1

Processing ADCP Data With the CODAS Software System Version 3.1

Eric Firing
Julie Ranada
Joint Institute for Marine and Atmospheric Research
University of Hawaii
USA
and

Patrick Caldwell
National Oceanographic Data Center

November 13, 1995

*The CODAS was originally designed by Ramon Cabrera in collaboration with Eric Firing at the University of Hawaii. The software continues to be supported by Eric Firing and staff. The present CODAS is version 3.1. Many individuals have contributed to the software development and documentation: Frank Bahr, Willa Zhu, Mei Zhou, Ramon Cabrera, and the authors listed above. The National Oceanographic Data Center sponsored the development of several of the data extraction features and participated in the creation of this manual.

Contents

1 Introduction

Processing ADCP data well is not a trivial task, but neither is it so difficult that it should limit their collection or use. In this manual we will discuss a data processing system that was developed at the University of Hawaii. It is centered around a hierarchical database called CODAS, short for Common Oceanographic Data Access System. Developing this type of software requires considerable time and effort, so we suggest that new ADCP users look into using existing systems rather than developing their own from scratch. The UH software was produced with this in mind, and is readily available.

This manual is partitioned into three major sections: following an overview and general comments in this chapter, the individual processing steps are described in detail in the ensuing chapters. Examples from previously collected ADCP datasets are included for illustration.

A documentation file, `process.doc`, is included with the CODAS software. This file summarizes the discussion in this manual with instructions on the various processing steps. For users anxious to get started, one can read the overview below then begin following the procedures in process.doc, while referring to the more detailed discussion in this manual as necessary.

The CODAS package has gained popularity in the last several years. The system has been used at UH since 1988 and has been distributed to over 32 agencies in 14 countries. The National Oceanographic Data Center (NODC) has adopted CODAS as the archive standard for the high-resolution shipboard ADCP data [1].

2 Overview

A brief description of the data acquisition methodology of shipboard ADCP systems and a first glimpse of post-cruise processing is presented. It is intended as a general primer for first-time users of ADCP data. For experienced users, please jump to Section 2.2, which describes the basic functions of shipboard ADCP processing.

2.1 Methodology of Data Acquisition

The hull-mounted ADCP estimates horizontal and vertical velocity as a function of depth by using the Doppler effect to measure the radial relative velocity between the instrument and scatterers in the ocean. Three acoustic beams in different directions are the minimal requirement for measuring the three velocity components. A fourth beam adds redundancy and an error estimate. The ADCP transmits a ping from each transducer element roughly once per second. The echo arrives back at the instrument over an extended period, with echos from shallow depths arriving sooner than ones from greater ranges. Profiles are produced by range-gating the echo signal, which means the echo is broken into successive segments called depth bins which

correspond to successively deeper depth ranges. The operator configures the length of each depth bin and the transmit pulse, which determines the degree of averaging in the vertical, depending on whether one is interested more in vertical resolution or profile penetration. The noisy velocity estimates from each ping are vector-averaged into 1- to 10-minute ensembles. The relative velocities are rotated from the transducer's to the earth's reference frame using the ship's gyrocompass. Finally, relative velocities and various ancillary parameters are stored on the ship using a data acquisition system (DAS) which also optionally records navigation information, such as provided by the GPS. Specifics of the instrument capabilities and configuration options are well documented [5].

Routine processing, quality control, and calibration are performed at the host institute. Standard checks include detecting and correcting time errors, applying transducer-level temperatures and salinities to obtain a better estimate of the sound speed for the velocity calculation, editing out bad bins or profiles that have been contaminated by interference with the bottom or some other physical object such as a hydro wire, and verifying the quality of the gyrocompass and the navigation data. The final gyrocompass estimates of ship heading and the navigation information are the primary sources for calibrating the ADCP's relative current velocities. Typically, one is correcting for a "angle" error due to misalignment of the transducer relative to the ship's hull and an "amplitude" component related mostly to minor imperfections of the transducer geometry. Relative current velocity errors caused by these components are orthogonal; the angle errors lead to uncertainties of the athwartships velocity component while the amplitude error introduces uncertainties along the ship track.

The navigation calculation is performed once calibration is complete. Absolute currents over a fixed depth range (reference layer) are obtained by subtracting the average of the ship velocity relative to a reference layer (i.e. ADCP velocities) from the absolute ship velocity over the ground (from navigation, i.e., GPS). The raw absolute current velocities relative to the reference layer are smoothed to reduce the effects of noise in the position fixes and combined with the navigation data to obtain the best estimates of ship positions and velocities, which are stored into the data base. Thus, absolute currents at any depth can be determined from the final ship navigation data and the relative ADCP measurements.

2.2 The Basic Functions

In the following, "profile" will refer to the recorded ensemble-average of single-ping profiles. With the UH system, the ADCP data processing can be broken down into the following basic steps:

Scan the raw data files to ensure they are readable, to identify gaps or other problems, and to extract information needed to correct the recorded profile times (the usual error source being the the PC clock).

Load the data into a database suitable for processing and analysis.

Evaluate the quality of the dataset as a whole by calculating and plotting diagnostic statistics. Signal strength (as measured by the Automatic Gain Control: AGC), percent good pings, error velocity, vertical velocity, and vertical first difference of the horizontal velocity components are informative. It is useful to compare these variables between on-station and underway periods.

Edit the profiles and the navigation data if necessary.

Calibrate the profiler-gyrocompass combination. Scale factor and rotation calibrations must be determined from all available data as a function of time during the cruise and then used to correct the velocity data.

Reference the relative velocity profiles by calculating the ship's position at the end of each profile and the average velocity of the ship during the profile.

Adjust depth for the difference between the actual vertically averaged sound speed (calculated from hydrographic data) and 1470 m/s, the nominal sound speed assumed by RDI in converting pulse travel times to ranges.

Grid and plot the absolute velocity field in useful ways, such as contoured sections of each horizontal velocity component and maps of velocity vectors at each of several depths.

The order given above is reasonable, but does not have to be followed exactly. Data processing is often done iteratively, beginning with a quick pass (minimal or no editing and calibration). Editing does not have to be done all at once. For example, clipping the bottom of the profiles can be delayed until the data are extracted for plotting. The above outline is summarized in the flow chart shown in Figure 1.

3 The CODAS System

A description of the hardware requirements and an introduction to the use and structure of the CODAS database are provided in this section.

3.1 Hardware Requirements

The UH system is designed to run on a variety of machines starting with a simple PC-compatible and including VAX/VMS and most Unix machines. The minimal configuration is a PC with 640K of RAM, a math coprocessor, and a 40-Mb hard disk. The system has been used extensively with such machines and also with Sun-3 and Sun-4 machines. A PostScript printer is required for plot output.

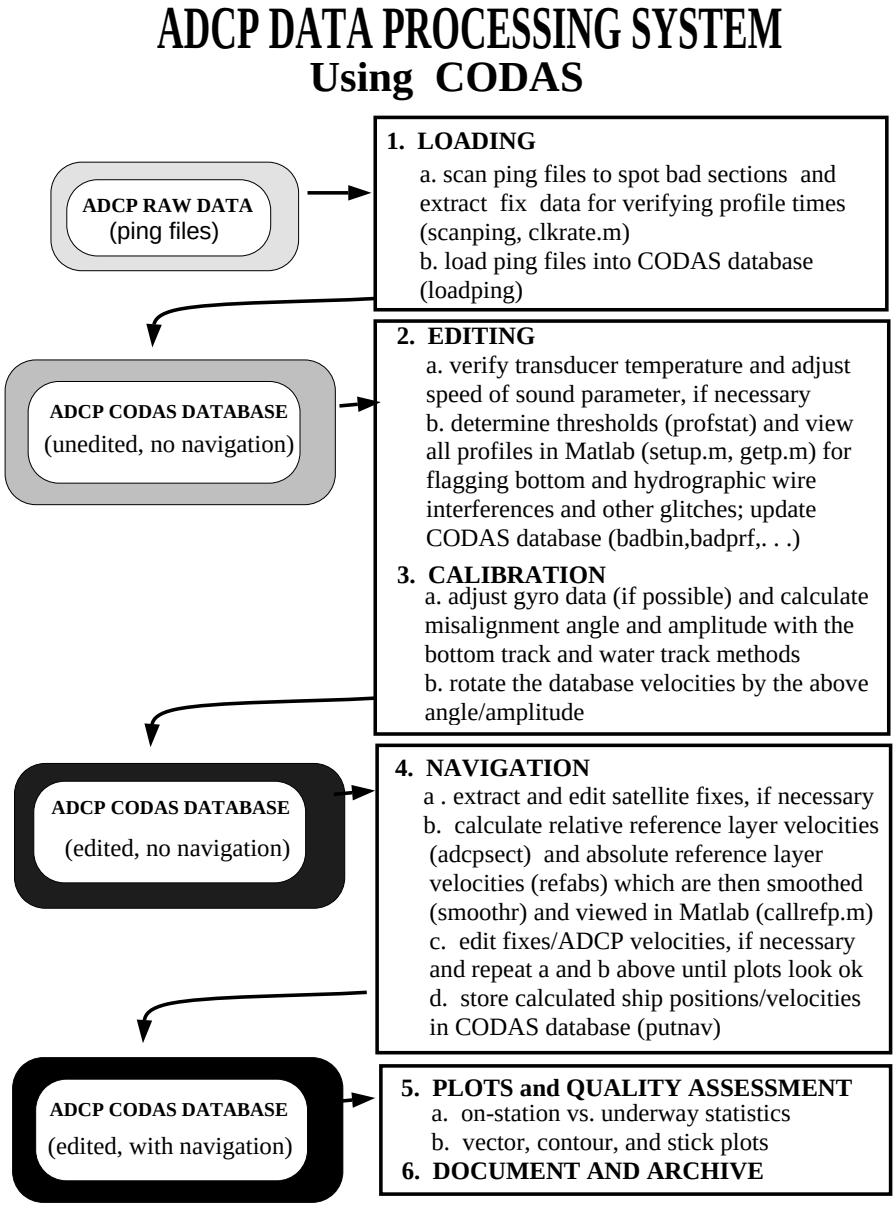


Figure 1: ADCP data processing flowchart.

3.2 User Interface

Most routines in the UH system are governed by control files: ASCII text files that the user writes and modifies with any text editor and that are designed to be read by humans as well as machines. Hence they can be used to document the processing of each data set. All control files can contain unlimited comments; standard sample control files start with a comment block that explains the parameters in the file and gives examples. (Comments are bracketed by `/*` and `*/`, C-style. They can extend over several lines, but cannot be nested.) Each parameter (or list) in a control file is preceded by a descriptive keyword, which identifies the parameter to both the machine and the user. Control files for complicated routines may have optional parameters; when not needed for a particular operation or when program defaults suffice, they can be omitted, shortening and simplifying the file.

