by rotation, in order to minimise the error in the streamer positioning (Figure 4). The average distance (over all shots in the Kununga survey) that compass 5 needed to be shifted was 31m.
- Data were converted to AGD84 datum.
- Records with bad inter-compass distances were omitted. A bad record was deemed to occur when any inter-compass distance differed by more than 30 metres from its physical length.
- Missing records at the start or end of lines were extrapolated from the nearest good pair of records in the line.

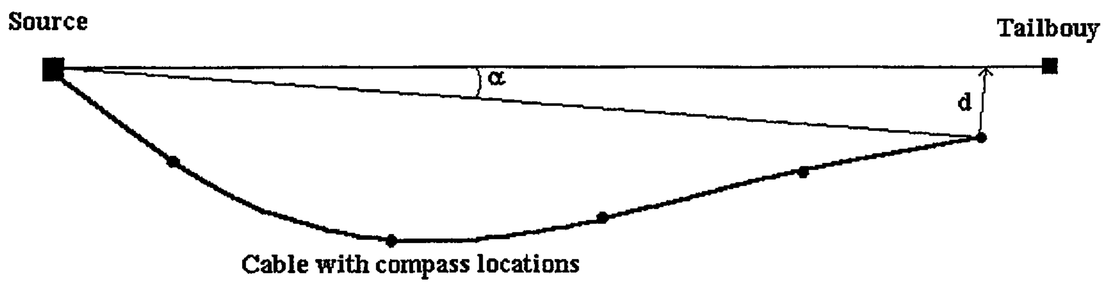

Figure 4: All 5 compass locations are rotated by angle α around the source location. The rotated distance, d is represented diagrammatically.

Shot-Receiver locations

- The compass location file was used as input to generate a file of shot-receiver locations by interpolation between compass locations. The program also interpolated shot and receiver locations over missing shotpoints in the input file.
- Streamer locations were plotted from the shot/receiver file.

[3] The vector distance between the shipboard GPS and the Racal DGPS positions.

Final processed streamer locations were presented as Kununga Maps 1-3 and Cognac Map 1. These data are confidential. Streamer locations are represented at the instant the shot was fired at every 10th shot for the Kununga survey and every 20th shot for Cognac.

Final shot/receiver positioning data were output as an ASCII file in Source-Receiver Positioning Format (SRPF) with a record per receiver. Table 1 lists the SRP format. A receiver number of 1 was assigned to the source and 2 to 241 were assigned to each of the receivers in the cable, where 241 refers to the receiver furthest from the ship.

COLUMNS	NAME	FORMAT
1 - 10	Linename	A10
11 - 17	Shot number	I7
18 - 21	Receiver number	I4
22 - 23	Blank	2X
24 - 25	Latitude degrees	I2
26 - 27	Latitude minutes	I2
28 - 32	Latitude seconds	F5.2
33 - 33	Latitude (N/S)	A1
34 - 36	Longitude degrees	I3
37 - 38	Longitude minutes	I2
39 - 43	Longitude seconds	F5.2
44 - 44	Longitude (E/W)	A1
45 - 52	Easting	I8
53 - 60	Northing	I8
61 - 65	Water depth	I5
66 - 68	Time (Julian day)	I3
69 - 70	Time (Hour)	I2
71 - 72	Time (Minute)	I2
73 - 74	Time (Second)	I2
75 - 75	Time (Tenth second)	I1
76 - 80	Blank	5X

Table 1: Format of Shot/Receiver Positioning File.

Once final location data had been derived for each receiver in the streamer at each shot, the information was then used to obtain the water depth at each receiver.

Determination of Water Depth at the Receiver Positions

All primary water depth data were measured to within 5m of the seismic lines, hence there was no direct way of exporting a water depth value into a receiver location, where in the general case the seismic cable was feathered and receivers were offset from the point of measurement. It was necessary to grid the water depth values over the entire survey and extract the water depth value at each receiver location from the grid. As water depth was quite flat in the Kununga survey area, a 100m x 100m grid was sufficient to represent the water depth surface while a 50m x 50m grid was used for Cognac. A coarse grid display was used to represent the Cognac water depths with the location of the seismic lines superimposed (Figure 5).

It was found that a relatively simple solution to this problem was achieved with the Petroseis[4] package by utilising a back-interpolation option off a grid. The following were the steps taken to obtain a water depth value at each receiver location:

1) A seismic master file was created for the UKOOA shot data.

2) The UKOOA file containing navigation and water depths for every line and shot was imported into the seismic master file.

3) Geographical positions of the shots were converted to AMG coordinates after choosing the relevant UTM Zone. Coordinate transformation parameters were saved.

4) Water depths were gridded on a map sheet covering the seismic area. The gridding parameters were saved.

5) A new seismic master file was created for the seismic streamer data.

6) The shot/receiver file was imported into the seismic streamer master file. The shot/receiver file was first formatted so that the streamer at each shot was given a unique name, which was made up by combining essential elements of original linename and shotnumber[5]. Each streamer was therefore represented in Petroseis as a "line", each with a "shot point" range of 1 to R+1, where R represents the number of receivers on the streamer. The number of lines created was equal to the number of shots in the survey.

7) The geographical positions of the shots and receivers were converted to AMG coordinates using the coordinate transformation parameters from step 3.

8) Each receiver for each streamer had a water depth value attached to it by interpolating the grid to the receiver position. As a check, the receiver water depths were used as input to produce a grid using the same gridding and display parameters from step 4. Figure 6 shows the results of regridding for the Cognac area.

[4] Product of Petrosys Pty. Ltd.
[5] For example, line 95MK-001 shot 1114 was represented as streamer 0010-1114 and line 95MK-034A shot 1356 as streamer 034A-1356.

COGNAC WATER DEPTH GRID

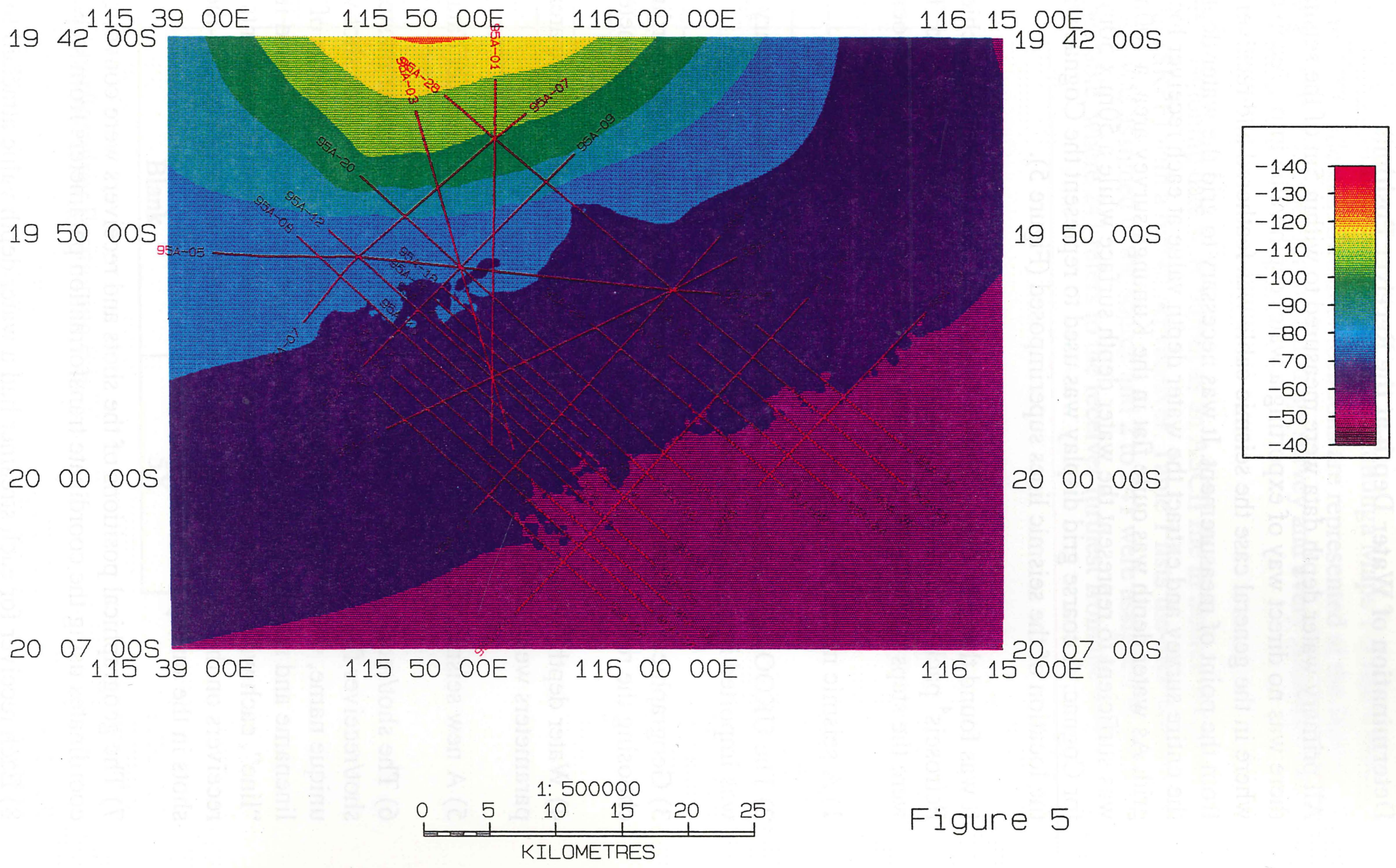

Figure 5

UNIVERSAL TRANSVERSE MERCATOR PROJECTION
AUSTRALIAN NATIONAL SPHEROID
CENTRAL MERIDIAN 117 00 00E

COLUMNS	NAME	FORMAT
1 - 10	Linename	A10
11 - 17	Shot number	I7
18 - 21	Receiver number	I4
22 - 25	Blank	4X
26 - 27	Latitude degrees	I2
28 - 29	Latitude minutes	I2
30 - 33	Latitude seconds	F4.1
34 - 34	Latitude (N/S)	A1
35 - 37	Longitude degrees	I3
38 - 39	Longitude minutes	I2
40 - 43	Longitude seconds	F4.1
44 - 44	Longitude (E/W)	A1
45 - 52	Easting	I8
53 - 60	Northing	I8
61 - 65	Water depth	I5
66 - 80	Blank	15X

Table 2: Format of exported shot/receiver positioning file.

LINE	SP	Receiver	Lat.	Long.	Easting	Northing	Water Depth
001	1001	0	"	"	"	"	"
	1001	1	"	"	"	"	"
		"	"	"	"	"	"
	1001	240	"	"	"	"	"
	1002	0	"	"	"	"	"
	1002	1	"	"	"	"	"
		"	"	"	"	"	"
	1002	240	"	"	"	"	"
	"	"	"	"	"	"	"
	"	"	"	"	"	"	"
	2465	0	"	"	"	"	"
	2465	1	"	"	"	"	"
		"	"	"	"	"	"
		240	"	"	"	"	"
002	1001	0	"	"	"	"	"
		1	"	"	"	"	"
		"	"	"	"	"	"
		240	"	"	"	"	"
"	"	"	"	"	"	"	"
"	"	"	"	"	"	"	"

Table 3: Structure of shot/receiver file (Blank = same as above; ditto = new value).

R9600303

COGNAC WATER DEPTH REGRID

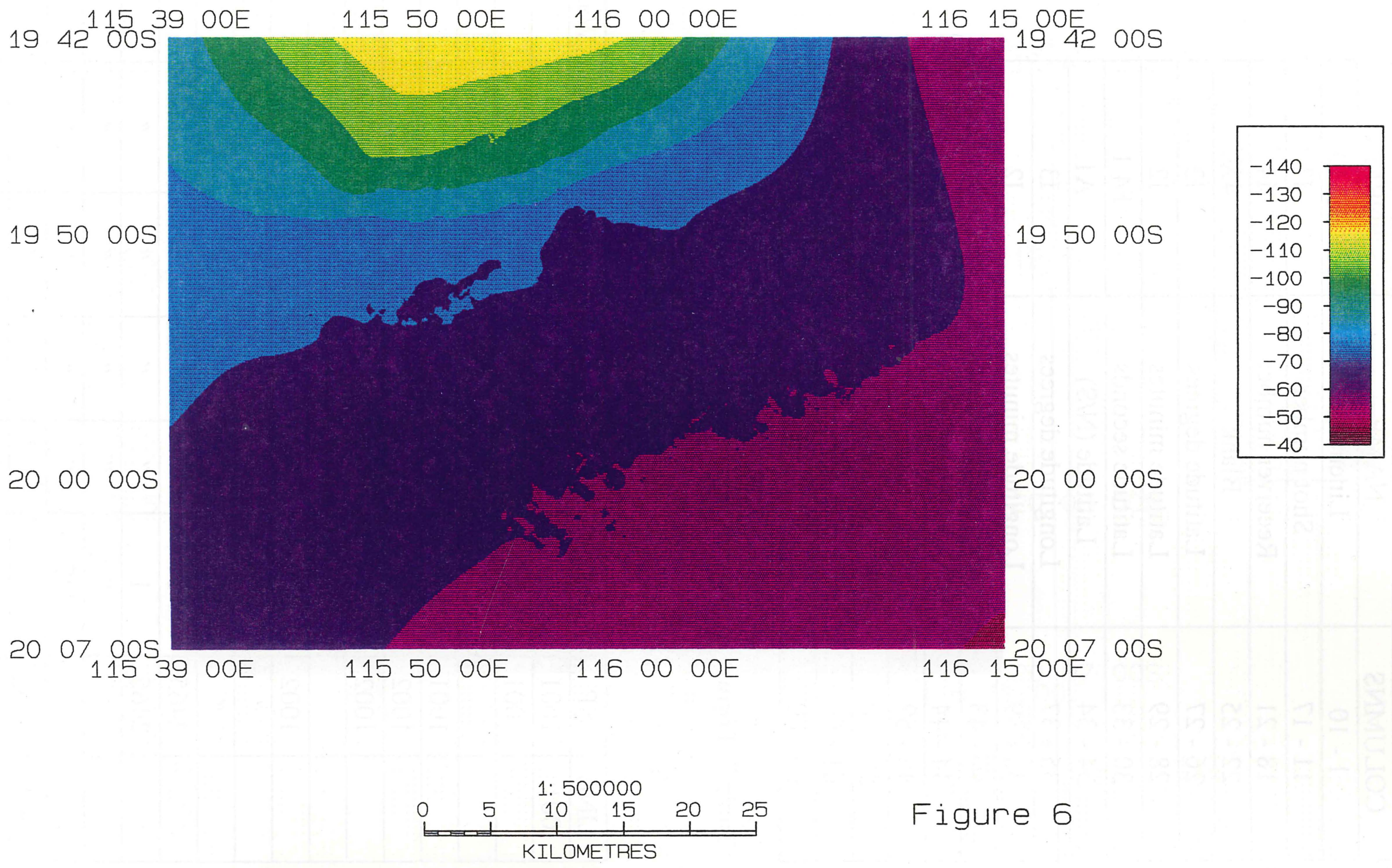

Figure 6

UNIVERSAL TRANSVERSE MERCATOR PROJECTION
AUSTRALIAN NATIONAL SPHEROID
CENTRAL MERIDIAN 117 00 00E

A comparison of the two grids at and around the seismic line locations show only slight differences, thus verifying the procedure.

The data were then exported from Petroseis to a file in UKOOA format. The file was reformatted using the Unix pattern-matching program, "awk", to the original linename and shot number while receiver numbers were resequenced from 0 to 240 (See Appendix 11 for program listings). Table 2 shows the reformatted shot/receiver positioning file. Navigation precision was reduced to 1 decimal place of an arc second as the Petroseis export option did not support a greater precision. The structure of the final shot/receiver file is shown in Table 3.

Conclusions and possible future work

Inspection of the streamer plots shows that on some occasions, the seismic cable is significantly feathered, such that receivers near the end of the cable are up to 500m offset from the seismic line. As well as not imaging near-surface reflectors below the programmed line, any seismic static corrections which are applied based on seismic line water depths may be erroneous and lead to incorrect statics being used in seismic processing.

Figure 7(a) is a display of water depths at each receiver for streamer 0730-1750 from the Kununga survey. There is a rapid change in water depth at receiver 125, where

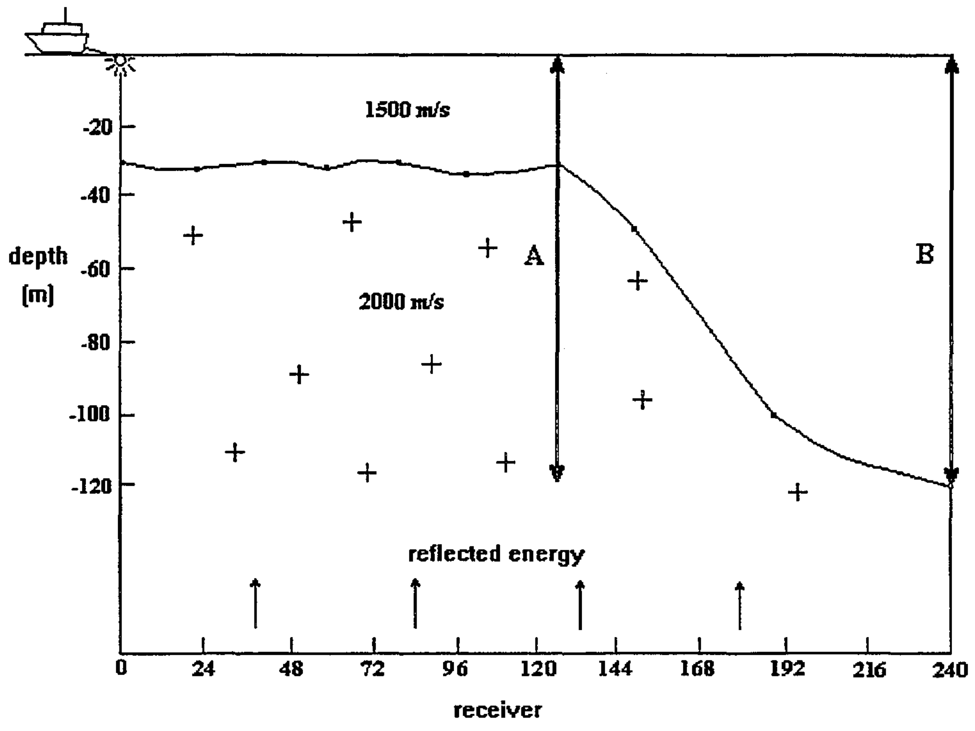

Figure 7(a): Diagram of the water depth profile for streamer 0730-1750.

R9600305

the depth drops from 31m to 119m at receiver 240. This is a worst-case example from the survey and will be used to illustrate the effect of variable time shifts caused by near-surface velocity inhomogeneities on the timing of deep relectors.

For simplicity, it is initially assumed that seismic energy is reflected from a flat layer at depth. At receiver 125 (path A), seismic energy passes through extra material compared to that at receiver 240 (path B), which has a larger water column. Assuming the material has a velocity of 2000m/s, this results in path 'B' being delayed by an additional two-way time of 15ms compared with path 'A'.

Consequently, the normal moveout curve for a shot recorded in such a situation would show a delay of 15ms at receiver 240 compared with a case where the topography was constant (Figure 7(b)). The delay diminishes as receiver 125 is approached, where the delay is zero. A similar effect will be seen in the seismic gathers at CDP locations between receiver 187 and receiver 62 (a quarter spread-length on either side of the dropoff in depth). A deformed hyperbolic curve as shown below will lead to erroneous velocity estimates in the velocity analysis. The expected final result under these circumstances is a degraded stack, as samples from adjacent traces will not line up in the seismic gathers after normal moveout correction. This is a significant problem in modern seismic interpretation where an attempt is made to use the wavelet characteristics to determine seismic stratigraphy. Correcting for time shifts on a trace-by-trace basis as early as possible in the seismic processing sequence will improve hyperbolic line-ups, allow better application of all subsequent processes such as velocity picking and Dip Moveout (DMO), and ultimately lead to better resolution in the final stack. An additional advantage of applying time shifts is that in areas of sea-bottom topography, the primary reflection energy will be adjusted correctly while multiple energy is only corrected once. This will cause the multiple to be degraded

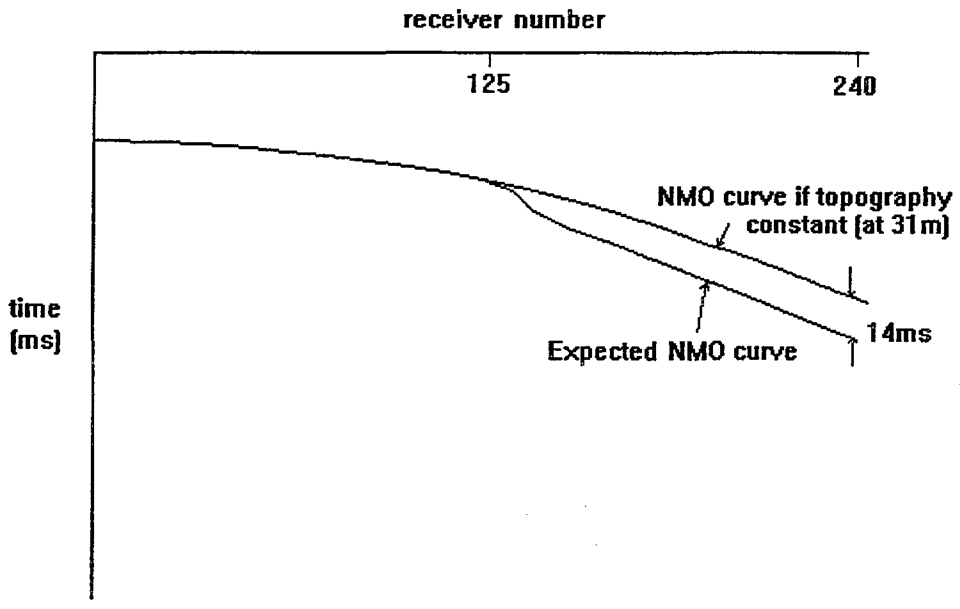

Figure 7(b): Schematic showing the response of a reflector on a shot record over the sea-bottom topography shown in Fig. 7(a).

when the section is stacked.

A process can be created in which streamer positioning and water depth information are integrated with the seismic data, to correct for time shifts (in the seismic data) caused by variable sea-bottom topography on deep reflectors. It is proposed by the author that a water depth grid, created either from water depth data acquired during the survey or from other sources of data, be combined with ray tracing of the seismic energy to each reflector to determine the two points on the sea bottom where the seismic wave enters and emerges for each trace/reflector combination. After extracting the water depth at these points from the grid and choosing a suitable datum level, a total timeshift (for each reflector) may be computed by adding the delay for the downgoing and upgoing wave for each trace. The ray tracing, if necessary, could be done as early as the picking of first stage velocities if the velocities are thought to be good enough to allow accurate ray path modelling. This process could be conducted separately from the seismic processing. All that would be needed is a good velocity field for ray tracing.

A qualitative description of the seismic processing procedure could be as follows. A streamer positioning file containing water depths at each shot and receiver is read and stored in the seismic database. A seismic line geometry is created from the shot-receiver positioning information and seismic processing is carried out up to and including first-pass velocity picking. At this stage, the geometry, the shot-receiver water depths and grid, the seismic velocity field and a brute stack may be used as input into a combined GIS/interpretation workstation, where the reflection horizons are manually picked. Ray tracing may then be automatically performed to each interpreted horizon for each shot-receiver pair, and two water depth values - one each at the point where the rays enter and emerge from the water bottom - can be stored in the seismic database (or file) along with the time to its associated reflector. Points on the water depth grid may be picked off a common location system such as a Geographic Information System (GIS), if necessary. These depth values may then be introduced back into the seismic processing stream and used to correct for the effect of water bottom irregularities and known near-surface velocity inhomogeneities on the timing of reflections for each trace in the gathered data. A near-surface velocity model and a datum level would need to be specified at this stage (Appendix 9 describes the theory of marine seismic static corrections). The corrected gathers would then be used in all subsequent processing up to final stack.

In areas where the sea-bottom topography changes significantly within a range of a streamer length, processing of the seismic streamer location and associating a water depth with each receiver is an important pre-processing step which needs to be included when considering a comprehensive solution to the effect of non-uniform time shifts caused by variable near-surface velocity distributions on marine seismic data.

Bibliography

Data Processing Manual for Marine Navigation, Gravity, Magnetics and Water Depth Data, AGSO, Marine Petroleum and Sedimentary Resources, 1994

References

AGSO, 1995a, Parameter Report: Kununga 2D Marine Seismic Survey Permit Block WA-235P Timor Sea, Australian Geological Survey Organisation internal report (unpub.), Marine Petroleum and Sedimentary Resources.

AGSO 1995b, Parameter Report: Cognac 2D Marine Seismic Survey Permit Block WA-238P North Western Australia, Australian Geological Survey Organisation internal report (unpub.), Marine Petroleum and Sedimentary Resources.

UKOOA, 1990, P1/1990 Post plot data exchange tape 1990 format, UK Offshore Operators Association (Surveying and Positioning Committee), internal document.

Appendix 1 - Marine Navigation Data Processing

The primary navigation systems used on board the Rig Seismic during the survey were the Racal Multifix DGPS system (Racal#1) and Racal Multifix II DGPS system (Racal#2). Navigation-related data and some geophysical data were written to a 100-channel binary format file. Data were passed from instruments on board the ship to the DAS computer and written to tape at 1 second frequency. In the processing phase, this file was copied to disk and resampled to 10 second frequency. All subsequent files were also in binary format. Upon completion, the final processed data were resampled to 60 second frequency and placed in the Non-Seismic Processing binary file database.

The following describes the processing steps necessary to refine the navigation data. Data were regularly checked to ensure that no unusual effects were created and passed on.

RESAM - Binary data were resampled to 10 second frequency.

SWAP - All channels which were relevant to the navigation processing were extracted from the DAS file. These were Racal#1 and #2 latitude, longitude, UTM time and time-of-fix; best estimate latitude and longitude.

XTIME - Records containing bad times or survey numbers were omitted.

FIXTM - Time-order errors were corrected by padding all missing time records and omitting duplicated time records. This process was repeated a number of times to produce a continuous time-indexed data set. The output was checked for time-order errors by running program MLIST.