Many routines also communicate with each other via intermediate ASCII files with headers and comments, that is, one program's ASCII output file becomes the input to a subsequent program. C programs have been written to recognize lines that begin with a `%` as comments or header lines to be ignored. The user can take advantage of this to incorporate processing commentaries and to reversibly edit out data lines that may contain undesired data. For example, files listing satellite fixes are edited, both mechanically and by hand, by prepending a `%` to lines that contain "bad" fixes. Such lines will be ignored in subsequent processing, and can easily be restored by uncommenting the line and redoing the subsequent processing, where uncommenting consists simply in deleting the `%` character.

One commercial program, Matlab by The MathWorks, Inc., is required for using the UH system. (The unbundled toolboxes sold in addition to the main package are not necessary.) Matlab is available for the PCs and workstations. It provides easy plotting and interactive calculations. For information, contact

The MathWorks, Inc.
Cochituate Place
24 Prime Park Way
Natick, Mass. 01760
phone: (508)653-1415
FAX: (508)653-2997
email: info@mathworks.com

3.3 The CODAS Database

A typical one-month cruise generates about 10 Mb of binary ADCP data including many variables, both scalars and arrays. The variables included can change from one cruise to another. The size and complexity of ADCP datasets therefore warrant the use of a database system rather than a simple fixed file format. To fill this need,

Eric Firing, Ramon Cabrera, and Julie Ranada have developed CODAS (Common Oceanographic Data Access System): a set of machine-independent subroutines for the storage and retrieval of oceanographic and other scientific data. It is designed for maximum storage efficiency combined with fast random or sequential access. It is flexible enough to comfortably accommodate data from a wide variety of instruments, such as the Acoustic Doppler Current Profiler (ADCP), the CTD, current meter moorings, and pressure gauges, together with auxiliary observations and instrument configuration parameters. It is not necessary for the user to understand the internal organization of the data; the user should only be concerned with using the control files and programs for accessing, manipulating, and archiving the database.

CODAS is hierarchical. Data are organized into profiles, each of which may consist of several arrays (for example, one for each of three velocity components) and other data (flags, data collection parameters, notes, etc.) Profiles are collected together in blocks, each of which is an independent, self-describing unit of data. The internal description of each variable in a block includes the name, units, and scale factors. A profile is located within a block via a profile directory that is part of the block. Blocks are catalogued in a block directory.

A CODAS database contains only two kinds of files: a set of data block files, and a single block directory file. The data block files are independent units and contain all the information required to make a block directory file. Hence, data blocks from different sources can be combined into a working database by generating a new block directory file. This is done automatically by a utility program (`mkblkdir`, short for "make block directory").

CODAS was written in C with care taken to make it portable among modern machines. It has been used on IBM PC-compatibles, a VAX 750, an Alliant, and Sun and SGI workstations. Data blocks are normally stored in the binary format of the machine on which they were created or are actively being used. When moved to another machine and assembled into a new database (by the utility program `mkblkdir` provided with the CODAS package), they are automatically translated to the new binary format if necessary (as in going from a PC to a Sun, for example).

Storage space is minimized. The user is free to use 1-, 2-, or 4-byte integers, floating point, double precision, or ASCII. However, use of the most compact format for each variable is encouraged, because routines have been provided to automatically convert and scale array data from any number format to floating point when reading, and the reverse when writing. Storage space is allocated as needed when the data are stored; there is no need to waste space by specifying fixed array lengths, for example. The extensive directory structure adds only a few percent of storage overhead.

4 Setting Up

In the following, program names, commands, specific filenames, directories, and other items that the user must type as is are set in typewriter-like face, as in `program-name`.

Arguments to be supplied by the user are set in *italic*. Optional arguments are further enclosed in square brackets [ ]. (Do not type the brackets.) Directory names are specified in Unix fashion (using forward slashes), and should be appropriately modified for non-Unix systems.

It is assumed that the CODAS software is available on your machine, and that the search path has been modified such that the executable programs can be found from the directory you are working in. The CODAS executables are typically stored in directory `./codas3/bin/‘arch‘`, where the `.` should be replaced by the absolute path to wherever the CODAS package has been unpacked, and the `‘arch‘` should be replaced by the appropriate directory name, depending on the machine being used. On a PC, for example, this can be accomplished by appending `C:\codas3\bin\pc` to the `path` command in the `autoexec.bat` file, assuming that CODAS has been unpacked in the root directory of the `C:` drive. Without the proper path, the location of the CODAS executables must be specified each time a program is executed.

Similarly, it is easiest if the environment variable `MATLABPATH` is properly specified to locate the Matlab M-files (`*.m`) used by CODAS. These are located in directory `./codas3/matlab/matlab*`, where the `*` should be replaced by either `3.5` or `4`, depending on which version of the Matlab package is being used. Users should consult the Matlab manual for their system on how to set the `MATLABPATH`.

The provided sample control files usually carry the same name as the corresponding program, but have the extension `.cnt`. The user may decide to change these names to indicate a particular operation (or cruise) that they were used for. Control files usually begin with a comment section describing the different parameters and options available.

4.1 The directory tree

To keep things organized, we have a standard subdirectory structure to correspond with the basic steps in ADCP processing. Each step is performed primarily within a particular subdirectory. The required control files and other input and processing files are copied from the CODAS ADCP demo to the specific subdirectories where they will be needed. This standard tree structure is rooted at a subdirectory that is named usually after a cruise identifier for uniqueness.

To facilitate setting up, we have devised a script (Unix csh and MS-DOS batch versions are available) to automatically create the ADCP tree structure, and to copy the sample control, definition, and Matlab M-files into the appropriate subdirectories. To execute the script, type:

`adcptree` *[path]cruise-name CODAS-path*

where *cruise-name* identifies the dataset by its cruise name (it should have no more than the number of characters allowed in a subdirectory name), the *path* preceding the *cruise-name* is needed only if the tree structure is to be created in other than

the current subdirectory, and the *CODAS-path* specifies the path to the CODAS subdirectories so the demo control files, etc. can be found and copied over. For example, to set up the processing tree for ADCP data from the cruise known as mw9401, the command to type is:

`adcptree` *mw9401 C:*

again assuming that the CODAS package has been unpacked in the root directory of the PC's `C:` drive.

The resulting tree structure will then start at subdirectory *cruise-name*, under which will be found the subdirectories shown below. They are enumerated in the sequence that they will normally be accessed.

ping — This is where the ADCP ping data files are copied to.

scan — This has a control file and sample user buffer definition files, used for scanning the ping data files. It also has a Matlab M-file used to calculate the time correction parameters (if needed) in the case where the satellite navigation data are stored with the ADCP ping files.

load — This is where the loading of the ping data into a CODAS database takes place. It has the sample `loadping.cnt` file.

adcpdb — This is designated to be the location of the CODAS ADCP database (`.blk` files). Presently, it should have sample producer definition files `*.def`, plus some utility control files that may be needed at a later stage.

quality — This contains control files and Matlab M-files useful for assessing the quality of the ADCP database at any time subsequent to loading.

edit — This is where editing for bottom and wire interference and other glitches is done. It will have several control files and Matlab M-files.

cal — This is where the calibration calculations and rotation take place. It will actually have 4 subdirectories: 1) `botmtrk` for bottom track calibration; 2) `heading` for heading-related calibration; 3) `watertrk` for water track calibration; and 4) `rotate` for database rotation These will have their respective control files, and Matlab M-files where needed.

nav	This is where the navigation calculations take place, which culminates in storing the best estimates of ship speed and position for each profile into the database. It will have several control files and Matlab M-files.
grid	This contains two control files: one for converting a geophysical grid into time ranges, and another for breaking a time range into smaller time intervals of fixed length. The grid output is used as control file input to the plotting steps that follow.
contour	This has a control file for extracting data from the CODAS database. It will also have a sample contour control file (`.cpa`) for use with the UH contour plotting program. (This contour plotting program is also available by anonymous ftp from the same site as the CODAS package. It is written primarily in FORTRAN and comes with executables for Sun 4 workstations. It has not yet been ported to PCs; it was used on VAX/VMS computers at one time. The source code is part of the package so users are welcome to port it to other systems, or may choose to use their own contouring package.)
vector	This has a control file for extracting data, and a control file for use with the vector plotting program. (The vector plotting program is distributed as part of CODAS, and has been ported to all systems that CODAS has been ported to.)
stick	This has a control file for extracting data, and a Matlab M-file for invoking routines that generate stick plots, harmonic analysis, etc.

After running `adcptree` and verifying that the tree structure has been created successfully, the next step is to copy the ADCP ping files to the `ping` subdirectory of the ADCP tree.

4.2 Copying and renaming the ping files

If each of the ADCP ping data files has a unique name, then they can easily be copied to the ping subdirectory in the adcptree structure using whatever the system's copy command is.

If the raw data are collected by a PC onto floppies, then each new floppy will start with the nondescriptive filename `pingdata.000`, and there will be as many files

bearing that name as there are floppies. For this reason, we have a PC program called `pingcopy.exe` to simultaneously generate unique names based on the PC file time stamp while copying the files to the `ping` directory. (There is a related PC program called `ping_ren.exe` that simply renames the files uniquely, without copying them anywhere.) To run the program, type:

`pingcopy` *source-path [destination-path]ship-name*

where *source-path* provides the current location of the ping data files, the *destination-path* directs where they will be copied to, and *ship-name* is, by convention, the two-character National Oceanographic Data Center (NODC)[1] code for the ship aboard which the data were collected. For example, suppose the ping files were on the root directory of the floppy in drive `A:`, and the user is currently in the `mw9401` directory. The command to copy the files would be:

`pingcopy` *a:\ ping\mw*

where *mw* is the NODC code for the R/V Moana Wave. The copied ping files will all have extension `.png` and filename roots that begin with those two characters. The remaining 6 characters of the root filename will encode the time of the original ping file, insuring uniqueness (at least over a 10-year period) by using the following scheme:

character 3 = last digit of the year (0 to 9)

character 4 = hexadecimal month (1 to C)

character 5 = day (1 to 9 then A to V cover 31 days)

character 6 = hour (0 to 9, A to N cover 24 hours)

characters 7 to 8 = minute (00 to 59)

5 Creating a Database

This section describes three stages in loading ADCP ping data into a CODAS database:

1. scanning the ADCP ping data files to verify readability and gather appropriate information
2. loading ADCP ping data into a CODAS database
3. examining the CODAS database to verify loading

[1]For information on NODC codes, contact Mr. Patrick Caldwell, NODC Liaison, Department of Oceanography, 1000 Pope Road MSB 317, Honolulu, HI 96822 (email: caldwell@kapau.soest.hawaii.edu, phone: 808-956-4105). This information is not used by the NODC archiving system. It is recommended as a way for individual users to tag their data sets.