IGPS - Navigation data were corrected by accounting for a time lag within the Racal instrument. The program corrected the navigation data by first subtracting the time lag between the time of satellite fix and the time of output record. The data were then interpolated to the 10 second mark. (Racal#1)

IGPS - Same as above for Racal#2.

BADLL - Bad latitude or longitude values were replaced by missing values.(Racal#1)

BADLL - Same as above for Racal#2.

FDATA - The data were despiked by applying a median filter. A 5-10 sample window and a 1.5e-5 radian threshold were applied.

FTAPE - Latitude and longitude were interpolated over gaps of up to 3 minutes.

PCEDAT - The data were interactively edited by using a PC-based program. This process was used to manually patch data between the two primary navigation systems to give the best data set. The navigation could be interpolated over more sizeable time gaps if it could be inferred after examining the track map and heading data that the positions during this period lay in a straight line and speed was constant. Navigation would not be interpolated where the ship travelled around a bend and would not be interpolated at all when the time gap was greater than about 20 minutes.

SMTH - The values were smoothed by applying a sinc function filter. Parameters used were 3 zero-crossings and a 60-180 second period.

SWAP - Final navigation (at the ship's antenna) were swapped to channels 3 and 4, which were the channel allocations for final latitude and longitude respectively.

RESAM - Binary data were resampled to 60 seconds and placed in the NSP database.

PLTCH - Latitude, longitude, Δ-latitude and Δ-longitude data were plotted in profile form on a zeta plotter. The plot was checked for any unusual effects.

PSMAP - A map of the ship's track was plotted over the whole survey.

The following processing steps were carried out to obtain a shot-based ASCII navigation file of the seismic data. Processing of the shot-times file was as follows.

CHECKST - The shot-times file was checked for shotpoint continuity. Missing or duplicated shotpoints were identified for each line.

VMS EDIT - The shot-times file was edited to first and last good chargeable shots, missing shots were interpolated and repeated shots omitted. Editing information was primarily obtained from the seismic acquisition line logs. Program STFIL may also have been used to interpolate over small gaps in the shot-times file.

DELAY - The shot-times were delayed so that they represented the time at which each shot was at the seismic stack point (normally at the source or the midpoint) rather than at the antenna. This enabled the shot number and the position of the seismic stack point to be matched at a common time. A file can be included which contains all source to near-trace offset changes (See Non-Seismic Data Processing Manual).

SPNUM - Seismic lines which were shot in parts may have had lineparts merged, lines renamed and shotpoints resequenced, if required.

A UKOOA format file was produced and added to the shotpoint location database.

UKOUT - Binary navigation were combined with the processed shot-times file to produce a modified UKOOA file which contained the navigation for each shot.
UKCHECK - The UKOOA file was checked for shotpoint continuity as well as time and distance differences between consecutive shots. These differences were reported after exceeding user-specified threshold limits.
AGDUKO - (Opt) Data were converted from WGS84 to AGD84 datum.
XYDATA - (Opt) Corresponding AMG coordinates were calculated for the relevant UTM zone and positioning datum (See Appendix 7 for format).

Appendix 2 - Water Depth Processing

Water depth data were written digitally to tape at 1 second sample rate for both the 3.5 KHz and 12 KHz echo sounders. In bad data areas or where the instrument lost tracking of the water bottom, the digital data were supplemented by digitising the analogue records. The sweep rate used in the instrument was 4 seconds (ie every 4th sample or so had good water depth data), therefore the first few steps of the following processing sequence were included to maximise the amount of good data retained for further processing. Files were subsequently processed at 10 second frequency.

SWAP - Channels containing 12 KHz and 3.5 KHz water depths were extracted from the DAS file at 1 second sample rate.

FTAPE - Water depth data containing error flags (written by the instrument) were omitted.

FTAPE - Missing water depths were interpolated over all gaps of less than 2 minutes. Gaps were usually not more than 4 second duration.

RESAM - Data were resampled to 10 second frequency.

FIXTM - Time-order errors were corrected by padding missing times and omitting duplicated times. It was necessary to iterate this procedure a number of times.

MATHS - Water depths were converted to negative values and then corrected to account for the depth of the transducer below the water line.

PCEDAT - Water depths were interactively edited by using a PC-based editor. The type of editing to be performed was decided after inspection of the analogue records.

SALVG - The water bottom was manually digitised over areas where the digital data were poor. This technique used the digitised values to control the digital data. If the difference between the digital data and the values of interpolated points between digitised values at the corresponding times were greater than a given threshold, the data were modified otherwise they were retained.

PCEDAT and SALVG may both be employed in the water depth processing. To obtain the best results, the data were first checked using PCEDAT. The method employed was dependent on the amount of 'bad' data in the survey. As the data quality was reasonably good and water depths were relatively flat, it was easiest to process with PCEDAT only, otherwise both techniques would have been used.

At this stage, a UKOOA file was produced.

UKOUT - The 'delayed' shot-times file was combined with the navigation/water depth file to produce a modified UKOOA file containing water depth.

Appendix 3 - Streamer Position Processing

Compass, source and tailbuoy positions were the data used to process the seismic streamer location at every shot. During acquisition, compass data were written to a 700-word binary format (QC) file, while tailbuoy data were written to the DAS file. The following procedure describes the method developed to first compute the streamer position and then individual receiver positions.

1. Compass Data

QCDATA - Compass data were extracted from the QC file.

ATSEC2 - Data were interpolated to the 10 second mark.

FIXTM - Time order errors were corrected. It was necessary to iterate this process once or twice until there were no remaining time gaps.

FTAPE - The data were interpolated over gaps of up to 3 minutes.

2. Tailbuoy Data

SWAP - The tailbuoy latitude, longitude and time information were extracted from the DAS file.

XTIME - Records containing bad survey numbers or bad times were omitted.

FIXTM - Time order errors were corrected.

FDATA - The data were despiked. A window of 5-10 samples and a threshold of 1.5e-5 radians were applied.

FTAPE - Data were interpolated over time gaps of up to 10 minutes.

PCEDAT - Interactive editing was applied to remove or modify obviously bad values.

SMTH - The tailbuoy positions were smoothed by applying a sinc function filter.

3. Compass, Heading and Course Data

MERGE - Compass data from **1.** were merged with course and heading information (from DAS file) into one file. Values for all data were in degrees.

MATHS - Angle values were converted from modulus 360 form to a continuous function.

FDATA - Compass, heading and course data were despiked.

FTAPE - Data were interpolated over time gaps of up to 3 minutes.

PCEDAT - All compass data were displayed on a PC-based screen editor and obvious errors in the data were interactively edited. Remaining time gaps in the compass readings were noted and an interpolation carried out on a particular compass if it could be seen that other compass values were linearly varying at the times corresponding to where they were at identical locations to the compass being edited.

MATHS - The compass values were converted back to modulus 360 values.

4. Compass Location Data

MERGE - Final navigation were merged into channels 3 and 4. Tailbuoy data from **2.** were also merged into the file.

COMPASS - Each compass location was converted to dx and dy coordinates, with the origin defined at the back of the boat. The algorithm assumed that the streamer was rigid between compass locations. The streamer was defined by the equation given by extrapolating from a compass location with a gradient given by the compass angle at that point back toward the ship for a distance equal to the inter-compass length. Heading and course data were used in the calculations.

FDATA - The dx, dy values were despiked. The despiking parameters chosen were a window of 6 samples and a 1m threshold.

SMTH - The compass dx, dy values were smoothed by applying a sinc function filter. The parameters used were 3 zero crossings with a 3 minute period.

CABLE - The latitude and longitude positions for each of the compasses were calculated. This program accepted course, heading and compass dx, dy data. A dx cutoff value of 800m was applied.

5. Compass Location UKOOA file

UKOUT - The ship, compass and tailbuoy position data were combined with the 'pre-delay' shot-times file to make a modified UKOOA file containing the positions of the ship, compasses and tailbuoy.

ANT2SRC - The shot location (at the source) was inserted from the 'delayed' shot UKOOA file into the compass location file.

ADJUST - Compass positions were adjusted by rotating the line between source and last compass location, by pivoting around the source, to the line between source and tailbuoy location.

STRUKO - (Optional) Source, compass and tailbuoy locations were converted from WGS84 to AGD84 datum.

FILT_REC - Records with any inter-compass distances differing by more than 30 metres from the expected distance were omitted.

EXTRAP - Shot locations for all missing records from SOL and up to EOL were extrapolated and inserted back into the file.

UKCHECK - The UKOOA file was checked for shotpoint continuity as well as time and distance differences between consecutive shots. These differences were reported whenever they exceeded the threshold limits as specified by the user. The start and end shotpoint numbers were listed for each seismic line.

At this stage the compass location file was fully processed and was ready to be used as input to generate the receiver locations.

6. Shot/Receiver positioning file

GENRATE - This program was used to generate a file containing geographical and rectangular coordinates for (i) each shot and (ii) all receiver locations on the cable for each shot in the survey. The coordinates were computed by evenly interpolating the receivers between compass locations. The output was produced in either normal modified UKOOA format or Petroseis format, where each line-shot combination was represented as a unique line name. In the Petroseis format, receiver labels were numbered from 1 to R+1, where 1 represented the source location, 2 represented the receiver closest to the ship and R+1 the receiver furthest from the ship (R was the number of receivers on the cable). The program interpolated shot and receiver locations across missing shotpoint ranges in the compass location file. **Note:** This receiver numbering convention was chosen as Petroseis did not accept an input shotpoint number of 0.

CHKREC - The shot/receiver file created by program GENRATE was checked. Start and stop shots for each original linename were printed, discrepancies in both the inter-receiver distances and the distance between consecutive shot positions were reported. Shot and receiver numbers were checked for continuity and also that there were R receivers associated with every shot.

Appendix 4 - Streamer Positioning Errors

Discussion of AGSO Streamer Processing

The seismic streamer processing sequence as described in Appendix 3 minimises error to some degree in the receiver positions. It is necessary however, to be aware of where errors are created in the derivation of these values. Understanding of the limitations of current techniques and technology may lead to software or hardware improvements which address the requirement to more accurately locate the streamer in future.

The positions of compasses on the seismic cable are determined by using compass angles and known inter-compass cable lengths between known source and tailbuoy positions. The equation defining the streamer location and shape is determined by extrapolating from the source position to a (compass) point with a gradient given by the compass angle at that point for a distance equal to the inter-compass length. This is repeated for each pair of compasses until the streamer is defined. Compass positions are then rotated around the source position to the tailbuoy (See Fig. 4). The positions of the receivers are computed by evenly interpolating between compasses. Receiver positions before the first and after the last compass positions, are determined by extrapolating to the first/last receiver on the cable at the nominal group interval.

Errors of between 30 and 100m are expected in the tailbuoy location. It is also expected that streamer positioning errors will be generated in the extrapolation process. The determination of streamer location is largely dependent upon compass data (with no absolute positioning check available), therefore it is likely that one or more small compass errors (or local compass anomalies) will lead to compound position errors at distances further down the cable. The assumption that the streamer is rigid between compasses will also lead to errors when the streamer is significantly curved (Figure 8), as may be expected in areas of rough seas or variable currents. This will lead to lateral displacements between the real and calculated receiver positions. It will also cause in-line displacements which increase the further down the cable that it is imaged. Although the algorithm may be expected to minimise the location error over the whole streamer, there are still some receivers which can have relatively large errors in their positioning.

If it is a requirement in future that streamer position accuracy be improved, it will be necessary to more accurately determine tailbuoy location and better determine and correct the errors which lead to incorrect receiver positions on the streamer.

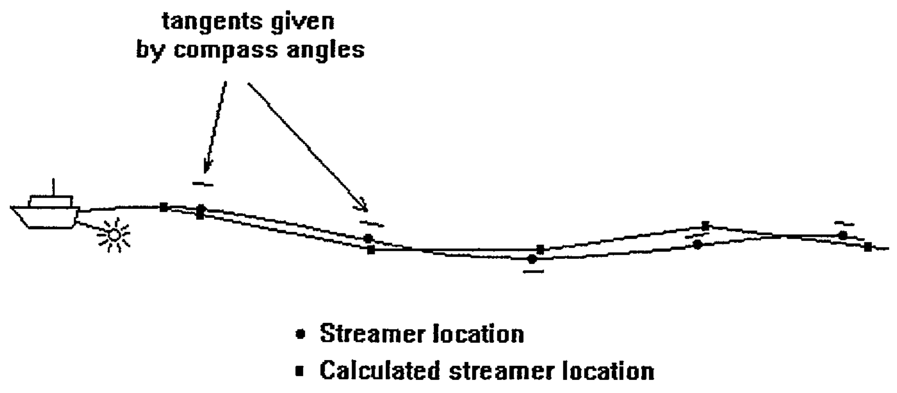
tangents given by compass angles
• Streamer location
▪ Calculated streamer location

Figure 8: Crude diagram showing a comparison between real streamer location and calculated streamer location based on the straight length of cable between compasses and the direction given at each compass. A curved streamer best illustrates the typical errors.

Review and Discussion of Industry Techniques for Streamer Positioning

Industry streamer positioning has been based upon curve fitting techniques between a known source and tailbuoy position with compass data providing the gradient of the curve at known points along it. Further research into streamer positioning has shown that errors can exist in the source and streamer head locations and that they are often not very accurately known (Manin et. al., 1988). In their paper they show that the conventional assumption for determining the source position, i.e. that it trails a fixed distance behind the ship and is where the ship has just been, is inadequate under some weather conditions where the heading of the ship can differ from the heading of the streamer by up to 10 degrees. This method can lead to a 10m lateral error in the source position. They directly determine the source position by placing a Simrad range/angle acoustic device on the source and vessel (The source positioning error was found to be 2m after checking it with a laser rangefinder and sextant). A similar technique is used to locate the streamer head.

A number of differences exist between Manin's streamer processing and ours, which are briefly mentioned below.

1) The validity of a compass value was assessed by taking the difference between it and another compass value and matching it with the change in course of the ship at the corresponding points of the ship's track. Both compasses were given a validation mark if the match fell within an acceptable value range. This procedure was repeated for each pair of compasses, and their values were accepted once each had a minimum number of validation marks. Non-validated values were replaced by an estimate based on the readings of the closest validated compasses and the model of the streamer.
2) The cable shape between compass positions was fitted by an arc. This reduced the effect where the computed receiver locations fell behind their real locations.

3) Bias was isolated in one compass measurement which led to creating a kink in the cable shape at either side of that compass. A 'relative calibration' was able to be applied after different sailing patterns and statistical techniques were applied.
4) The streamer profile was processed to correct for compass values to true north.
5) Absolute positioning of the tailbuoy was carried out by using a Syledis SB5 beacon on both the ship and tailbuoy. A shore station was interrogated to give distance from the station to the ship and a distance from station to the tailbuoy. The distance to within 3-5m could be deduced for the ship to tailbuoy.

After 1988, it was felt in the seismic industry that compass data alone were not accurate enough to determine the streamer shape (Court, 1991). It had been noted (Krail and Brysk, 1989) and (Court, 1991) that streamer shape was highly sensitive to compass errors and that errors in positioning could propagate down the cable. Some additional factors which led to this assessment were that the compasses could not be calibrated in their operational setting, the magnetic variation over a survey area may not remain constant and that curve fitting between compass locations was arbitrary in the absence of extra information.

The use of multipath acoustic ranging systems to determine the source and streamer head positions were described by Court (1990). In his paper, he refers to a similar system proposed by Hall et. al. (1988) for use in multi-streamer operations to determine the complete positioning for a survey, by acoustic ranging between any number of underwater locations. In this system, the position of source arrays, streamer heads and streamer shapes could be determined by the placement of transducers at various strategic locations on the ship, source and streamers.

Major aspects of Court's methodology are briefly described. The transducers are tuneable and have operational frequencies of between 50 and 100 kHz, well outside the frequency band of seismic data. Multipath ranges are acquired by configuring one of the transducers as a transmitter and the others as receivers. All receivers detect the transmitted pulse and a traveltime is returned at each. By sequentially reconfiguring each of the devices, in turn, as a transmitter, a net of interleaving measurements can be obtained, giving the propagation time between them. This procedure is repeated at every shotpoint. Hull-mounted transducers which are a fixed length apart provide the means to measure the velocity of sound in seawater and monitor any changes in velocity. The calculated speeds are used to convert acoustic traveltimes between pairs of transducers into ranges. Using an iterative technique known as the variation of coordinates, each set of distance measurements is used to determine the locations of these transducers (Court, 1989). A correction for the Doppler shift is first applied whenever a velocity is calculated between two transducers which lie in a line along the motion of the ship through the water. The effects of ship motion are eliminated by applying a correction involving the traveltimes in the aft-fore and fore-aft directions, the transducer separation and the velocity of the ship (p560). He is careful to note the effect of bubbles created by the seismic source on the transducer signals which have to pass through them (the speed of sound can be reduced by up to a factor of 2 when passing through aerated water). This was only a problem for a short time at the time of the seismic shot - transducers located on the source arrays being immediately moved ahead of the bubbles as the ship progressed.

Possible Future Improvements in AGSO Streamer Positioning

Based on the current industry standards for streamer processing, it should be considered how use might be made of industry techniques for application to AGSO's streamer positioning in future. The points raised after reviewing Manin's paper could all be introduced (with some effort) to our acquisition and processing. The major improvement being the deployment of a tailbuoy positioning system which is accurate to 5m (Canter et. al., 1989). Acoustic devices on the ship's hull, source arrays and streamer head could also be deployed straitforwardly to determine the positions of the source and streamer head in a similar way to Manin and others.

The application of a ranging network as suggested by Hall and Court would be impossible as only one streamer is deployed on the Rig Seismic. Assuming an effective range of the acoustic pulse of at least one streamer length (about 3km), it may be possible to place acoustic devices at two known positions (e.g. at the ship and tailbuoy) and at various known points along the streamer. By transmitting a signal from the ship's transducer which is received by all the others and then sending a pulse from each transducer in turn while the others receive, a number of ranges may be determined to known points along the cable. It is hard to envisage how positioning could be effectively performed on points on the streamer however, without some sort of short-range, omnidirectional ranging control.

Finally, the shape of the streamer between the known positions of the acoustic devices may be estimated by combining compass information with an indication of the degree of curvature between transducer points. An indicator which could be used is the ratio of the range between any two acoustic devices and their cable separation. For the ease of mathematical curve fitting, it would be advantageous if, during acquisition, the acoustic devices were physically located close to the compasses on the streamer. The shape of the streamer, as has been noted in the literature, is the most difficult aspect to correctly image due to subtleties in its physical behaviour in sea conditions.

References

Canter, P., Nordmoen, B., Asheim, S., Vigen, E., 1989, Evolution of Positioning in Marine 3D Seismic, Presented at 59th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, p. 606-609.

Court, I., 1989, Application of acoustics to streamer/source positioning, Presented at 59th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, p. 610-612.

Court, I., 1991, Application of acoustics to source-array and streamer tow-point positioning, Geophysics, V.56, p. 558-564.

Hall, M., Norton, J., Court, I., 1988, Point location determination at or close to the surface of the sea, U.S. Patent 07/245,006.

Krail, P., Brysk, H., 1989, The shape of a marine streamer in a cross-current, Geophysics, V.54, p. 302-308.

Manin, M., Boucquaert, F., Regnaudin J., Regnault, A. & Thevenot, A., 1988, Recent developments in source and streamer positioning, First Break, Vol.6, No.6, p.183-188.

Appendix 5 - Acquisition Parameters

The following lists the acquisition parameters for the Kununga (Survey 160) and Cognac (Survey 161) seismic surveys.

Parameter	Distance (m)
Antenna to stern	45.9
Stern to centre of source	36.25
Source to channel 1	75
Antenna to channel 1	157.15
Tailbuoy to channel 240	110
Channel 9 (Compass 1) to antenna	257
Channel 73 (Compass 2) to antenna	1057
Channel 137 (Compass 3) to antenna	1857
Channel 185 (Compass 4) to antenna	2457
Channel 233 (Compass 5) to antenna	3057
Channel 240 (End of cable) to antenna	3157
Tail buoy to antenna	3267
Group interval	12.5
Shot interval (Kununga)	25
Shot interval (Cognac)	18.75
Streamer depth (Kununga)	6
Streamer depth (Cognac)	7
Stretch length	50
Number of compasses	5
Number of channels	240
Front stretches	1
Rear stretches	1

Appendix 6 - NSP Binary Database channel allocations

At the completion of processing, binary files were stored in the database with the following channel allocations:

Channel 3 - Latitude
Channel 4 - Longitude
Channel 5 - Water depth
Channel 6 - Gravity
Channel 7 - Magnetics
Channel 8 - Magnetic IGRF anomaly
Channel 9 - Gravity free-air anomaly

Appendix 7 - AGSO Modified UKOOA Formats

The AGSO UKOOA high precision format is defined as:

Columns	Information	Format
1 - 16	Line name (left justified)	A16
17 - 23	Shot-point number	I7
24 - 33	Latitude (Deg Min Sec, N/S)	I2,I2,F5.2,A1
34 - 44	Longitude (Deg Min Sec, N/S)	I3,I2,F5.2,A1
45 - 52	Gravity ($\mu m/s^2$)	I8
61 - 65	Depth (m)	I5
66 - 75	Julian day (ddd) and UTC (hhmmsst)	I3,I2,I2,I2,I1
76 - 80	Magnetic Field (nTesla)	I5

This format has been in use from Survey 127 (Enderby Terrace, May 1994) onwards.

The AGSO UKOOA normal precision format used before Survey 127, is:

Columns	Information	Format
1 - 16	Line name (left justified)	A16
17 - 23	Shot-point number	I7
26 - 34	Latitude (Deg Min Sec, N/S)	I2,I2,F5.2,A1
35 - 44	Longitude (Deg Min Sec, N/S)	I3,I2,F5.2,A1
45 - 52	Gravity ($\mu m/s^2$)	I8
61 - 65	Depth (m)	I5
66 - 74	Julian day (ddd) and UTC (hhmmss)	I3,I2,I2,I2
75 - 80	Magnetic Field (nTesla)	I6

Note: All missing data are filled with 9's for the full field column width.
Easting and northing information overwrite the gravity data and are placed in columns 45-52 and 53-60 when required. In this case the format is:

Columns	Information	Format
1 - 16	Line name (left justified)	A16
17 - 23	Shot-point number	I7
24 - 33	Latitude (Deg Min Sec, N/S)	I2,I2,F5.2,A1
34 - 44	Longitude (Deg Min Sec, N/S)	I3,I2,F5.2,A1
45 - 52	Easting	I8
53 - 60	Northing	I8
61 - 65	Depth (m)	I5
66 - 75	Julian day (ddd) and UTC (hhmmsst)	I3,I2,I2,I2,I1
76 - 80	Magnetic Field (nTesla)	I5

Appendix 8 - Compass Position File Format

The ASCII file containing source, five compasses and tailbuoy locations were written to a 206-character length file. The length may be extended if more than 5 compasses were used.

Column	Parameter		Format
01 - 16	Line Name		a16
17 - 23	Shot Number		i7
24 - 33	Latitude	source	i2,i2,f5.2,a1
34 - 44	Longitude	source	i3,i2,f5.2,a1
66 - 75	Time that antenna was at source		i3,3i2,i1
81 - 90	Latitude	compass 1	i2,i2,f5.2,a1
91 - 101	Longitude	compass 1	i3,i2,f5.2,a1
102 - 111	Latitude	compass 2	i2,i2,f5.2,a1
112 - 122	Longitude	compass 2	i3,i2,f5.2,a1
123 - 132	Latitude	compass 3	i2,i2,f5.2,a1
133 - 143	Longitude	compass 3	i3,i2,f5.2,a1
144 - 153	Latitude	compass 4	i2,i2,f5.2,a1
154 - 164	Longitude	compass 4	i3,i2,f5.2,a1
165 - 174	Latitude	compass 5	i2,i2,f5.2,a1
175 - 185	Longitude	compass 5	i3,i2,f5.2,a1
186 - 195	Latitude	tailbuoy	i2,i2,f5.2,a1
196 - 206	Longitude	tailbuoy	i3,i2,f5.2,a1

Appendix 9 - Marine Seismic Statics

Marine seismic statics can be computed as follows. A datum level should first be chosen which is a small distance below the low-velocity near-surface layers and into bedrock. The aim is to adjust each trace (in time) as though each wave has passed through a flat, uniform water bottom. This is achieved mathematically by replacing the material overlying the datum level with water and computing the extra time delay that would be expected for each trace. Assuming rays enter the subsurface at near to normal incidence and there are n near-surface sediment types, each with velocity V_n and thickness a_n; and thickness b_n for emerging waves (Figure 9), the following can be applied to correct all seismic data below the datum level.

Data samples may first be zeroed from sea level (T=0) to the datum level at time

$$T = (a_0/V_0) + (a_1/V_1) + (a_2/V_2) + \ldots\ldots + (a_n/V_n) = \sum_{i=1}^{n+1} a_{i-1} / V_{i-1}$$

A delay equal to the sum of the differences in travel time between water and the sediment for each layer is then applied to each trace.

$$\text{one-way delay} = (a_1/V_0 - a_1/V_1) + (a_2/V_0 - a_2/V_2) + (a_3/V_0 - a_3/V_3) + \ldots\ldots + (a_n/V_0 - a_n/V_n)$$

$$= \frac{a_1(V_1 - V_0)}{V_1V_0} + \frac{a_2(V_2 - V_0)}{V_2V_0} + \frac{a_3(V_3 - V_0)}{V_3V_0} + \ldots\ldots + \frac{a_n(V_n - V_0)}{V_nV_0}$$

$$\text{total delay} = \sum_{i=1}^{n} \frac{(a_i + b_i)(V_i - V_0)}{V_iV_0}$$

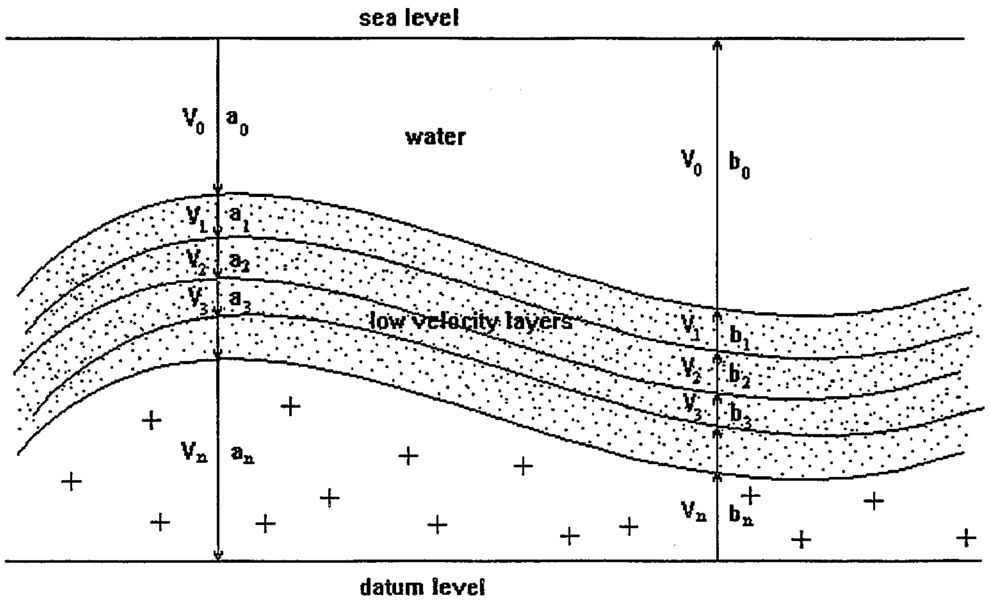

Figure 9: Sediment thickness and velocity model for near-surface layers. The seismic delay can be computed by replacing all material above the datum level with water.