5.1 Scanning the ping files

The program `scanping` scans the binary ping data files to ensure they are readable, to identify gaps or other problems, and to extract information needed to correct the recorded profile times during subsequent loading of the data into the CODAS database. The basic steps are as follows. First, one edits the `scanping.cnt` control file to set the appropriate options and file names. Then, the `scanping` program is executed and the output files are examined. These steps are explained in detail below.

5.1.1 Setting Up

The first step is to prepare `scanping.cnt`. A sample control file is shown in Appendix A.24. In the control file, the parameters `OUTPUT_FILE` and `PINGDATA_FILES` are straightforward, the former indicating the name to use for the `scanping` output file (we like to use the extension `.scn` as a convention to indicate this is output from `scanping`), and the latter signalling the list of ping data filenames to be scanned. The other parameters require more explanation.

The `SHORT_FORM` option can be switched to `yes` or `no`. Setting it to `no` makes scanping list the names of all the variables that have been recorded by the DAS after scanning each header and its component profiles. This is useful when the user does not know exactly which variables the DAS has been configured to record. It does make for verbose output, so normally, we might just set `SHORT_FORM` to `no` and scan one or two ping data files, and then set it back to `yes` and rerun scanping to scan the remainder.

The remaining parameters pertain to the user buffer, which is a special area set aside by the DAS to allow the user to record other variables not collected by the ADCP, but by a user-exit program that is interleaved with the DAS. A common example of a user-exit program is RDI's `navsoft` program. A typical use of the user buffer is for storage of raw satellite fixes. Installation of a user-exit program and the size of the user buffer is configured into the DAS. Setting `SHORT_FORM` to `no` is useful for determining if indeed a user buffer has been recorded. If it is, `scanping` will list `USER_BUFFER_#_BYTES` among the variables recorded, where # is the number of bytes recorded.

There are 2 main reasons for our interest in the user buffer contents at this stage.

1. In the case of non-UH type user buffers (those written by a user-exit program other than those developed at the University of Hawaii), the scanping stage is the only opportunity currently provided for extracting the user buffer contents. If, for example, the user buffer contains navigation information, then extracting this to a file for later use in the navigation calculations should be done at this point.

2. If the user buffer contains information on satellite times, then this can be used as a basis for checking the ADCP ensemble times derived from the PC clock. There

are two types of problems commonly encountered with the PC clock: a) an offset due to an erroneous initial setting. This may vary anywhere from a second to several hours (wrong time zone, AM vs PM), even years; and b) a drift due to either too fast or too slow a clock speed. (Rarer problems include PC clocks that have a habit of resetting themselves.) Synchronizing the ensemble times with the navigation is essential for performing the calibration and navigation calculations later. Correcting for both offset and drift is done during the loading stage,[2] so checking must be done during scanning.

If no user buffer was recorded or if none of the two conditions enumerated above apply, then the `scanping.cnt` parameters `UB_OUTPUT_FILE`, `USER_BUFFER_TYPE`, and `UB_DEFINITION` can all be set to `none`, and the user can skip the remainder of this section and proceed to running `scanping`.

If there is a user buffer which the user wishes to examine at this stage, then it probably is either 1) ASCII-formatted, or 2) a PC binary structure or array, depending on the user-exit program that wrote it.

ASCII-formatted user buffers can be extracted by setting `UB_OUTPUT_FILE` to some output filename, `USER_BUFFER_TYPE` to `ascii`, and `UB_DEFINITION` to `none`. When `scanping` is run, it will print out each ADCP ensemble time to the `UB_OUTPUT_FILE`, along with the contents of the user buffer. The user should note that the ADCP ensemble time corresponds to the ADCP PC time at the end of the ensemble, and must determine (ideally through familiarity with the DAS/user-exit setup) at what point within the ensemble the user buffer contents are acquired.

PC binary-formatted user buffers can also be handled by `scanping`. We distinguish between those that have been written by the UH user-exit program (versions ranging from `720` to `2240`), and those written by some other user-exit program (`USER_BUFFER_TYPE: other`).

For UH-type user buffers, if the user does not know which of the numeric types it is, `scanping` can initially be run on a couple of files, setting `SHORT_FORM` to `no` and the three user buffer options to `none` just for the purpose of establishing the number of bytes recorded; the `USER_BUFFER_TYPE` is then this number multiplied by 10.[3] The `scanping.cnt` `USER_BUFFER_TYPE` should be set to this value and the `UB_DEFINITION` should be set to `ub_#.def`, where # is 10 times the number of user buffer bytes. A `ub_#.def` file for each of the UH user buffer types has already been copied to the `scan/` subdirectory by the `adcptree` script.

[2] An offset correction can be done post-loading if necessary; see the discussion on calibration and miscellaneous utilities. But correction for drift is currently available only with the `loadping` program.

[3] UH user buffer types `1281` and `1021` are later versions of types `1280` and `1020`, respectively. They contain the same information but in different order so as to comply with byte-alignment restrictions on Sun 4 architectures. They were never actually used during data acquisition, but only exist for databases that had the original `1280/1020` user buffers but have since been ported to sun4 platforms.

For the UH user buffer types `720`, `1320`, and `2240`, `scanping` automatically generates an extra column in the `OUTPUT_FILE` that gives the difference between PC and GPS fix times, along with the column of ADCP ensemble times. These two columns are sufficient for determining the offset and drift in the ADCP PC clock. The `UB_OUTPUT_FILE` is thus unnecessary and set to `none`.

For the older UH user buffer types `1020` and `1280`, this extra column is not available. Instead, a `UB_OUTPUT_FILE` has to be specified so `scanping` will generate a second file with a summary of the Transit fixes, and a last column that gives the PC minus Transit fix times.

For non-UH PC binary-type user buffers, it is still possible for scanping to decipher the user buffer if the user provides an appropriate `UB_DEFINITION` file that specifies the data types in the order that they appear in the user buffer. The file `ub_720.def` copied to the `scan/` directory by `adcptree` is a concise yet comprehensive example of how to create a `UB_DEFINITION` file. Details are also available in another CODAS document (see *The External Structure Definition File* section in `codas3/doc/codas3.doc`). The `USER_BUFFER_TYPE` will be specified as `other` and a `UB_DEFINITION` file must be specified so `scanping` will attempt to apply the definition file to decipher the user buffer and print out its contents. The user may find it useful to also learn how to use the `DEFINE_FORMAT` statements (also described in `codas3.doc`) within the `UB_DEFINITION` file in order to control the output format; otherwise, `scanping` will use a whole line for each field in the binary structure!

5.1.2 Running Scanping

After setting up the control file, the program can then be run by typing:

`scanping` [*scanping.cnt*]

The output log file, named after the `OUTPUT_FILE` specification in `scanping.cnt`, (Figure 2) lists the header and profile number, recorded profile time, the time interval between two consecutive profiles, and if applicable, the difference between the PC and satellite clocks. Below we discuss the things a user should check for when examining `scanping`'s output.

The number of headers in a file depends on how many times data collection was interrupted and restarted.[4] The number of profiles within each header depends on how long the data collection process is allowed to continue without interruption, up to some maximum per file (the actual number depends on how many variables and bins are being recorded). In general, therefore, a change in header number should alert the user to a possible change in instrument configuration or to a gap in data

[4]This is not the case for the demo files distributed with CODAS because the ping files used there were not directly written by the DAS but by way of a serial connection controlled by the UH user-exit program. For simplicity, the program sends the header block with each data block through the serial port, so there are as many headers as there are profiles.

```
DATA FILE NAME ../ping/pingdata.000

head  prof   date      time       day    interval   pc-fix time
                                        (min:sec)      (sec)
   1     1  93/04/09  00:02:32  98.001759   0: 0          1
   2     1  93/04/09  00:07:32  98.005231   5: 0          1
   3     1  93/04/09  00:12:32  98.008704   5: 0          1
   .
   .
   .
 294     1  93/04/09  23:57:32  98.998287   5: 0         -1

END OF FILE: ../ping/pingdata.000


DATA FILE NAME ../ping/pingdata.001

head  prof   date      time       day    interval   pc-fix time
                                        (min:sec)      (sec)
   1     1  93/04/10  00:02:33  99.001771   5: 1       99999
   2     1  93/04/10  00:07:33  99.005243   5: 0       99999
   3     1  93/04/10  00:12:33  99.008715   5: 0           0
   .
   .
   .
  79     1  93/04/10  06:39:51  99.277674   5: 0         -2
---- bottom velocity is ON
  80     1  93/04/10  06:46:02  99.281968   6:11         1
  81     1  93/04/10  06:51:00  99.285417   4:58        -1
  82     1  93/04/10  06:56:02  99.288912   5: 2        -1
   .
   .
   .
 242     1  93/04/10  20:16:00  99.844444   4:58        -1
---- bottom velocity is OFF
 243     1  93/04/10  20:22:12  99.848750   6:12         1
 244     1  93/04/10  20:27:11  99.852211   4:59         0
 245     1  93/04/10  20:32:11  99.855683   5: 0         0
   .
   .
   .
 286     1  93/04/10  23:57:12  99.998056   5: 0         1

END OF FILE: ../ping/pingdata.001
```

Figure 2: Sample `scanping` output file (abridged). The columns are header number, profile number, profile time in year/month/day hour:minute:second, profile time in decimal days (decimal day 0.5 is noon of 1 January), the difference in minute:second between the times of the current and preceding profiles, and the time difference in seconds between the PC clock and satellite clock. This last column is available only when using the UH user-exit program versions that write 1320-, 2240-, or 720-type user buffers.

recording. In the case where the interruption is due to toggling bottom tracking, `scanping` reports whether `bottom velocity is ON` or `OFF`. By scanning the file for these messages, the user can determine whether bottom track calibration should be attempted as part of processing.

The profile time should routinely be examined for a reasonable value. Something obvious like the year setting may be totally off!

The sixth column (time interval between successive profiles) gives the user a rough idea of the ensemble interval settings used, gaps in data recording, and other quirks. Normally, the column should be within a couple of seconds of the ensemble interval setting in the DAS. An occasional, slightly longer interval usually occurs as a consequence of brief interruptions, as when toggling bottom tracking. Longer intervals should be noted and explanations sought (missing file? ADCP PC lockup? disk full?) so recording gaps can be minimized for future cruises if possible. Noting them now also helps to distinguish recording gaps from gaps encountered later as a consequence of editing or missing navigation.

Careful examination can also alert the user to sections of the data that need not be loaded. For example, a section with very short ensemble lengths may indicate a period when the profiler was merely being tested, perhaps in port. Or a section at the beginning followed by a rather large gap may indicate data collected during port testing, and then a respite, followed by the actual start of the cruise.