The above formula will give the time shift to apply at a particular reflector given normally incident rays. This may be improved upon during raypath modelling where the slope of the raypath through the near-surface layers can be taken into account to compute the total distance travelled through each layer.

Blackburn (1981) notes that for raypath modelling over irregular or steeply-dipping water-bottom situations, replacement static corrections cannot be considered time-invariant, and if correct stacking is to be performed, then appropriate static corrections should be made for various layers. Dent (1983) similarly states that in areas of sea-bottom topography, the delay dynamically varies with time down the trace. This occurs because each ray must pass through a different part of the water bottom and near-surface to arrive at different reflectors given that the source/receiver pair is the same. His results were based on ray tracing to various reflectors and then applying a dynamic time shift to each trace. The time shift to each reflector was computed and applied and the intervening portions of the trace between reflectors were interpolated.

References
Blackburn, G., 1981, Seismic Static Corrections in Irregular or Steeply Dipping Water-Bottom Environments, Bull. Aust. Soc. Explor. Geophys., V.12, p. 93-100.

Dent, B., 1983, Compensation of marine seismic data for the effects of highly variable water depth using ray-trace modeling - A case history, Geophysics, V.48, p 910-933.

Appendix 10 - FORTRAN Programs

Some of the FORTRAN computer routines used in the streamer position processing are shown below.

```
      subroutine COMPASS_work(lun_in, lun_out, day1, time1, day2, time2,
     +    number_of_front_stretches, number_of_rear_stretches,
     +    seismic_channels, number_of_compasses, compass_chan1,
     +    compass_channels, source_to_chan1,
     +    dist_antenna_to_rear_of_ship,
     +    energy_source_offset, tail_rope_length, channel_interval,
     +    Nprint, lat_chan_bird_1,
     +    heading_ch, course_ch)
c***
c***  COMPASS
c***
c***  Author: N. Johnston        Mods: P. Petkovic
c***  Version June 1995
c**
c**   COMPASS computes the shape of the seismic streamer
c**


      IMPLICIT NONE

      integer max_sensors
      parameter (max_sensors=10)

      integer cli$get_value, cl_status
      integer len, lun_in, lun_out, ios, ior, iow, Nrec
      integer numfields, Nrecords, i, isec, Nshot, Nline
      integer number_of_front_stretches, number_of_rear_stretches
      integer seismic_channels, num_good_points
      integer number_of_compasses, compass_chan1 !first channel of compass data
      integer compass_channels(max_sensors)
      integer lat_chan_bird_1, heading_ch, course_ch
      integer day1, time1, day2, time2, tstart, tstop, Nprint
      integer feather_chan

      character file_in*40, file_out*40, io_errmsg*40

      real store(64), dist, head, dist0, extra
      real comp_headings(max_sensors)  !heading of each bird
      real delta_x(32), delta_y(32)   !(x,y) values relative to rear of ship
      real compass_x(max_sensors)     !easting of compass
      real compass_y(max_sensors)     !northing of compass
      real feather(max_sensors)       !feather angle
      real bird_headings(max_sensors)
      real dist_last_channel_to_tail_buoy
      real dist_antenna_to_rear_of_ship
      real energy_source_offset               !from stern to source centre
      real front_stretch, tail_stretch, cable_length
      real das_heading_local, das_course_local
      real bird_headings_local(max_sensors)
      real tail_buoy_feather_angle
      real tail_buoy_dist_from_ship
      real tail_rope_length
      real channel_interval
      real course, rear_x, rear_y, tail_dx, tail_dy
      real offsets(300)      !source to channel distances
      real source_to_chan1

      real small, rad
      parameter(rad=57.29577951, small=0.1)
c=================================================================

c Compute processing time interval
      call cvt_dhmsecyr(float(day1)/1000.0,float(time1)/1.0e6,tstart)
      call cvt_dhmsecyr(float(day2)/1000.0,float(time2)/1.0e6,tstop)

      front_stretch = 50.0 * number_of_front_stretches
      tail_stretch =  50.0 * number_of_rear_stretches
      dist_last_channel_to_tail_buoy = tail_stretch + tail_rope_length

      write(*,'(" course channel: ",i3)') course_ch
      write(*,'(" heading channel: ",i3)') heading_ch
```

```
c Compute offsets to each channel
    do i=1,seismic_channels
      offsets(i) = source_to_chan1 + (i-1) * channel_interval
    end do

c Compute the length of the cable from the stern to tail buoy
    cable_length = seismic_channels * channel_interval !active length
   +  + offsets(1) - channel_interval/2          !source to 1st active
   +  + tail_stretch + tail_rope_length          !last active to buoy
   +  + energy_source_offset                     !stern to source

    write(*,'(" Stern to Tail Buoy: ",f8.1," m")')cable_length
    write(*,'(" Last channel to tail buoy: ",f8.1," m")')
   +  dist_last_channel_to_tail_buoy

    do while(.true.)
     Nrecords = Nrecords + 1
     call io_rbuff(lun_in, store,0, ios)
     if(ios .NE. 0) goto 990
     call cvt_dhmsecyr(store(1), store(2), isec)
     if(isec.lt.tstart) goto 800

     Nshot = 99999
     if(store(21).lt.6000.0) Nshot = int(store(21) + small)

     Nline = 99999
     if(store(22).lt.300.0) Nline = int(store(22) + small)

     das_heading_local = store(heading_ch)  !instrument room gyro
     das_course_local = store(course_ch)    !DAS course

     do i = 1,number_of_compasses           !bird compass headings
       bird_headings(i) = store(compass_chan1 - 1 + i)
     end do

c Check bird headings; if <0 or >360 then replace with last good value
c If first one bad, replace with ship's course
     if(bird_headings(1).gt.360.0 .or.
   +    bird_headings(1) .lt. 0.0) then
        bird_headings_local(1) = das_heading_local
     else
        bird_headings_local(1) = bird_headings(1)
     end if

     do i = 2,number_of_compasses
       if(bird_headings(i).gt.360.0 .or.
   +    bird_headings(i) .lt. 0.0) then
          bird_headings_local(i) = bird_headings_local(i-1)
       else
          bird_headings_local(i) = bird_headings(i)
       end if
     end do

     do i=1, number_of_compasses
       comp_headings(i+1) = bird_headings_local(i)
     end do
     comp_headings(1) = das_heading_local
     course = das_course_local

c Calculate positions of bird compasses relative to ship.
c Extrapolate to tail buoy using last bird compass and known length of cable.
     call compass_sub(number_of_compasses,
   +              comp_headings,
   +              compass_channels,
   +              delta_x, delta_y,
   +              rear_x, rear_y,
   +              compass_x, compass_y,
   +              seismic_channels,
   +              channel_interval, offsets,
   +              course, feather,
   +              num_good_points,
   +              dist_last_channel_to_tail_buoy,
   +              dist_antenna_to_rear_of_ship,
   +              energy_source_offset)

     tail_dx = rear_x
     tail_dy = rear_y
     tail_buoy_feather_angle = feather(num_good_points)
     tail_buoy_dist_from_ship =
   +    sqrt(rear_x * rear_x + rear_y * rear_y)

     do i=1,number_of_compasses
      store(lat_chan_bird_1 + 2*i-2) = delta_x(i+1)
      store(lat_chan_bird_1 + 2*i-1) = delta_y(i+1)
      end do

      call io_wbuff(lun_out, store, 0, iow)

c       if(tail_buoy_feather_angle.gt.45.0) then
c         write(*,'(f8.3,f7.6," f:",f7.1," dx:",f7.0,
c    +     " dy:",f7.0," c:",f7.1)') store(1), store(2),
c    +    tail_buoy_feather_angle, tail_dx, tail_dy,
c    +    course
c       end if

      if(mod(isec,Nprint).eq.0) then
       write(*,'(f8.3,f7.6,x,i3,"/",i5.5,
     +   " dx,dy: ",2f7.1,
     +   " angle: ",f5.1," tail: ",f8.1)',iostat=ios)
     +   store(1), store(2), Nline, Nshot,
     +   tail_dx, tail_dy,
     +   tail_buoy_feather_angle, tail_buoy_dist_from_ship
      end if
 800 end do

 990 call io_closedat(lun_in)
     call io_closedat(lun_out)
     write(*,'(/," COMPASS completed")')

     END




     program cable
c***
c***  CABLE
c***  Author : P. Petkovic
c***  Version: June 1995
c***
c***  CABLE computes the positions of the compasses along the seismic streamer
c***  based on the dx and dy values computed by program COMPASS. Both
c***  programs assume that there are 5 compasses.
c***
c***  An attempt is made to correct the tail-buoy position for selective
c***  availability effects by applying as correction the vector from MX-100
c***  measured position to actual MX-100 position. The position of the
c***  MX-100 antenna relative to the differential GPS antenna is pre-set
c***  in the code. This part of the code has not been thoroughly tested, so
c***  caveat empor.
c***
c***  The code also contains pre-set values as cutoffs for bad dx (cutoff)
c***  and distance from tail-buoy to last compass (maxdist).
c***
c**   CABLE assumes that the input dx channel 1 is the same as the
c**   output channel for latitude 1 (ie out_chan)
c**   The antenna position is in channels lat_chan and lat_chan+1
c**

     implicit none

     character*40 File_in, file_out
     character string*10, progname*12
     character*20 datetime, user

     integer Nrecords, lun_in, lun_out, lun_log, isec, ios
     integer i, out_chan, Nline, Nline_last, temp, proglen
     integer Nprint, tail_lat_chan, out_lat, Nline_chan
     integer Nc, numfields, lat_chan, course_chan, heading_chan
     integer cli$get_value, cl_status, len, iow
     integer bad_count, bad_count_line, bad_xdist, line_rec
     integer bad_tail, bad_pos, bad_dist, bad_dxdy
     integer mx100_lat_chan, tstart
     integer lun_ohis, lun_nhis


     real*8 latitude, longitude, dlat, dlon, tail_lat, tail_lon
     real*8 distance, RHUMB, pi

     real lat_compass(10), lon_compass(10)
     real delta_x(10), delta_y(10), feather, course, heading
     real store(64), radcon, mpnm, unknown, cutoff, maxdist
     real mx100_x, mx100_y, dgps_x, dgps_y, nrp_to_dgps, nrp_to_mx100
     real alpha1, alpha2, beta1, beta2, gamma1, gamma2
c    real t1/161.127/, t2/0.15/    !start of sesimic data
     real t1/160.107/, t2/0.06/    !start of sesimic data
```

```
      logical error/.false./, printed/.false./

      parameter(radcon=57.2957795, Nprint=3600, mpnm=1852.0)
      parameter(Nc=5, unknown=1.0e10, pi=3.1415927, Nline_chan=22)
      parameter (cutoff=800.0, maxdist=1200.0)

c==============================================================
c Get name of user
      call io_getuser(user)

c Read input parameters from screen prompt
      cl_status=cli$get_value('P1',file_in,len)
      cl_status=cli$get_value('P2',file_out,len)
c     cl_status=cli$get_value('P3',string,len)
c     read(string,*) Nc
      cl_status=cli$get_value('P3',string,len)
      read(string,*) out_chan
      cl_status=cli$get_value('P4',string,len)
      read(string,*) lat_chan
      cl_status=cli$get_value('P5',string,len)
      read(string,*) course_chan
      cl_status=cli$get_value('P6',string,len)
      read(string,*) heading_chan
      cl_status=cli$get_value('P7',string,len)
      read(string,*) tail_lat_chan
      cl_status=cli$get_value('P8',string,len)
      read(string,*) mx100_lat_chan

      write(*,'(/," CABLE assumes number of compasses is ",i2)')Nc
      write(*,'(" Cutoff for good delta_x is ",f8.0," m")')cutoff
      write(*,'(" Maximum allowed distance from tail buoy is ",
     +   f8.0," m")')maxdist

      call cvt_dhmsecyr(t1,t2,tstart)   !start of seismic data

c Coordinates of antennae relative to NRP; starbord and fore is +ve (m)
      mx100_x = 3.68
      mx100_y = -0.2
      dgps_x = -5.6
      dgps_y = - 5.3

c Compute alpha, which is the bearing of the antenna when heading = 0
c alpha1 for mx100 and alpha2 for dgps respectively
      if(mx100_y.ne.0.0) then
        alpha1 = atan(mx100_x/mx100_y)
        if(mx100_x.lt.0.0) alpha1 = alpha1 + 180.0
      else
        if(mx100_x.eq.0.0) then
          alpha1 = unknown
        elseif(mx100_x.gt.0.0) then
          alpha1 = 90.0
        else
          alpha1 = 270.0
        end if
      end if
      if(dgps_y.ne.0.0) then
        alpha2 = atan(dgps_x/dgps_y)
        if(dgps_x.lt.0.0) alpha2 = alpha2 + 180.0
      else
        if(dgps_x.eq.0.0) then
          alpha2 = unknown
        elseif(dgps_x.gt.0.0) then
          alpha2 = 90.0
        else
          alpha2 = 270.0
        end if
      end if

c Compute bearings at which x and y coordinates of antennae are zero
      beta1 = atan(-mx100_x/mx100_y)
      gamma1 = atan( mx100_y/mx100_x)
      beta2 = atan(-dgps_x/dgps_y)
      gamma2 = atan( dgps_y/dgps_x)

c Distance of antennae to NRP origin (m)
      nrp_to_dgps = sqrt(dgps_y**2 + dgps_x**2)
      nrp_to_mx100 = sqrt(mx100_x**2 + mx100_y**2)

c Open log file
      cl_status = CLI$GET_VALUE('$LINE',progname,proglen)
      proglen = INDEX(progname,' ')-1
      progname = progname(1:proglen)
      CALL IO_OPENLOG(progname(1:proglen),lun_log)
```

```
c  Open OLD binary file.
      call io_OpenDat(File_in,'old',numfields,lun_in,1)

c  Open NEW binary file.
      call io_OpenDat(File_out,'new',numfields,lun_out,1)

      do while (.true.)
       Nrecords = Nrecords + 1
       Line_rec = Line_rec + 1
       call io_rbuff(lun_in, store,0, ios)
       if(ios .NE. 0) goto 990
       call cvt_dhmsecyr(store(1), store(2), isec)
c        if(isec.lt.tstart) goto 800
       if(isec.lt.tstart) then
c         write(*,'(" Should not come up")')
        goto 800
       endif

       latitude = store(lat_chan)        !antenna position
       longitude = store(lat_chan+1)

       course = store(course_chan)
       heading = store(heading_chan)

c  Compute difference between antennae in N-S and E-W directions based
c  on their positions on the ship relative to the NRP as origin
c       call correction(heading, nrp_to_dgps, nrp_to_mx100,
c   +       alpha1, alpha2, beta1, beta2, gamma1, gamma2, dlat, dlon)

c  compute difference between mx100 position and dGPS position,
c  and subtract the effect of the physical difference between the antennae
c       dlat = latitude - store(mx100_lat_chan) - dlat
c       dlon = longitude - store(mx100_lat_chan+1) - dlon

c  and apply this shift to the tail buoy position.
c       tail_lat = store(tail_lat_chan) + dlat   !tail buoy position
c       tail_lon = store(tail_lat_chan+1) + dlon
      tail_lat = store(tail_lat_chan)            !tail buoy position
      tail_lon = store(tail_lat_chan+1)

      Nline_last = Nline
      if(store(Nline_chan) .lt. 200) then
       temp = int(store(Nline_chan)+.1)
       printed = .false.
      elseif(.not.printed) then
       write(*,'(" bad line number ",f12.0," at ",f8.3,f7.6)',
     +     iostat=ios)
     +     store(Nline_chan), store(1),store(2)
       printed=.true.
      endif

      if(temp.gt.0.and.temp.lt.130) Nline = temp     !survey 160
c       if(temp.gt.0.and.temp.lt.32) Nline = temp      !survey 161


c       if(abs(latitude).gt.pi) write(*,'(" Lat = ",f25.10)')
c       if(abs(longitude).gt.pi) write(*,'(" Long = ",f25.10)')
c       if(abs(tail_lat).gt.pi) write(*,'(" Taillat = ",f25.10)')
c       if(abs(tail_lon).gt.pi) write(*,'(" Taillong = ",f25.10)')
c       if(course.gt.360.0) write(*,'(" Course = ",f25.10)')
c       if(course.lt.0.0) write(*,'(" Course = ",f25.10)')

      if(abs(latitude).gt.pi .or. longitude.gt.pi .or.
     +    course.gt.360.0 .or. course.lt.0.0) then
c       write(*,*) latitude,longitude
c       write(*,*) tail_lat,tail_lon
c         write(*,'(f8.3,f7.6," missing data; course=",f6.1)')
c    +       store(1), store(2), course
        do i=1,Nc
         out_lat = out_chan + 2*i-2
         store(out_lat) = unknown
         store(out_lat+1) = unknown
        end do
      write(*,'(" Bad lat/long or course")')
        feather = unknown
        distance = unknown
        bad_count = bad_count + 1
        bad_count_line = bad_count_line + 1
        bad_pos = bad_pos + 1
        goto 700
      end if
      if (abs(tail_lat).gt.pi .or. tail_lon.gt.pi) then
       tail_lat = unknown
       tail_lon = unknown
```

```
          bad_tail = bad_tail + 1
        endif

        do i=1,Nc
          out_lat = out_chan + 2*i-2

          delta_x(i) = store(out_lat)
          delta_y(i) = store(out_lat+1)

c          if(i.ge.2 .and. i.le.Nc-1) then
c            if(abs(delta_x(i)-delta_x(i-1)).ge.200) then
c              store(out_lat) = unknown
c              store(out_lat+1) = unknown
c              feather = unknown
c              distance = unknown
c
c              bad_count = bad_count + 1
c              bad_count_line = bad_count_line + 1
c              bad_xdist = bad_xdist + 1
c              goto 700
c            end if
c          end if

         if(delta_y(i) .ne. 0.0
     +      .and. abs(delta_x(i)) .lt. cutoff) then
           feather = atan(delta_x(i) / delta_y(i)) * radcon
         else
           store(out_lat) = unknown
           store(out_lat+1) = unknown
           feather = unknown
           distance = unknown
c          write(*,'(f8.3,f7.6," missing deltas")')
c    +        store(1),store(2)
           bad_count = bad_count + 1
           bad_count_line = bad_count_line + 1
           bad_dxdy = bad_dxdy + 1
           goto 700
         end if

c compute latitude and longitude offset wrt antenna
        call dpos(error, delta_x(i),delta_y(i), course, feather,
     +    latitude, dlat, dlon)

        lat_compass(i) = latitude + dlat
        lon_compass(i) = longitude + dlon

       end do

c Compute distance to tail buoy for checking validity of data for last
c compass
       if (tail_lat.lt.pi .and. tail_lon.lt.pi) then
         distance = rhumb(tail_lat, tail_lon,
     +               lat_compass(Nc), lon_compass(Nc))*mpnm
       else
         distance = unknown
         tail_lat = unknown
         tail_lon = unknown
       endif

c      write(*,'(f8.3,f7.6,"  Distance = ",f15.2)') store(1),
c    +                                  store(2),distance
c      write(*,'(" T_lat= ",f15.10," T_lon= ",f15.10," lat_Nc= "
c    +      ,f15.10," lon_Nc= ",f15.10)') tail_lat,
c    +            tail_lon,lat_compass(Nc),lon_compass(Nc)

       if(distance.le.maxdist) then
         do i=1,Nc
           out_lat = out_chan + 2*i-2
           store(out_lat) = lat_compass(i)
           store(out_lat+1) = lon_compass(i)
         end do
         error = .false.

       elseif (tail_lat.eq.unknown .or. tail_lon.eq.unknown) then
         do i=1,Nc
           out_lat = out_chan + 2*i-2
           store(out_lat) = lat_compass(i)
           store(out_lat+1) = lon_compass(i)
         end do

       elseif (distance.gt.maxdist .and. (tail_lat.ne.unknown .or.
     +                          tail_lon.ne.unknown)) then
         do i=1,Nc
           out_lat = out_chan + 2*i-2
```

```
            store(out_lat) = unknown
            store(out_lat+1) = unknown
           end do


c        write(*,'(f8.3,f7.6," L:",i4.4,
c   +       " dx: ",f7.1," dy: ",f7.1,
c   +       " t:",f6.0," f:",f5.1," c:",f7.1,/)',iostat=ios)
c   +         store(1),store(2), int(store(Nline_chan)+.1),
c   +         delta_x(Nc), delta_y(Nc),
c   +         distance, feather, course
          feather = unknown
          distance = unknown
          bad_count = bad_count + 1
          bad_count_line = bad_count_line + 1
          bad_dist = bad_dist + 1
          error = .true.
        end if

  700   call io_wbuff(lun_out, store, 0, iow)

        if(Nline.ne.Nline_last.or.mod(line_rec,200).eq.0)then
         write(*,'(f8.3,f7.6," L:",i4.4,
c   +           4(f9.1,x),
    +          f8.1," m", f8.1," bad:",f6.1,"%",5i5)',
    +          iostat=ios) store(1), store(2), Nline,
c   +      delta_x(1), delta_y(1), delta_x(Nc), delta_y(Nc),
    +     distance, feather, float(bad_count_line)*100.0/Line_Rec,
    +     bad_dist, bad_pos, bad_dxdy, bad_xdist, bad_tail
         write(lun_log,'(f8.3,f7.6," L:",i4.4,
c   +           4(f9.1,x),
    +          f8.1," m", f8.1," bad:",f6.1,"%",5i5)',
    +          iostat=ios) store(1), store(2), Nline,
c   +      delta_x(1), delta_y(1), delta_x(Nc), delta_y(Nc),
    +     distance, feather, float(bad_count_line)*100.0/Line_Rec,
    +     bad_dist, bad_pos, bad_dxdy, bad_xdist, bad_tail
         Line_rec = 0
         bad_count_line = 0
         bad_dist=0
         bad_pos = 0
         bad_dxdy = 0
         bad_xdist = 0
         bad_tail = 0
        end if
  800 end do


c HISTORY COMPONENT
c*************************************************************************
c History file (output) format
 2001 format(' PROCESS: ',a,6X,' DATE CREATED: ',a,6X,'USER: ',a,//,
    +             ' IN:    ',a,/,
    +             ' OUT:   ',a,//,
    + ' # compasses:',i3,' Output ch (comp 1):',i3,' dGPS ch:',i3,
    + ' Course ch:',i3,/,' Heading ch:',i3,' Tail_lat ch:',i3,
    + ' mx100_lat ch: ',i3,/,
    +              70(' '),/,70('-'))


c Open history files and copy.
    call io_copyhis (File_in,File_out,lun_ohis,lun_nhis,datetime)

c Append process details to the history file.
     write(lun_nhis,2001)progname(1:proglen),datetime,user,
    + File_in, File_out, Nc, out_chan, lat_chan, course_chan,
    + heading_chan, tail_lat_chan, mx100_lat_chan


c Close history files.
    close(lun_ohis)
    close(lun_nhis)
c*************************************************************************


 990 call io_closedat(lun_in)
    call io_closedat(lun_out)
    call io_closelog(lun_log)

    write(*,'(/," CABLE completed",/,x,i6,
   +     " records with missing data")')bad_count
    end
c*************************************************************************
    subroutine dpos(error, delta_x, delta_y, course, feather,
   +     latitude, dlat, dlon)
```

```
      implicit none
      integer sgnE, sgnN, ios

      real delta_x, delta_y, course, feather, latitude, beta
      real*8 dlat, dlon, dN, dE, radcon

      logical error
      parameter (radcon=57.29577951)
c=========================================================================
c Let beta be the bearing of compass from Antenna
      beta = 180.0 + course - feather
      if(beta.gt.360.0) then
       beta = beta - 360.0
      elseif(beta.lt.0.0) then
       beta = beta + 360.0
      end if

c     beta = amod(beta,360.0)

c Work out sign of dE and dN
      if(beta.eq.180.0.or.beta.eq.0.0) then
       sgnE = 0
      elseif(beta.gt.180.0) then     !bird is west of antenna
       sgnE = -1
      else                           !bird is east of antenna
       sgnE = +1
      end if

      if(beta.eq.90.0.or.beta.eq.270.0) then
       sgnN = 0
      elseif(beta.lt.90.0.or.beta.gt.270.0) then
       sgnN = +1
      else
       sgnN = -1
      endif

c work out magnitude of dN, dE
      if(abs(beta).ne.90.0.and.abs(beta).ne.270.0) then
       beta = beta / radcon
       dN = sqrt((delta_x**2 + delta_y**2)/(1d0 + tan(beta)**2))
       dE = abs(tan(beta)) * dN
      else
       dN = 0.0
       dE = sqrt(delta_x**2 + delta_y**2)
      end if

      dLat = sgnN * dN / (1852.0d0 *60.0d0 * radcon)
      dLon = sgnE * dE / (1852.0d0 * cos(latitude) * 60.0d0 * radcon)
      if(dlat.gt.1 .or. dlon.gt.1) then
       write(*,'(" Dlat or dlon gt 1",2f22.10)') dlat,dlon
      endif

      error = .false.
      if(error) then
       write(*,'(" dx:",f6.1," dy:",f7.1," b:",f6.1,
     +   " dN:",f7.1," dE:",f7.1,
     +   " dLat:",f8.4," dLon:",f8.4)',iostat=ios)
     +        delta_x, delta_y, beta*radcon,
     +        sgnN*dN, sgnE*dE, dLat*radcon, dLon*radcon
      end if

      return
      end
c*************************************************************************
      subroutine correction(heading, nrp_to_dgps, nrp_to_mx100,
     +   alpha1, alpha2, beta1, beta2, gamma1, gamma2, dlat, dlon)
c***
c*** Compute the latitude and longitude difference in position between the
c*** two antennae.
c** heading - heading of ship
c** nrp_to_dgps - distance NRP to dgps antenna
c** nrp_to_mx100 - distance NRP to mx100 antenna
c** alpha1 - bearing of mx100 antenna when bearing of ship is zero
c** alpha2 - bearing of dgps antenna when bearing of ship is zero
c** beta1 - bearing of ship when x coordinate of MX100 antenna is zero
c** beta2 - bearing of ship when x coordinate of dgps antenna is zero
c** gamma1 - bearing of ship when y coordinate of MX100 antenna is zero
c** gamma2 - bearing of ship when y coordinate of dgps antenna is zero
c** dlat - latitude difference between antennae
c** dlon - longitude difference between antennae

      implicit none
      integer sgnX, sgnY
      real heading, nrp_to_dgps, nrp_to_mx100, bearing
      real dgps_y, dgps_x, mx100_x, mx100_y, alpha

      real*8 dlat, dlon, alpha1, alpha2, beta1, beta2, gamma1, gamma2
      real*8 radcon, mpnm

      parameter (radcon=57.29577951, mpnm=1852.0)
c=====================================================================

      if(bearing .lt. beta1)  then
        sgnX = +1
      elseif(bearing .gt. beta1) then
        sgnX = -1
      else
        sgnX = 0
      end if
      if(bearing .lt. gamma1)  then
        sgnY = +1
      elseif(bearing .gt. gamma1) then
        sgnY = -1
      else
        sgnY = 0
      end if

c  coordinates of MX100 antenna relative to NRP origin
      alpha = (alpha1 + heading) / radcon
      mx100_x = sgnX * nrp_to_mx100 * sin(alpha)
      mx100_y = sgnY * nrp_to_mx100 * cos(alpha)

      if(bearing .lt. beta2)  then
        sgnX = +1
      elseif(bearing .gt. beta2) then
        sgnX = -1
      else
        sgnX = 0
      end if
      if(bearing .lt. gamma2)  then
        sgnY = +1
      elseif(bearing .gt. gamma2) then
        sgnY = -1
      else
        sgnY = 0
      end if

c  coordinates of dGPS antenna relative to NRP origin
      alpha = (alpha2 + heading) / radcon
      dgps_x = sgnX * nrp_to_dgps * sin(alpha)
      dgps_y = sgnY * nrp_to_dgps * cos(alpha)

      dLon = (dgps_x + mx100_x) / mpnm / 60.0d0 / radcon  !radians
      dLat = (dgps_y + mx100_y) / mpnm / 60.0d0 / radcon

      return
      end



      subroutine ADJUST_work(lun_in,lun_out,lun_log,Nc,prec,Nprint)

c***           ADJUST_WORK
c***
c***   Author:  Robert Parums
c***   Created:  August 1995
c**
c**   Adjust compass locations by rotating by a fixed angle (given by the angle between source-
c**   last compass and source-tailbuoy) around the source.
c**   It accepts a compass location file and outputs adjusted compass locations.
c**
c*    This program assumes navigation is in high precision format
c*
c*    Parameters:
c*    lun_in -  LUN of input file
c*    lun_out - LUN of output file
c*    lun_log - LUN of log file
c*    Nc      - number of compasses
c*    prec    - precision of input lat/lons
c*    Nprint - Number of shots
c*    lat    - latitude string
c*    lon    - longitude string
c*    dlat   - latitude (decimal degrees)
c*    dlon   - longitude (decimal degrees)
c*    alat   - adjusted latitude string
c*    alon   - adjusted longitude string
c*    adjlat   - adjusted latitude (decimal degrees)
```

```
c*     adjlon   - adjusted longitude (decimal degrees)
c*
c*********************************************************************************

      implicit none

      integer lun_in, lun_out, lun_log, Nprint, Nrec/0/
      integer ios, ion, i, j, Nc, N(6)

      character*(*) prec
      character*206 record
      character*80 str1
      character*21 str2
      character*40 io_errmsg
      character*10 lat(7), alat(5), nolat
      character*11 lon(7), alon(5), nolon

      real*8 dlat(7), dlon(7)
      real*8 adjlat(5), adjlon(5), bear, bear_tb, bear_lc
      real*8 rhumb, rad, raddeg, dist(7)

      real unknown, mpnm, dx(6)


      logical interactive, write_log

      parameter (unknown=1.0E10, raddeg=57.29577951, mpnm=1852.0)
      parameter (nolat='999999.99S', nolon='9999999.99S')  !missing coords

c*********************************************************************************
      do while (ion.eq.0)
       Nrec=Nrec + 1
       read(lun_in,'(a)',iostat=ion,end=100) record

       read(record(1:80),'(a80)',iostat=ion) str1
       read(record(24:33),'(a10)',iostat=ion) lat(1)
       read(record(34:44),'(a11)',iostat=ion) lon(1)
       read(record(81:90),'(a10)',iostat=ion) lat(2)
       read(record(91:101),'(a11)',iostat=ion) lon(2)
       read(record(102:111),'(a10)',iostat=ion) lat(3)
       read(record(112:122),'(a11)',iostat=ion) lon(3)
       read(record(123:132),'(a10)',iostat=ion) lat(4)
       read(record(133:143),'(a11)',iostat=ion) lon(4)
       read(record(144:153),'(a10)',iostat=ion) lat(5)
       read(record(154:164),'(a11)',iostat=ion) lon(5)
       read(record(165:174),'(a10)',iostat=ion) lat(6)
       read(record(175:185),'(a11)',iostat=ion) lon(6)
       read(record(186:195),'(a10)',iostat=ion) lat(7)
       read(record(196:206),'(a11)',iostat=ion) lon(7)
       read(record(186:206),'(a21)',iostat=ion) str2

c Break up into source,compasses and tailbuoy components.
c If source, last compass or tailbuoy data not available, write all
c  compass data to output as is (unadjusted).
      do i=1,Nc+2
       if(prec.eq.'H' .and. (lat(1).eq.nolat .or. lon(1).eq.nolon.or.
     +       lat(Nc+1).eq.nolat .or. lon(Nc+1).eq.nolon .or. lat(Nc+2)
     +                  .eq.nolat .or. lon(Nc+2).eq.nolon)) then
         do j=2,Nc+1
          alat(j)=lat(j)
          alon(j)=lon(j)
         enddo
         goto 2100
       endif
       if(prec.eq.'H' .and. (lat(i).eq.nolat .or. lon(i).eq.nolon))
     +                                       then
        dlat(i)=unknown
        dlon(i)=unknown
       else
        call cvt_lldeg(prec,lat(i),lon(i),dlat(i),dlon(i))  !decimal degrees
        dlat(i)=dlat(i)/raddeg
        dlon(i)=(dlon(i) - 100.0d0)/raddeg
       endif
      end do

      bear_tb = bear(dlat(1),dlon(1),dlat(7),dlon(7))
      bear_lc = bear(dlat(1),dlon(1),dlat(6),dlon(6))
      rad = bear_tb - bear_lc

c Rotate the (radian) location of each of the compasses
      call rotate (dlat, dlon, rad, adjlat, adjlon, Nc)   !In radians

c Compute the distance adjusted for each compass
      do i=2,Nc+1
```

```
          if(adjlat(i).ne.unknown) then
           dist(i)=rhumb(dlat(i),dlon(i),adjlat(i),adjlon(i))*mpnm
           adjlat(i)=adjlat(i)*raddeg              !Back in decimal degrees
           adjlon(i)=adjlon(i)*raddeg + 100.0d0    !  "   "   "    "
           call cvt_degll(prec,alat(i),alon(i),adjlat(i),adjlon(i)) !lat/lon strings

           N(i)=N(i)+1
           dx(i)=dx(i)+dist(i)
          else
           alat(i)=nolat
           alon(i)=nolon
          endif
         end do

c  Write final values to output file