Other quirks include a negative interval, which may indicate that the PC clock either reset itself or was manually reset. The user must determine the proper time correction to be applied during loading so ensembles are once again in ascending time order. Another quirk, as shown in the demo, is the very short 2-3 second ensemble, also due to PC clock resets. This will probably yield a very noisy profile, which should not be loaded or else will need to be edited out subsequently.

For UH user buffer types 1320, 2240 and 720, the last column indicates the difference between the PC clock and the satellite clock. This is used to establish what time corrections should be applied, if any, during loading. For user buffer type 1320, this column can be used to determine the offset and drift, and whether the PC clock was reset at any point. A Matlab script `clkrate.m` is located in the `scan/` directory precisely for this purpose and is discussed below. For user buffer types 2240 and 720, this column indicates whether the clock correction feature of the program has been activated and is working as expected. In the demo example, the feature is enabled to correct the PC clock within 2 seconds of the satellite time, and this last column confirms that it is working as expected (except where fixes were not available for brief periods, hence the bad value `99999`). If, for some reason, the automatic clock correction does not seem to be working,[5] then the user should proceed to determine

[5] Reasons we have encountered to date include exceeding the maximum correction limit because the PC clock is off by several hours to begin with, and a very short (10 seconds or less) ensemble setting—the user-exit program currently is set to perform the clock check/correction on the tenth ping. A maximum time correction limit is a precaution against unwarranted PC clock resets based

the necessary clock correction using the same method for the 1320-user buffer type (see `clkrate.m` discussion below).

Finally, there may be warnings in the `scanping` output file indicating sections of the file that do not conform to expected format, e.g.,

```
    1    55  93/03/26  14:52:05  84.619502   2: 3
 WARNING: bad data found in HDR 1 PRF 55
```

The exact causes may vary, including a section of the recording disk gone bad or problems with the ADCP PC itself. In general, both `scanping` and the loading program `loadping` are able to skip over the bad profile(s) and read the subsequent ones. The user merely has to make a note of which headers or profiles should be skipped during loading so `loadping` will not attempt to read bad profile(s) and crash with an error. This is discussed further in the section on loading.

If a `UB_OUTPUT_FILE` has been specified, then `scanping` will also output the contents of the user buffer. For UH buffer types 1280 and 1020, the `UB_OUTPUT_FILE` will contain a columnar summary of the Transit fixes grabbed toward the end of each ensemble. For brevity, we will not go into the details of this obsolete format. For UH buffer types 1320, 2240 and 720, specifying a `UB_OUTPUT_FILE` will result in creating an empty file by that name, since it is not necessary for checking PC clock performance (all the necessary information is already in the `OUTPUT_FILE`) or for extracting the information for later navigation calculations (this is done as a separate step). For other user buffer types, the `UB_OUTPUT_FILE` will print, for each ensemble, the profile time (both in decimal days and year/month/day hour:minute:second format), followed by the contents of the user buffer.

```
PROFILE TIME:  159.059815 ddays (91/06/09  01:26:08)
5819.10,W,,,319,M,010.4,N,000
$GPGGA,012246,2125.45,N,15813.52,W,1,5,001,00034,f,,
$GPRMC,012246,A,2125.45,N,15813.52,W,11.5,327.6,090691,11.0,E*69
$GPRMA,V,0000.00,S,00000.00,E,,,00.0,000.,11.,E*7
```

If the user buffer contains a satellite time in one of the messages, then the time of the fix can be compared with the profile time to determine PC clock accuracy. The user should know at what point during the ensemble the user buffer acquires its contents, so that proper adjustments can be made when comparing with the profile time that corresponds to the end of the ensemble. In the above example, the `$GPGGA` message contains the fix time in the second field, expressed as hhmmss (hh = 00 to 23 hours, mm = 00 to 59 minutes, and ss = 00 to 59 seconds). It indicates a discrepancy of 3 minutes and 22 seconds with the PC profile time. The user must be able to tell whether that fix time is toward the start or end of the 150-second ensemble, before

on spurious satellite times.

being able to conclude by how much the PC profile time needs to be adjusted, if at all.

5.1.3 Estimating PC Clock Offset and Drift

If a need for correcting the ensemble times has been determined, the user must then establish several parameters:

`start_header_number`: This is the header number at which to start applying the time correction. The time correction will be applied to this and all subsequent headers *within the file*, or until another set of time correction parameters are specified.

`correct_time`: This constant is the correct GMT time derived from the satellite clock at some reference point in time, such as the time of the first ensemble at the beginning of a cruise.

`PC_time`: This constant is the PC clock reading at time equal to `correct_time`.

`clock_rate`: This constant is the PC to satellite clockspeed ratio. A value of one means there is no drift in the PC clock, less (more) than one means the PC clock is faster (slower) than the satellite clock.

`recorded_time`: This variable is the PC clock time of each ensemble as seen in the output of scanping (Figure 2).

`corrected_time`: This variable is the final correct time of each ensemble.

The time correction is applied as:

$$corrected\_time = correct\_time + clock\_rate * (recorded\_time - PC\_time)$$

If a file containing the satellite fix times (or the recorded profile times) and the time difference (in seconds) between the PC and satellite clocks is available, then the process of estimating the above parameters can be automated using the Matlab script `clkrate.m`. For cases where fix times are not recorded with the ADCP ensembles, the user is referred to the subsequent section.

For UH buffer types 1280 and 1020, the `UB_OUTPUT_FILE` containing the columnar summary of Transit fixes can directly be used with this script, since the first and eighth columns of the file contain the fix times (in decimal days) and the time difference (in seconds).

For UH buffer types 1320, 2240 and 720, the `scanping OUTPUT_FILE` also contains the required information, but needs to be stripped of text and other extraneous lines before it can be loaded into the Matlab workspace. Also, the fix times need to be recalculated by adding the PC minus fix time back to the profile times. Unix users can take advantage of a csh script `cleanscn` to do this automatically:

`cleanscn` *scanping_output_file cleanscn_output_file*

The result will be a two-column file that can be loaded into Matlab by the `clkrate.m` script.

For other user buffer types that contain the necessary information, the user can write his or her own script or program to parse `scanping`'s `UB_OUTPUT_FILE` and derive the necessary columns. (The first column must be fix time in decimal days.)

Once the proper input file is available, the user can then apply the `clkrate.m` script under Matlab. All this actually does is fit a line to the scatter plot of PC minus fix time versus fix time (Figure 3). The x-intercept of the line would correspond to the time when both PC and satellite clocks coincided (PC minus fix time goes to zero). This directly yields values for two of the required parameters: `PC_time` and `correct_time` (as seen in Figure 4 as `PC clock reset`). The slope of the line would correspond to the reciprocal of the third required parameter `clock_rate`. However, in some cases, the `correct_time` (fix time when PC minus fix is equal to zero), can be substantially long ago. In this case, it would be preferable just to use the satellite fix time (`correct_time`) and PC clock time (`PC_time`) of the first good ensemble as the reference point in time.

The script takes care of eliminating outliers, including the bad values that occur when a fix is unavailable to match against the PC profile time.

The script also takes care to detect if the PC clock may have been reset over the period, in which case a single linear trend would be inappropriate. Instead, for each detected reset, a new linear estimate is generated. It should be reasonable to expect the slope (hence the PC clock rate) to vary little over the various estimates; we generally pick the estimate derived over the longest time span. The x-intercept estimates should roughly correspond to when the resets take place.[6]

With this background, the user should now be able to edit the relevant section of `clkrate.m`, as shown in Appendix B.3. The parameter `infile` specifies the name of the `Matlab`-loadable file containing fix times in the first column and the time difference in seconds between the PC clock and the fix time in another column. The `dtcolumn` specifies which column in the `infile` contains this time difference field. The `outfile` specifies the name of the output file used for logging `clkrate`'s results. The `yearbase` is the year to use when converting between decimal day and year/month/day hour:minute:second formats. The `abs_max_gap` is a threshold value in seconds (unsigned) used for detecting a PC clock reset. If the `dtcolumn` data jumps by a value greater than this threshold, the script concludes that a clock reset has occurred and yields a new linear fit over the data starting from that point up to the end or the next reset. The user may need to fiddle with this value depending

[6] Often, the PC clock is not even reset at the beginning of the cruise, and may have drifted significantly over a period of time, so finding the x-intercept may require a rather extended extrapolation. A bad initial setting (several hours off) may yield an x-intercept that indicates that the last time PC and satellite clocks were synchronized was several years back! Such estimates would also be more sensitive to slight differences in the slope.