2100   if (prec.eq.'H') then
        write(lun_out,'(a80,<Nc>(a10,a11),a21)')
     +                  str1,(alat(i),alon(i),i=2,Nc+1),str2
       endif


       if (mod(Nrec,Nprint).eq.0) then
c        do i=2,Nc+1
c         write(*,'(i3,x," Raw: ",a10,2x,a11," Adj: ",a10,2x,
c    +                  a11)')i, lat(i),lon(i), alat(i), alon(i)
c        end do
        write(*,'(i7,<Nc>("  ",i2,":",f6.2,"m  "),/)')
     +                              Nrec,(i,dist(i),i=2,Nc+1)
       endif


      end do

100   continue

      write(*,'(<Nc>(" Compass ",i2,2x,i7,x,f7.2"m",/))')
     +                      (i-1,(N(i),dx(i)/N(i)), i=2,6)


      end
c******************************************************************************
      subroutine rotate(xold,yold,rad,xnew,ynew,Nc)

c     Author:  David Collins
c     Version:  August 1992
c     Modified: August 1995  (R. Parums)
c
c     Calculate new position (xnew,ynew) after rotating a
c     point at (xold,yold) by an angle of rad radians around the origin(0,0)
c     in an anti-clockwise direction.
c
c     This subroutine has been modified to rotate about the seismic source
c     location followed by a translation.       (R. Parums 26/8/95)
c
c
c      Note: cos(pie/2) computes to -4.37114E-08 for a real*4 (not 0)
c      Thus the value for tiny is set at 1E-07
c******************************************************************************
      real unknown
      parameter (tiny=1E-07, unknown=1.0e10)
      real*8 xold(7), yold(7), rad, xnew(7), ynew(7)
      integer Nc


      cosrad=cos(rad)
      if (abs(cosrad).lt.tiny) cosrad=0

      do i=2,Nc+1
       if (xold(i).ne.unknown) then
        xnew(i) = xold(1) + (xold(i)-xold(1))*cosrad -
     +                      (yold(i)-yold(1))*sin(rad)
        ynew(i) = yold(1) + (xold(i)-xold(1))*sin(rad) +
     +                      (yold(i)-yold(1))*cosrad
       else
        xnew(i)=unknown
        ynew(i)=unknown
       endif
      enddo
      end
```

```
C*******************************************************************************
      subroutine map_streamer(lun_in, lun_log, ShotSymFreq, ShotLblFreq,
     +  ll_flag, tlat, blat, elong, wlong, Npoints, file_type, aflag,
     +                  addlines, Nls, dbflag, pen)

c***
c***    Map streamer positions either from compass location file or shot/rec file.
c***
c***    Author:  Peter Petkovic
c***    Version: June 1995
c***    Modified: R. Parums,  October 1995
c***
c*
c*      lun_in - logical unit number of input file
c*      lun_log - logical unit number of log file
c*      Shotsymfreq - Shot symbol frequency
c*      Shotlblfreq - Shot label frequency
c*      ll_flag - line label flag (not used at present)
c*      tlat - top latitude
c*      blat - bottom latitude
c*      elong - easterly longitude
c*      wlong - westerly longitude
c*      Npoints - number of points
c*      file_type - type of streamer file (.STR or .REC)
c*      aflag - addlines flag
c*      addlines - array of linenames to plot
c*      Nls - number of lines
c*      pen - pen colour
c*      dmatch - double line strength flag
c*
c*
C*******************************************************************************
      implicit none

      include 'mapper.fin'      !58 lines
      character*206 record
      character*40 io_errmsg
      character*80 compass_rec(12)
      character*16 Line_label, last_line
      character*3 file_type
      character*(*) addlines(200)

      integer lun_in, lun_log, survey, done, Npoints, pen
      integer i, j, sec, ios, recnum/0/, plotflag
      integer ShotLblFreq, ShotSymFreq, len, len1
      integer shotnum, Ncomp, ll_flag, recN, Nls

      real*4 buffer(64)
      real*4 tlat,blat,elong,wlong
      real*4 top_limit,base_limit,west_limit,east_limit
      real*8 stnlat,stnlong, shotlat, shotlon, x, y
      real*4 big, missing

      real*8 raddeg
      parameter(raddeg=57.295 779 51d0, big=1.0e9, missing=1.0e10)
      parameter(Ncomp=5)

      logical complete/.true./, tb/.true./, dbflag, aflag
      logical match/.false./

c*******************************************************************************
      border = 0.0                    !temporary fix to annoying bug!

      top_limit  = tlat  + border
      west_limit = wlong - border
      base_limit = blat  - border
      east_limit = elong + border

      write(*,'(/," Boundary N,S,W,E: ",4(f10.6,x))')
     +    top_limit, base_limit, west_limit, east_limit
      write(*,'(/," Border: ",f10.6," deg")')border

c Skip headers if they exist
      read(lun_in,'(a)',iostat=ios) record
      if(record(:1).eq.'#') then
        do while (.true.)
          read(lun_in,'(a)',iostat=ios) record
          if(ios.ne.0) then
            write(*,'(" Error ",i3," while reading headers")')ios
            stop ' '
          end if
          if(record(:1).eq.'#') goto 100
        end do
      else
       rewind (lun_in)
      end if

 100  recnum = 1


      do while (.true.)

c  Skip any lines not included in the control file line list.
      if(file_type.eq.'REC') then
       if (line_label.ne.last_line) then
         match=.false.
         if (aflag) then
           do i=1,Nls
            len1=index(addlines(i),' ')-1
            if (line_label(:len).eq.addlines(i)(:len1)) then
              match=.true.
            endif
           end do
         elseif (.not. aflag) then
           match=.true.
         endif
       endif
      endif
       if (.not.match .and. file_type.eq.'REC') goto 2000    !Skip the loop


       last_line = Line_label

       if (file_type.eq.'STR') then
        call getstreamer(record, Line_label, buffer)
        if (line_label.ne.last_line) then
          match=.false.
          if (aflag) then
            len=index(line_label,' ')-1
            do i=1,Nls
             len1=index(addlines(i),' ')-1
             if (line_label(:len).eq.addlines(i)(:len1)) then
               match=.true.
             endif
            end do
          elseif (.not. aflag) then
            match=.true.
          endif
        endif
        if (.not.match) goto 2000              !Skip the loop

       elseif(file_type.eq.'REC' .and. recN.ne.241) then
        goto 2000
       endif

c  Determine if whole streamer lies within map boundary. If not, skip
c  the plotting

       do i=1,Ncomp + 2
        stnlat  = dble(buffer(2*i+1)*raddeg)
        stnlong = dble(buffer(2*i+2)*raddeg) + 100.0d0
        if ((stnlat.ge.tlat .or. stnlat.le.blat .or.
     +         stnlong.le.wlong .or. stnlong.ge.elong) .and.
     +  buffer(2*i+1).ne.missing .and. buffer(2*i+2).ne.missing) then
            complete=.false.
            goto 333
        else
            complete=.true.
        end if
       end do
333    continue

c  Determine whether the tailbuoy data exists
      if ((buffer(2*(Ncomp+2)+1).eq.missing
     +           .or. buffer(2*(Ncomp+2)+2).eq.missing))tb=.false.


       do i=1,Ncomp + 2

        stnlat  = dble(buffer(2*i+1)*raddeg)
        stnlong = dble(buffer(2*i+2)*raddeg) + 100.0d0
        shotnum = int(buffer(2))

c       Check if station is within map border. If so, plot the station.
c       Pass all lats/longs within the boundary AS WELL AS unknown values.
        if (stnlat.lt.top_limit .and. stnlat.gt.base_limit .and.
     +     stnlong.gt.west_limit .and. stnlong.lt.east_limit .or.
     +  (buffer(2*i+1).eq.missing .and. buffer(2*i+2).eq.missing))then

           if(i.eq.1) then
            if(buffer(2*i+1).eq.missing .and. buffer(2*i+2).eq.
     +                                 missing) goto 2000
             call map_shot(lun_log,line_label,shotnum,stnlat,stnlong,
     +         tlat, blat, elong, wlong, ShotSymFreq, ShotLblFreq,
     +              ll_flag, charht, symht, symtype, dbflag)

             shotlat=stnlat         !Keep coords of local shotpoint
             shotlon=stnlong

             Npoints = Npoints + 1   !count number of points plotted
             if (.not.complete) then
               goto 2000
             endif

           else
            if(( mod(shotnum,ShotSymFreq).eq.0 .or.
     +                        line_label.ne.last_line)) then

              if(.not. tb) then
                call plt_newpen(7)          !Red, Signifies no TB data
              else
                call plt_newpen(mod(pen,8)+1)        !Change pen colour
              endif
             if (buffer(2*i+1).ne.missing .and. buffer(2*i+2).ne.
     +                                   missing) then
               call prj_getxy(stnlat,stnlong,x,y)
               call plt_plot(x,y,2)              !Draw streamer
c                call plt_newpen(1)              !Change pen colour
c                call plt_symcen(x,y,symht/2.0,12,0)  !Plot marker at compass
c                call plt_newpen(mod(pen,8)+1)       !Change pen colour
             endif

             if (i.eq.Ncomp+2) then             !If at tailbuoy
              if(buffer(2*i+1).ne.missing .and. buffer(2*i+2)
     +                                .ne.missing) then
c                 call plt_symcen(x,y,0.02,3,0)      !Plot marker at TB
              endif
                call prj_getxy(shotlat,shotlon,x,y)
                call plt_plot(x,y,3)             !Put pen(up) at shot locn
                call plt_newpen(pen)             !Change pen colour back
                tb = .true.
             endif
           end if
          end if
        endif

       end do

2000   recnum=recnum+1
      read(lun_in,'(a)',iostat=ios,end=999)record
c       if (.not.match) goto 3000
      if(file_type.eq.'REC') then
        read(record(18:21),'(i4)',iostat=ios) recN
        if(recN.eq.1) then
          compass_rec(1)=record
        elseif(recN.eq.9+1) then
          compass_rec(2)=record
        elseif(recN.eq.73+1) then
          compass_rec(3)=record
        elseif(recN.eq.137+1) then
          compass_rec(4)=record
        elseif(recN.eq.185+1) then
          compass_rec(5)=record
        elseif(recN.eq.233+1) then
          compass_rec(6)=record
        elseif(recN.eq.240+1) then
          compass_rec(7)=record
          call decode_receiver(compass_rec,line_label,buffer)
          line_label=line_label//'     '
          len=index(line_label,' ')-1
        endif
      endif

      if(ios.ne.0) then
        write(*,'(" Error",i5," reading record ",i6)')ios, recnum
        write(*,*) io_errmsg(ios)
        stop ' '
      end if

3000 enddo
```

```
 999 line_label = ' '
     call map_shot(lun_log, line_label, shotnum, stnlat, stnlong,
    +      tlat, blat, elong, wlong, ShotSymFreq, ShotLblFreq,
    +                ll_flag, charht, symht, symtype, dbflag)

     end
C*******************************************************************************
     subroutine getstreamer (record, Line_label, buffer)
c***
c***  Extract data from STREAMER record
c***  Assumes data from compass location file
c**

     implicit none

     character*(*) record
     character*(*) line_label
     character*1 latH, lonH

     real buffer(*), latS, lonS
     real*8 raddeg, missing, small, flat, flon
     parameter(raddeg=57.295 779 51d0, small=0.1, missing=1.0e10)

     integer Ncomp
     parameter(Ncomp=5)

     integer day, k, m, i, survey, latD, latM, lonD, lonM
     integer ios, line_num, shot_num
c-------------------------------------------------------------------

     read(record(:16),'(a)',iostat=ios) line_label

     read(record(18:23),*,iostat=ios) shot_num
     buffer(2) = float(shot_num) + small

     read(record(24:25),*,iostat=ios) latD
     read(record(26:27),*,iostat=ios) latM
     read(record(28:32),'(f5.2)',iostat=ios) latS
     read(record(33:33),'(a)',iostat=ios) latH

     read(record(34:36),*,iostat=ios) lonD
     read(record(37:38),*,iostat=ios) lonM
     read(record(39:43),'(f5.2)',iostat=ios) lonS
     read(record(44:44),'(a)',iostat=ios) lonH

     buffer(3) = (float(latD) + float(latM)/60.0 +
    +  latS/3600.0) / raddeg
     if(latH.eq.'S') buffer(3) = - buffer(3)

     buffer(4) = (float(lonD) + float(lonM)/60.0 +
    +  lonS/3600.0 - 100.0) / raddeg
     if(latH.eq.'W') buffer(4) = - buffer(4)

     do i=2,Ncomp+2
      m = 21*i + 39
      read(record(m:m+1),*,iostat=ios) latD
      read(record(m+2:m+3),*,iostat=ios) latM
      read(record(m+4:m+8),'(f5.2)',iostat=ios) latS
      read(record(m+9:m+9),'(a)',iostat=ios) latH

      read(record(m+10:m+12),*,iostat=ios) lonD
      read(record(m+13:m+14),*,iostat=ios) lonM
      read(record(m+15:m+19),'(f5.2)',iostat=ios) lonS
      read(record(m+20:m+20),'(a)',iostat=ios) lonH

      read(record(m:m+8),'(f9.2)',iostat=ios) flat
      read(record(m+10:m+19),'(f10.2)',iostat=ios) flon


      if(flat.eq.999999.99) then
        buffer(2*i+1)=missing   !Set to unknown if 9's
        goto 112
      endif
      buffer(2*i+1) = (float(latD) + float(latM)/60.0 +
    +  latS/3600.0) / raddeg
      if(latH.eq.'S') buffer(2*i+1) = - buffer(2*i+1)

 112    if(flon.eq.9999999.99) then
        buffer(2*i+2)=missing   !Set to unknown if 9's
        goto 113
      endif
      buffer(2*i+2) = (float(lonD) + float(lonM)/60.0 +
    +  lonS/3600.0 - 100.0) / raddeg
      if(latH.eq.'W') buffer(2*i+2) = - buffer(2*i+2)
 113    continue
        end do

        return
        end
C*******************************************************************************
        subroutine decode_receiver(compass_rec, Line_label, buffer)

c***
c***  Extract data from SHOT/RECEIVER records
c***  Assumes data from receiver location file (Source at every 241st record)
c***
c***  Written by Robert Parums, September 1995
c**
c**   compass_rec - Array of records containing compass locations.
c**
c**

       implicit none

       character*(*) compass_rec(12)
       character*(*) line_label
       character*1 latH, lonH

       real buffer(*), latS, lonS
       real*8 raddeg, missing, small, flat, flon
       parameter(raddeg=57.295 779 51d0, small=0.1, missing=1.0e10)

       integer Ncomp
       parameter(Ncomp=5)

       integer day, k, m, i, survey, latD, latM, lonD, lonM
       integer ios, line_num, shot_num
c-----------------------------------------------------------------

       do i=1,Ncomp+2

        read(compass_rec(i)(:10),'(a)',iostat=ios) line_label
        read(compass_rec(i)(11:17),*,iostat=ios) shot_num

        buffer(2) = float(shot_num) + small

        read(compass_rec(i)(24:25),*,iostat=ios) latD
        read(compass_rec(i)(26:27),*,iostat=ios) latM
        read(compass_rec(i)(28:32),'(f5.2)',iostat=ios) latS
        read(compass_rec(i)(33:33),'(a)',iostat=ios) latH

        read(compass_rec(i)(34:36),*,iostat=ios) lonD
        read(compass_rec(i)(37:38),*,iostat=ios) lonM
        read(compass_rec(i)(39:43),'(f5.2)',iostat=ios) lonS
        read(compass_rec(i)(44:44),'(a)',iostat=ios) lonH


        if(flat.eq.999999.99) then
          buffer(2*i+1)=missing    !Set to unknown if 9's
          goto 112
        endif
        buffer(2*i+1) = (float(latD) + float(latM)/60.0 +
     +  latS/3600.0) / raddeg
        if(latH.eq.'S') buffer(2*i+1) = - buffer(2*i+1)

 112    if(flon.eq.9999999.99) then
          buffer(2*i+2)=missing    !Set to unknown if 9's
          goto 113
        endif
        buffer(2*i+2) = (float(lonD) + float(lonM)/60.0 +
     +  lonS/3600.0 - 100.0) / raddeg
        if(latH.eq.'W') buffer(2*i+2) = - buffer(2*i+2)
 113    continue
       end do

       return
       end
```