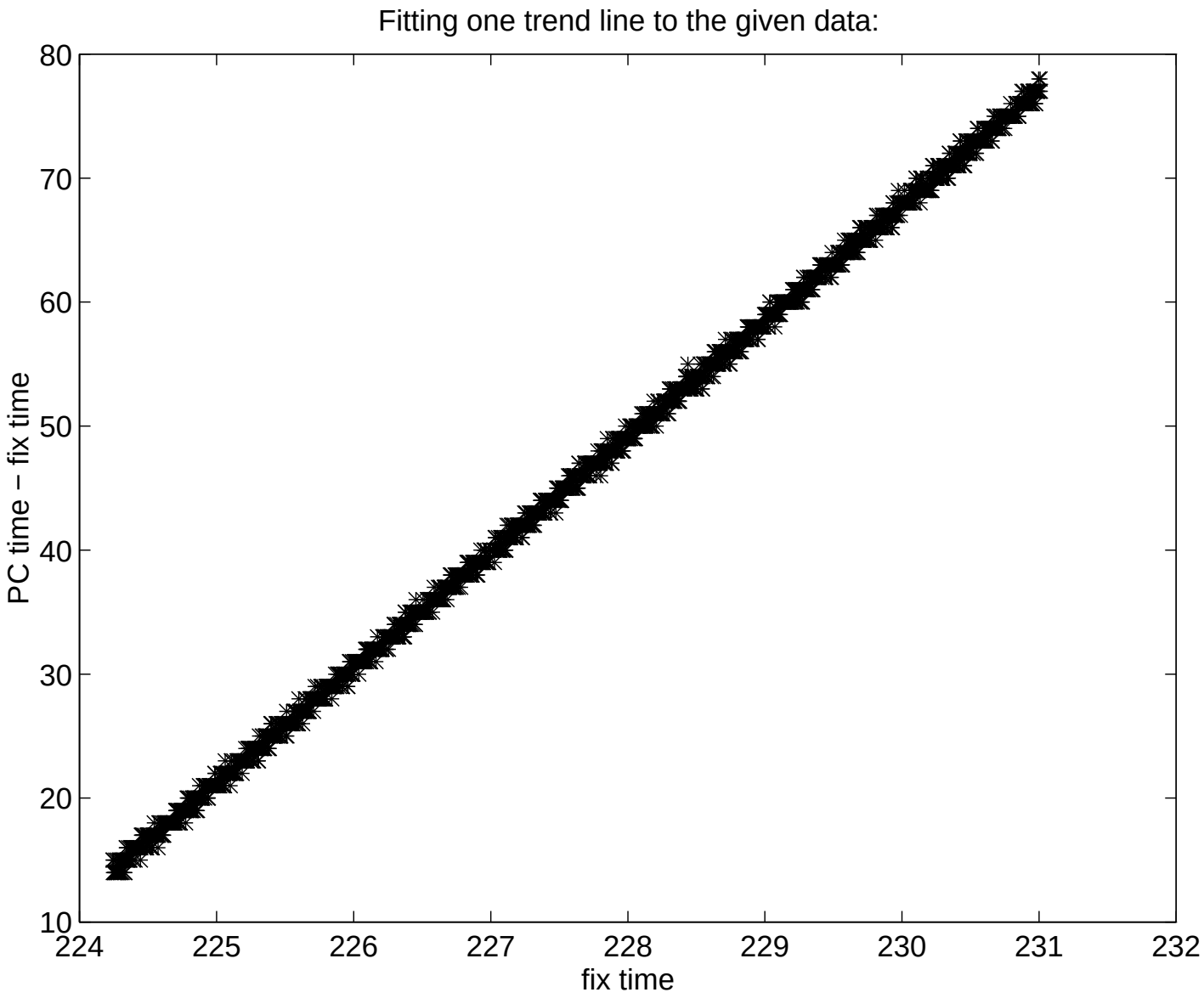


Figure 3: PC clock drift (ratio of PC to satellite clock speeds) as viewed using `clkrate.m` within Matlab.

```
abs_max_gap = 7  abs_max_bad = 1000
 Fitting one trend line to the given data:
 PC clock rate = 0.99989214915
 PC clock reset at 92/08/10  16:48:00
```

Figure 4: Output of `clkrate` . The `PC clock rate` corresponds to the the clock drift rate and the `PC clock reset time` is when the PC and satellite clocks are equal.

on the underlying data. Finally, the `abs_max_bad` is a threshold value in seconds (unsigned) used for weeding out outliers in the `dtcolumn` data. This may need to be increased if the PC clock is set too differently from the satellite clock, or is allowed to drift over a very long cruise. (Bad or missing satellite fix times are indicated by a value of 99999 in the `dtcolumn`).

After editing `clkrate.m`, the user starts up Matlab and simply types:

```
>> clkrate
```

The script loads the input file and first attempts to fit a single line to the entire dataset (less any detected outliers). It then prints a message regarding how many PC clock resets it has detected. The user needs only to press the carriage return in order to see the various fits. If he or she thinks that the `clkrate abs_max_gap` and/or `abs_max_bad` parameters need some adjustment, then the script can be interrupted by pressing `control-C`. In the end, the `outfile` will contain a convenient summary of `clkrate`'s results. Together with `scanping`'s `OUTPUT_FILE`, this can guide the user in filling out all the time correction parameters needed for the loading stage.

Examples of setting parameters in the `loadping.cnt` control file for correcting clock problems are shown in Section 5.2.

5.1.4 Other Means for Correcting Ensemble Times

Many installations rely on navigation recorded separately from the ADCP ensembles, or record positions without fix times along with the ADCP. This makes it difficult for the user to determine the clock correction.

In the past, we have tried two ways around this problem. One way depends on a ship log where the ADCP technician onboard has to record the time(s) at which the ADCP PC clock has been reset, always noting both the pre- and post-reset readings on the PC clock. The problem with this method is obvious.

In another situation, the availability of bottom track velocity throughout most of the cruise made it possible to establish a rough estimate of the time correction parameters by lining up the ship motion as tracked by the ADCP versus that tracked by the navigation. Of course, this method depends a lot on having substantial variations in the ship movement. This method can also be used with the water track velocities referenced to some appropriate layer. The program `timslip` was originally written with this purpose in mind (discussed under Chapter 7). Since the program can be used only after the data have been loaded into the CODAS database, the user may need to reload if a time correction consisting of both an offset and a drift is required. Otherwise, the utility program `chtime` (discussed under Chapter 11) can be used to apply an offset correction to the ADCP ensemble times post-loading.

5.2 Loading ping files into a CODAS database

Loading the ping files into the database begins with creating/editing the producer definition file. An example is shown in Appendix C.1. Details on how to set up this file is provided in an online document `codas3/doc/codas3.doc`. It is probably easiest to start with one of the example `*.def` files copied by the `adcptree` script into the `adcpdb/` directory of the ADCP tree.[7] For vessel-mounted ADCP datasets, the only parameters that the user may need to modify are:

- `PRODUCER_ID`: This is a string *cciippnnnn* that encodes the producer identification using the two-digit NODC codes for country (cc), institute (ii), platform (pp). The last 4 digits are the instrument code, which may be used to distinguish one profiler from another on the same platform.

- Data Definitions: This section should be reviewed to include only those variables that have been recorded by the DAS or will later be added to the database. Everything else should be tagged `UNUSED`. Of the included variables, a distinction is made between block variables (`BLOCK_VAR`, that is, the data may vary only from block to block and are therefore stored only once per block, e.g., the instrument configuration, depth) and profile variables (`PROFILE_VAR`, that is, the data may vary from profile to profile, e.g., current velocities, navigation). The numeric IDs have been standardized for the more common oceanographic variables. The user may expropriate unused numbers for variables that are not currently on the list.[8] The value type column is set to the smallest possible storage type that can accommodate the range of values for a given variable, after applying the offset and scale factors. For example, velocities are scaled to mm/s to enable storage as short (2-byte) integers (range -32767 to 32767) instead of floats (4 bytes). In general, the value types for numeric arrays have been judiciously selected. Value types for other variables like the user buffer are more implementation- dependent. Currently the sample producer definition file is set up for a UH PC binary-type user buffer. Installations with ASCII-type user buffers may need the value type reset to `TEXT`.

- Structure Definitions: This section should be reviewed such that all variables of type `STRUCT` that will get loaded into the database will have an accompanying structure definition that can be used to decipher the structure contents. Care must be taken to arrange structures to comply with byte-alignment restrictions on target host machine(s). So far, the rule that a field must fall on a boundary address divisible by its size suits machines that CODAS has been ported to. This means that `DOUBLE` types should fall on boundaries that are a multiple

[7] The various `*.def` files copied by `adcptree` differ only with regard to the user buffer definition.

[8] Note that using variables other than those already tagged as either `PROFILE_VAR` or `BLOCK_VAR` will require modifications to the loading program `loadping`.

of 8. The easiest way to comply with this rule is to order the fields starting from those with the largest value types (`DOUBLE`s) to the smallest (`CHAR`, etc.). Otherwise, the user may need to apply "padding" or unused fields in order to force the next field to the proper alignment (as we have done for some of our more mixed structures). Details on setting up the structure definition are also available from `codas3/doc/codas3.doc`. The user should also pay attention to deleting unused structure definitions from the producer definition file as these take up some disk space, trivially slows down access, and may cause unnecessary confusion.

Once a suitable producer definition file has been set up, the user can proceed to edit the `loadping.cnt` file (Appendix A.13):

The parameter `DATABASE_NAME` specifies the name of the database to be loaded, with pathname if needed (by convention, we store the database in the `adcpdb/` directory of the ADCP tree). We normally stick with 5 characters or less for the database name, so that after a 3-digit sequential numbering has been appended to form each block file name, the file names still comply with PC DOS restrictions. If the database already exists, then it will be appended to. [9]

The parameter `DEFINITION_FILE` specifies the producer definition file that the user prepared during the preceding step. The `OUTPUT_FILE` specifies a log file for the loading process. By convention, we use extension `.lod`.

The next four parameters determine how a given database will be split up into block files. Since instrument configuration is a block variable in an ADCP-VM dataset, any change in one of its fields will always cause a new block file to be started. The user can then introduce other conditions for starting a new block file by setting up these other four parameters, but should always be mindful of the tradeoff in storage required for each block file's overhead.

The `MAX_BLOCK_PROFILES` parameter can be used to set a maximum number of profiles loaded into each block file. It is useful for controlling the size of each block file.

The `NEW_BLOCK_AT_FILE` can be set to `yes` if the user wants to be able to map block files directly to the ping data files. This may not be such a good idea when ping files are recorded to 360-Kb floppies, for example. Each floppy would typically have one large and one small ping file in that case, and this would carry over into the block file sizes.

The `NEW_BLOCK_AT_HEADER` is set to `yes` if the user wants to start a new block file whenever a new header occurs. A new header coincides with a recording interruption, usually due to change in instrument configuration, or when a new ping file is started.

[9] A fairly common problem arises from neglecting this fact: loading the same set of ping files twice into the same database will cause duplication and lead to confusion when searching the database by time. If a database has to be reloaded in its entirety, then all the block (`*.blk`) files should first be deleted.

So this parameter overlaps with all the other conditions and is usually left disabled (set to `no`) unless peculiar problems are encountered.

Finally, the parameter `NEW_BLOCK_TIME_GAP(min)` specifies a recording gap threshold in minutes, which, when exceeded will cause a new block file to be started. The appropriate setting depends on the ensemble length, but it is a good idea to enable this feature, as there could be all sorts of reasons for recording gaps that may be determined later. It may be that a ping file was missing; it would be less complicated to add it to the database when found at a later time. It may also turn out that the database should be split up into two cruises where the gap occurs. Reconstruction into two separate databases will be much easier if the data are already broken into separate files at the gap.

The keyword `PINGDATA_FILES` heralds the list of ping data files to be loaded. Again, it is not necessary to load everything at once, since subsequent runs will append to the database. It is recommended that the ping data files be listed in sequential order.

The list of ping data files minimally consists of one or more ping data file names, each followed by the keyword `end`. In between each file name and its matching `end`, the user may specify one or more options:

The option `skip_header_range` can be used to specify a range of header numbers within that ping file to be skipped during loading. This is useful when the `scanping` run has revealed these to be unloadable (`WARNING`), or when the user has concluded that these are test data collected in port or data that belong to some other cruise.

The option `skip_profile_range` is useful for the same reasons as `skip_header_range` but gives finer control for situations where only selected profiles within a header need to be skipped.

Finally, the `time_correction` specifies how the profile times are going to be corrected as they are loaded into the database. The subparameters have already been discussed in a previous section.

These three options (along with their component subparameters, where applicable) can be repeated as many times as needed. For example, if the ADCP PC clock was reset in the middle of a ping file, then two time corrections starting at different header numbers may become necessary. The first time correction specified will affect all profiles under the headers from the `start_header_number` given to the header right before the subsequent time correction `start_header_number` specification.