Appendix 11 - Awk Programs

The two awk scripts shown below were used on the shot/receiver file to (i) reformat the file to the original line and shot and (ii) check the reformatted shot/receiver file for shot and receiver continuity. Both these programs are invoked by the command:

awk -f *scriptfile inputfile*

(i) Reformatting

```
#     This program reformats the receiver file from Petroseis format to original
#     linename and shot, negates water depths and resequences the receiver numbers.
#     Author:  R. Parums,  Oct 1995

                    {i=index($1,"-");
                     shot=substr($1,i+3);
                     line="95A-"substr($1,1,2);

                     recN=$2-1
                     wdepth=-$6
                     printf ("%-10s%7d%4d   %19s%8d%8d%5d          \n",
line,shot,recN,$3,$4,$5,wdepth)

                    }
```

(ii) Checking

```
#     This program checks the receiver file after reformatting back to
#     original line-shot-receiver format.
#
#     Author:  R. Parums,  Oct 1995

BEGIN               { recN=240 }

                    {
                      N++;
                      line = $1;
                      shot = $2;
                      if (line!=lastline){
                         if (N != 0)
                           print "                              "lastshot
                         print " LINE: "line"    "shot
                         lcount++;
                      }
                     if (shot != lastshot){
                      if ($3 != 0)
                         print "** Line does not start at receiver #0    ",N
                      if (recN != 240)
                    print "** Line does not start at receiver #240 ",N
                  }
                 recN = $3;
                 lastline = line;
                 lastshot = shot;
                }

END             {   print " "lcount"  Mimpex lines read";
                    print" " N" records read"
                }
```