An example of skipping bad headers and profiles and of correcting the simple case of clock offsets (no drift) is shown with both the output file from `scanping` and the control file for `loadping` in Figures 5 and 6, respectively. Within the first pingfile, the first profile of the second header is skipped because of a long interval. At that time, the clock offset changes. Within the control file, the constant value `correct_time:` is calculated by hand by subtracting the `pc-fix time` from the `PC clock time` beginning with the time offset of the first ensemble which has length as set during configuration (5-minutes in this example). In other words, this gives the satellite fix

```
DATA FILE NAME ../ping/94020413.000
head  prof   date      time       day    interval   pc-fix time
                        (min:sec)      (sec)
  1     2  94/04/11  20:55:57  100.872187   5: 1          -2
  1     3  94/04/11  21:00:57  100.875660   5: 0          -1
  .
  .
  1    28  94/04/11  23:05:57  100.962465   5: 0       99999
  2     1  94/04/12  01:02:22  101.043310  116:25          -3
  2     2  94/04/12  01:07:21  101.046771   4:59          -4
  2     3  94/04/12  01:12:21  101.050243   5: 0          -4
  .
  .
  .
END OF FILE: ../ping/94020413.000
DATA FILE NAME ../ping/94020414.001
head  prof   date      time       day    interval   pc-fix time
  1     1  94/04/13  04:27:21  102.185660   5: 0          -4
  1     2  94/04/13  04:32:21  102.189132   5: 0          -4
  .
  .
END OF FILE: ../ping/94020414.000
DATA FILE NAME ../ping/94020420.003
head  prof   date      time       day    interval   pc-fix time
  1     1  94/04/15  15:57:20  104.664815   4:59          -5
  1     2  94/04/15  16:02:21  104.668299   5: 1          -5
  .
  .
  1   124  94/04/16  02:12:21  105.091910   5: 1          -6
  1   125  94/04/16  02:17:21  105.095382   5: 0          -5
  2     1  94/04/16  02:25:18  105.100903   7:57         305
  3     1  94/04/17  19:13:43  106.801192  2448:25         -6
  3     2  94/04/17  19:18:43  106.804664   5: 0          -6
  3     3  94/04/17  19:23:43  106.808137   5: 0          -5
  .
  .
  3   122  94/04/18  05:18:43  107.221331   5: 0          -5
  3   123  94/04/18  05:23:43  107.224803   5: 0          -6
  4     1  94/04/19  08:39:57  108.361076  1636:14         -61
  4     2  94/04/19  08:44:57  108.364549   5: 0         -61
```

Figure 5: `scanping` output file (abridged) which accompanies the example on setting parameters in the loadping control file (Figure 6). Note the PC clock resets that change the PC minus fix time difference. These resets must be corrected for during `loadping`.

```
 PINGDATA_FILES:
../ping/94020413.000
     skip_profile_range:
       hdr=  2
       prof= 1 to 1
     time_correction:
          start_header_number:  2
          correct_time:         94/04/12-01:07:25
          PC_time:              94/04/12-01:07:21
          clock_rate:           1
end
../ping/94020414.001
     time_correction:
          start_header_number:  1
          correct_time:         94/04/13-04:27:25
          PC_time:              94/04/13-04:27:21
          clock_rate:           1
end
../ping/94020420.003
     time_correction:
          start_header_number:  1
          correct_time:         94/04/15-15:57:25
          PC_time:              94/04/15-15:57:20
          clock_rate:           1
     skip_header_range:      2 to 2
     skip_profile_range:
          hdr=    3
          prof=   1 to 1
     skip_profile_range:
          hdr=    4
          prof=   1 to 1
     time_correction:
          start_header_number:  4
          correct_time:         94/04/19-08:45:58
          PC_time:              94/04/19-08:44:57
          clock_rate:           1
end
```

Figure 6: `scanping` Load control file (abridged): simple case of how to skip bad headers and profiles and how to set time corrections for a PC clock without drift but with resets. See Figure 5 for corresponding `scanping` output file.

time of the first good ensemble after the clock reset. For the same ensemble, the constant value `PC_time:` is simply the `PC clock time`. The time offset, which in this case -4 seconds, continues into the next pingfile. Thus, the subparameters in the control file for the next pingfile must also be set for the `time_correction:` parameter. The `correct_time` and `PC_time:` are derived from the first 5-minute ensemble. Finally, in the following pingfile, time corrections for offsets are applied twice, as well as controls to skip bad headers and profiles.

Another example of setting the `loadping.cnt` control file is provided for the case of a drifting clock. Figure 7 shows the beginning of the scan file (output of `scanping`). By scrolling through this file one notices a change in the PC minus FIX time difference. By using the Matlab utility, `clkrate`, as described in Section 5.1.3, one determines the drift of the PC relative to the satellite clock, which is shown numerically in Figure 4 and graphically in Figure 3. This information is then incorporated into the `loadping.cnt` control file as shown in Figure 8. Note the same time is used for the `correct_time` and for `PC_time` which is the time when the PC clock read the same time as the satellite clock. For each ping file while the drift remained steady, these parameters remained the same.

Once the control file is set up, the program is run by typing:

`loadping` [*loadping.cnt*]

The program generates two types of output: the database itself, and the log file. Before proceeding to examining the database, the user should examine the log file, especially in the case where a time correction has been applied. An excerpt from a sample log file is given in Figure 9.

The user should confirm that only those profiles/headers specified to be skipped are not loaded. When a time correction has been specified, the log file also reports on how many minutes and seconds have been added to each profile time. This should roughly correspond to the negative of the PC minus fix time column from the `scanping OUTPUT_FILE`, if available.

5.3 Examining the Database

The program `showdb` is used for directly examining the contents of the database. There are many instances when it is useful: right after loading to make sure that the variables display reasonable values; immediately before and after updating the database in any way, to confirm that the expected result has been accomplished; and checking on the processing status of a given database.

To start `showdb`, the user types:

`showdb` [[*path*]*dbname*]

```
DATA FILE NAME ../ping/mw28d205.png

head  prof   date      time       day    interval   pc-fix time
                        (min:sec)      (sec)
  1     1  92/08/12  05:50:02  224.243079   0: 0          15
  1     2  92/08/12  05:55:01  224.246539   4:59          14
  1     3  92/08/12  06:00:01  224.250012   5: 0          15
  1     4  92/08/12  06:05:00  224.253472   4:59          14
  1     5  92/08/12  06:10:00  224.256944   5: 0          14
  1     6  92/08/12  06:15:00  224.260417   5: 0          14
  1     7  92/08/12  06:20:01  224.263900   5: 1          15
  .          .
  .          .
  .          .
  1    58  92/08/12  10:35:01  224.440984   5: 1          16
  1    59  92/08/12  10:40:00  224.444444   4:59          15
  1    60  92/08/12  10:45:01  224.447928   5: 1          16
  1    61  92/08/12  10:50:01  224.451400   5: 0          17
  1    62  92/08/12  10:55:01  224.454873   5: 0          17
  .          .
  .          .
  .          .

  1   179  92/08/12  20:40:01  224.861123   5: 0          19
  1   180  92/08/12  20:45:00  224.864583   4:59          20
  1   181  92/08/12  20:50:00  224.868056   5: 0          20
  1   182  92/08/12  20:55:00  224.871528   5: 0          20
  1   183  92/08/12  21:00:01  224.875012   5: 1          21
  1   184  92/08/12  21:05:00  224.878472   4:59          20
  1   185  92/08/12  21:10:00  224.881944   5: 0          20
```

Figure 7: `scanping` output file (abridged) which accompanies the example on setting parameters in the loadping control file (Figure 8). Note the PC clock is drifting.

```
 DATABASE_NAME:            ../adcpdb/a9212
 DEFINITION_FILE:          ../adcpdb/adcp1320.def
 OUTPUT_FILE:              a9212.lod
 MAX_BLOCK_PROFILES:       400
 NEW_BLOCK_AT_FILE?        yes
 NEW_BLOCK_AT_HEADER?      no
 NEW_BLOCK_TIME_GAP(min):  30

 PINGDATA_FILES:
../ping/mw28d205.png
    time_correction:
     start_header_number:  1
     correct_time:         92/08/10-16:48:00
     PC_time:              92/08/10-16:48:00
     clock_rate:           0.99989214915
    end
../ping/mw28e115.png
    time_correction:
     start_header_number:  1
     correct_time:         92/08/10-16:48:00
     PC_time:              92/08/10-16:48:00
     clock_rate:           0.99989214915
    end

  .
  .
  .
```

Figure 8: `scanping` Load control file (abridged): simple case for correcting a PC clock drift but with no clock resets. See Figure 7 for corresponding `scanping` output file, and Figure 4 for drift rate and `correct_time`: as determined by the Matlab utility `clkrate`.

```
Data file name: ../ping/pingdata.000
% OPTIONS:
% =======
% skip_header_range:
% skip_header_range:
% skip_header_range:
% =======

Header 1
     1   93/04/09 00:02:32
Header 2
     1   93/04/09 00:07:32
Header 3
     1   93/04/09 00:12:32
Header 4
     1   93/04/09 00:17:32
Header 5
     1   93/04/09 00:22:32
Header 6
     1   93/04/09 00:27:29
Header 7    not loaded.
Header 8    not loaded.
Header 9
     1   93/04/09 00:32:32
     .
     .
     .
Header 294
     1   93/04/09 23:57:32
end of file ../ping/pingdata.000

Data file name: ../ping/pingdata.001
% OPTIONS:
% =======
% =======

Header 1
     1   93/04/10 00:02:33
Header 2
     1   93/04/10 00:07:33
Header 3
     1   93/04/10 00:12:33
     .
     .
     .
Header 286
     1   93/04/10 23:57:12
end of file ../ping/pingdata.001

DATABASE CLOSED SUCCESSFULLY
```

Figure 9: Sample log file produced by `loadping`. Three periods indicate where lines were omitted for this figure.

```
% showdb ../adcpdb/ademo

 DBOPEN DATABASE NAME = ../adcpdb/ademo

 BLK:     0   OPEN: 1    PRF:     0   OPEN: 1    IER:      0

 1 - Show BLK DIR HDR          10 - SRCH TIME
 2 - Show BLK DIR              11 - SRCH BLK/PRF
 3 - Show BLK DIR ENTRY        12 - MOVE
 4 - Show BLK HDR              13 - GET TIME
 5 - Show PRF DIR              14 - GET POS
 6 - Show PRF DIR ENTRY        15 - GET DEPTH RANGE
 7 - Show DATA LIST            16 - GET DATA
 8 - Show DATA DIR             17 - SET SCREEN SIZE
 9 - Show STRUCT DEFN          99 - QUIT


 ENTER CHOICE ===>
```

Figure 10: Opening menu of `showdb` program.

where *dbname* is the name under which the database was loaded (preceded by the path to the location of the block files if not in the current directory). The *dbname* should coincide directly with the `DATABASE_NAME` specification in the `loadping.cnt` file, and not include 'dir' or the any of the sequential numbering appended to this during the loading stage.

If `showdb` succeeds in finding and opening the database,[10] it will present a menu and prompt the user for a selection (10).

Items 10 to 17 of the menu are useful for directly examining the database variables. Items 10 to 12 are positioning commands for accessing different profiles. Items 13 through 16 retrieve data for the profile at which the database pointer is currently positioned. Item 16 in particular is useful for retrieving different variables. Item 17 is used for changing the number of lines displayed on the screen (default is 20 lines) so variables displayed under item 16 can be viewed better.

When choosing item 16, the user is prompted for either a variable name or numeric ID. Item 7 is useful for displaying the list of available variables. The user must

[10] Common causes of failure are misspecification of path or database name; not having read permission on the files; or trying to read files generated by a different host environment (e.g., a database loaded on a PC is currently not readable in a Sun environment unless a conversion step has been done—this is discussed in Chapter 11).

correctly type in the variable name (case does not matter here) or its numeric ID.

After loading, it is a good idea to make a point of examining any new user-defined structures to make sure that their fields display correct/reasonable values without crashing the program. A weird result or program crash would be indicative of a bad structure definition (perhaps a bad value type for a field, or byte-alignment incompatibilities) and the database may need to be reloaded to fix this. Item 9 is a convenient way of checking what structure definitions `showdb` is applying to the structure-type data.

Item 1 is useful for a number of reasons. It displays the block directory header, that is, information that spans the entire database (Figure 11).

```
 BLOCK DIRECTORY HEADER

db version: 0300
dataset id: ADCP-VM
time:  1993/04/09 00:02:00 --- to --- 1993/04/10 23:58:00
longitude: - 91 47' 59.28" --- to --- - 89 11'  3.03"
latitude:    11 40' 50.40" --- to ---   13 32' 10.07"
depth: 21 --- to --- 493
directory type -----> 0
entry nbytes -------> 80
number of entries --> 4
next_seq_number ----> 5
block file template > ademo%03u.blk
data_mask ----------> 0000 0000  0000 0000  0000 0011  1000 0001
data_proc_mask -----> 0000 0000  0011 0000  0000 0011  0101 0010

 ---Press enter to continue.---
```

Figure 11: Database block directory header displayed by `showdb`.

The user should take note of the `time` range, as this will need to be pasted into various control files used in subsequent processing steps. If the navigation step has not yet been completed (as with a freshly loaded database), the the `longitude` and `latitude` ranges will not be filled in. Otherwise, these indicate spatial coverage of the database. The `depth` range is filled in during loading, and is calculated from the following fields of the `CONFIGURATION_1` structure: `num_bins`, `tr_depth`, `bin_length`, `pulse_length`, and `blank_length`. Of these, the `tr_depth` may have been inappropriately set in the DAS, so checking the depth range at this point can indicate if a depth correction may be necessary.

The `number of entries` specifies how many block files comprise the database. The user should note that while the block files are sequentially numbered starting

from `001`, the logical block and profile numbering starts from 0. The sequential numbering of the database block files are in the order in which they were loaded into the database and never change once loaded, while logical block numers are ordered by the time of the earliest profile in each block file and may change when entire blocks are deleted from the database, or when significant time offsets are applied to selected blocks by running the `chtime` utility program.

The `data_mask` marks the presence/absence (1/0) of the variables with numeric IDs 0 through 31 (rightmost to leftmost bit). These IDs cover the fundamental oceanographic quantities like current velocities, salinity, temperature, etc. (Check any of the sample producer definition files in the `adcpdb/` directory for exactly which variables have ID 0 through 31.)

Finally, the `data_proc_mask` marks the processing status of a given database. Each bit position has a specified meaning as defined in the file `codas3/adcp/include/dpmask.h` (Appendix C.2). A freshly loaded database without any time correction applied will show all bits turned off. As the database is updated by each processing step, the appropriate programs turn the applicable bit to indicate that the step has been completed and applied to the database. For example, when any type of time correction is applied to the database (including during the `loadping` step), then the rightmost bit gets switched on. Currently, not all 32 bits have meanings assigned to them.

Items 2 through 4 on the showdb menu yields similar information as the block directory header, but specific to a particular block file rather than the entire database. Items 5 and 6 give time, position and depth range information for each profile in the current block. Item 8 is mainly used for administrative purposes, as these give the relative offset and number of bytes occupied by each profile variable.

5.4 Preparing the Cruise Track

It is useful to generate a cruise track plot before proceeding so the user has an idea of what problems to anticipate during processing, and also as a second check for gaps, this time in the navigation recording. We normally do a longitude versus latitude plot, as well as longitude versus time and latitude versus time plots (Figure 12). For example, a cruise track that shows the ship in shallow water indicates that editing for bottom interference will be necessary. Outliers in the cruise track plot can immediately show where the navigation will need to be edited. Gaps will need to be attributed to either ADCP recording gaps, or just gaps in the navigation. A cruise track plot may help in accounting for some gaps by showing that the ship was actually at a standstill (perhaps in port) over that entire period. The cruise track plot may also show that the user has failed to exclude large sections of in-port data, which probably ought to be deleted from the database (either by reloading or running the utility program `delblk`, described in Chapter 11). Finally, a cruise track plot clues the user on what current features to expect.

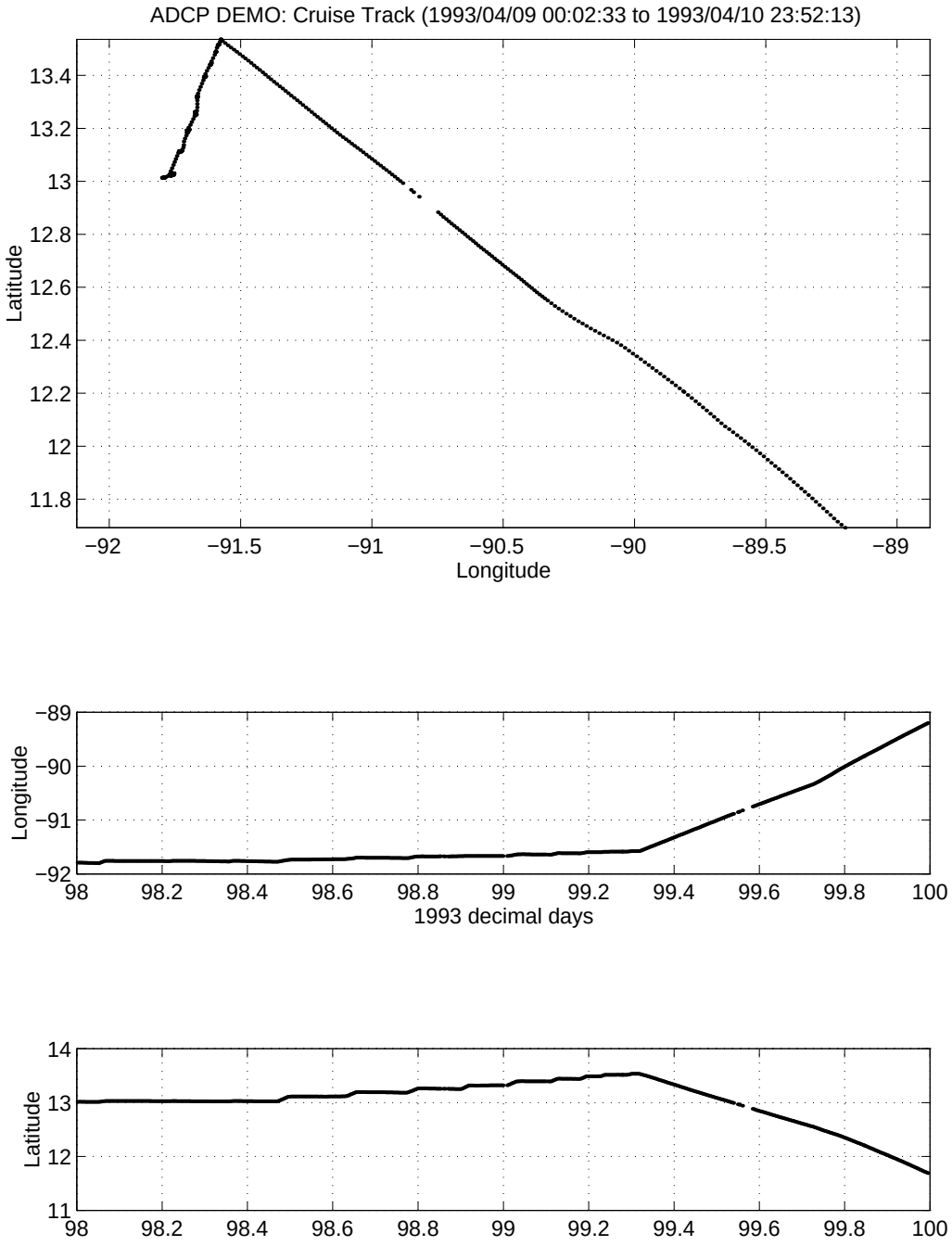


Figure 12: Example of a cruise track created by MATLAB.

Figure 12 was generated by a Matlab routine `cruistrk.m` located in the `nav/` directory of the ADCP tree. Its input is an ASCII file with the first 3 columns as time (in decimal days), longitude and latitude (both in decimal degrees).

The first step in plotting the cruise track, therefore, is obtaining this input file. If a user-exit program was employed to record the navigation in either the navigation structure or the user buffer, then this information needs to be extracted from the database, if it hasn't been done already (the `UB_OUTPUT_FILE` extracted by `scanping`). The first section of Chapter 8 discusses in detail how this input file can be obtained from various sources.

Once this file is ready, the user needs to edit the relevant section of the Matlab script `cruistrk.m` (Appendix B.4). The `fix_file` specifies the name of the input ASCII file. The `cruise_id` is just a label used for the plot title. The `year_base` is used for converting decimal days to yy/mm/dd hh:mm:ss format; this also goes into the plot title. Finally, the `plot_symbol` selects one among the available Matlab plotting symbols and line types. Using a symbol is recommended over using a line type so gaps are immediately evident.

After editing, the user starts up Matlab and simply types:

```
>> cruistrk
```

The script plots to the screen as well as to the PostScript file `cruistrk.ps`.

6 Editing

The premises behind the UH editing system are:

- Because of the volume of ADCP data, automated scanning for possible problems is essential.
- Although the interactive CODAS/MATLAB editing system greatly facilitates the task of identifying suspicious data, a purely independent, automated, editing system is not suitable. It is still mandatory for a trained operator to examine staggered profile plots of various parameters in order to distinguish real from erroneous signals and to decide exactly how the editing should proceed.
- Because editing is sometimes done iteratively, and one might sometimes want to undo a step, the eventual changes to the database should be kept to a minimum and should be reversible where possible.
- The editing process should be self-documenting. An array of flags, corresponding to the arrays of relative velocity profiles, are allocated in the database. Then for each bin containing suspicious relative velocities, a flag is set. The original relative velocities remain intact.

The editing of the ADCP database consists of the following checks and actions:

- Verify whether the speed of sound used during data aquisition was appropriate. Apply the best possible speed of sound parameter as a function of time, if applicable.
- Set the bottom flags to reject all data not in the water column. The bottom is sensed by the acoustic beams as the ship enters and leaves port and any time during the cruise when the water depth is shallow (within the range of the beams). The reflection of the acoustic beams off the solid bottom surface causes a very sharp signal in the amplitude as measured by the Acoustic Gain Control. However, the high amplitude signals must be distinguished between whether the strong reflection is from the bottom or from a dense "scattering layer".
- Set flags for bins contaminated by interference from the physical objects within the path of one or a few beams, such as the winch wire during hydrocasts. These usually occur only in several consecutive bins among the top dozen.
- With a poor installation or a poor choice of setup parameters, it may also be necessary to reject data from the top depth bin(s).
- Set flags for the occasional random glitches caused by instrument failures, configuration errors, or aquisition system disruptions.
- Eliminate any bad satellite fix values (this step is done separately).

The basic methodology behind the CODAS editing system is as follows.

- Update speed of sound used during the Doppler signal processing, if necessary. This is accomplished by verifying whether the data acquisition system was configured to use a constant speed of sound throughout the cruise or whether the speed of sound was determined from the temperature at the transducer, in which case this temperature series must be examined. If it is determined that a better estimate of the speed of sound could be obtained, then a CODAS program `fix_temp` is employed to update the database.
- Create statistics on various parameters such as the relative U, V, and W velocities, error velocity, and signal return amplitude. These mean, standard deviation, second derivative, and variance statistics are used for setting thresholds of the quality control tests.
- Employ the CODAS/MATLAB editing tool to view staggered profile plots of the various parameters. The quality control tests are automatically performed and suspicious bins or profiles that exceed the operator-defined thresholds are

marked on the plots. For a given range of profiles (typically the operator views 50 or 100 at a time), plots can be made for a variety of parameters for cross-referencing whether the signal is real or erroneous. For signals that are designated as bad, the CODAS/MATLAB editing tool automatically lists these suspicious profiles/bins in separate files as a function of the type of error. The operator has the liberty to jump around in time if need be and can refer to the actual bin-by-bin values for various parameters in the CODAS database with the `showdb` function.

- After the staggered plots have been examined and the lists of bad profiles/bins and bottom encounters are finalized, the operator updates the CODAS database. As previously mentioned, only flags are set; the original data are left intact.

- At this time or during the Calibration or Navigation steps, one can correct bad satellite fixes, if necessary.

6.1 Verification of the Speed of Sound

For non-obvious reasons, the constant of proportionality between the Doppler shift and the water speed involves the speed of sound *at the transducer* only. During data acquisition, the sound speed could have been either left constant throughout the cruise or could have been determined for each ensemble as a function of the temperature and salinity at the transducer. The temperature is usually determined by a thermistor at the transducer while a constant salinity is configured into the DAS. Sound speed changes by about 0.1% for each psu change in salinity; thus, if the salinity used in the DAS is within 0.5 psu of the in situ salinity, the error will be very small. During the course of a cruise the salinity constant may only have to be reset in the DAS once or only a few times. On the other hand, sound speed changes by 0.3% per °C at 0°C, and by 0.13% per °C at 30°C. Thus, temperature is significant in the sound speed calculation. It would be useful for the processor if notes on the health of the thermistor are available within the ship log of the ADCP DAS operator.

In either case, first we examine the performance of the thermistor. If the instrument was set to calculate sound speed from the thermistor reading, then we just want to verify that the thermistor was indeed behaving properly. If the instrument was set to use a constant sound speed and the cruise track is such that the constant sound speed value is inappropriate, then we need to establish a better estimate of sound speed from measurements of salinity and temperature.

From within the `edit/` subdirectory, one must edit the control file `lst_temp.cnt` to set the appropriate database name (and path), the year base, and the time span of the cruise. The program is executed by typing:

`lst_temp [ lst_temp.cnt ]`

The output file from this program can be plotted in MATLAB. One first edits the MATLAB script `plottemp.m` to set filenames. For Non-UNIX users, the output file from `lst_temp` must be edited to remove headers and comments. For UNIX users, this is done automatically by `plottemp.m`. Enter MATLAB and type `plottemp`. MATLAB produces a plot on the screen and a post script file of the temperature trace versus time.

At this time it would be advantageous to pull in independent measurements of temperature at the transducer depth, such as from CTD or XBT data. This temperature versus time data can be fit into a MATLAB-loadable file and superimposed on the thermistor trace. This plot will give the operator more information for making the judgement on the quality of the thermistor data. If one has confidence that the CTD and XBT data are reliable, then the superimposed plot shows if a constant offset exists over the course of the cruise, if the thermistor has a drift, or if the thermistor data are completely unreliable. If a correction is not needed, then jump to Section 6.2; otherwise, the correctional procedure is explained below.

The operator should prepare either a constant offset temperature or a file of ADCP profile times versus the offset between the thermistor and CTD/XBT data. By matching up the ADCP profile times to the nearest CTD/XBT points, a constant offset temperature can be determined as the mean difference between the CTD/XBT and the thermistor data. Or the offset variation with time can be established as the point-by-point difference between the two temperature samples.

Likewise for salinity, independent estimates of the salinity can be obtained from a CTD or other instrument and compared with the value(s) configured in the DAS. A similar offset or point-by-point variation can be established. In both cases (temperature and salinity), interpolation can be used to obtain an offset for each ADCP profile time.

The sound speed correction is implemented with the CODAS program `fix_temp`. From the `edit/` subdirectory, modify the file `fix_temp.cnt` (Appendix A.10). Note that the `true_temperature=` and `true_salinity=` parameters can be either a constant or a file of ADCP profile times and values (examples of such files shown in this Appendix). To run the program, enter:

`fix_temp [ fix_temp.cnt ]`

One can verify the updates to the database using `showdb` and looking at the `ANCILLARY_1`: variable (37) via the `GET DATA` (16) option.

6.2 The CODAS/MATLAB Editing System

The bulk of the effort in editing centers around the use of the CODAS/MATLAB editing system. This interactive system greatly facilitates the editing process by producing staggered profile plots of the ADCP variables with marks set on suspicious bins. The user can choose how many profiles to place on a given plot and can produce a

cruise track for this range of profiles to determine if the ship was on station or moving. The user analyzes the marked bins and profiles to determine if the suspicious signal is an error or a realistic feature. Before moving on to the next range of profiles, the user either accepts or rejects the marked bins and profiles as bad. The accepted flags are placed in separate files determined by the type of error (bottom interference, bad bins, or bad profiles–see Figure 13). As the editing continues, the operator can jump around in time if need be and add or delete messages to or from the files of error lists. Once the viewing of the profiles is completed, the files of error lists are used for setting flags in the ADCP database.

The principal objective of the editing stage is to identify when and at what depth the acoustic beams reflect off the bottom, to set the top bin at which a profile contains good data, and to flag bins contaminated by interference from physical objects, such as the winch wire during a CTD cast, or by other random occurrences such as instrument failure or DAS glitches. The main editing steps are:

- **Set thresholds for the quality control tests.** These can be left at the default values, assigned by means of an educated guess from an experienced processor, or quantitatively determined based on statistics of the data through application of CODAS programs. The thresholds are loosened or tightened during the course of editing as needed through judgement of the operator.

- **Employ the CODAS/MATLAB editing system.** View various parameters (relative velocities as raw or as statistical products, return signal amplitude, error velocity, percent good) on staggered profile plots which mark suspicious signals that deviate beyond the thresholds. Reject or accept the flagged bins and profiles. This usually requires exploratory work such as cross-referencing among the various parameters, inspecting the ship log, or referring to the actual values in the database with `showdb`, to oceanographic atlases, or to bathymetric charts. It is preferable to have an operator with at least an introductory level knowledge of physical oceanography. The accepted flagged bins are automatically placed in output files based on the type of error. The operator steps through the cruise by viewing typically 50 or 100 profiles at a time although due to the iterative nature of editing, one can jump around as necessary and either add or delete suspicious bins/profiles to/from the output files containing the error lists.

- **Update the database.** Using the error list files as input to various CODAS programs, the ADCP database is updated by setting the maximum depth (bottom), setting the top good bin, and flagging suspicious bins for each of the profiles. One can use `showdb` after this step is finalized for viewing the arrays of flags corresponding to the profiles.

```
Key to columns:
B: Block #
P: Profile #
N: # of marked bins
B1(n): Bin #
XB: bin of maximum amplitude,
XD: depth of maximum amplitude
A: administrative (used internally)

==> badbin.asc <==
 date      time      B   P   N  B1  B2  B3  B4 .  .  Bn
92/12/04  20:10:05   1   7   1  34
92/12/05  23:20:02   2  29   4   2   3   4   5
92/12/05  23:25:02   2  30   4   2   4   5   6
92/12/06  00:40:01   2  45   3   4   5   6
92/12/06  00:45:01   2  46   4   3   4   5   6
92/12/06  00:50:01   2  47   4   3   4   5   6
92/12/06  00:55:01   2  48   4   3   4   5   6
92/12/07  00:24:35   3  17   1   2
92/12/07  00:39:33   3  20   1   2

==> badprf.asc <==
 A   B   P   date      time
-1   0   0 92/12/04  18:48:33
-1   0   1 92/12/04  18:53:29
-1   0   2 92/12/04  18:58:28
-1   0   3 92/12/04  19:03:29
-1   0   4 92/12/04  19:08:27
-1   0   5 92/12/04  19:13:27
-1  16 130 92/12/19  07:50:20
-1  16 131 92/12/19  07:55:21
-1  16 132 92/12/19  08:00:20
-1  17  17 92/12/20  00:30:21

==> bottom.asc <==
 A   B  P    date      time   A      A      XB  XD
 0   3 244 92/12/07  19:19:13 A    32767     6  61
 0   3 245 92/12/07  19:24:14 A    32767     7  69
 0   3 246 92/12/07  19:29:14 A    32767     7  69
 0   3 247 92/12/07  19:34:14 A    32767     7  69
 0   3 248 92/12/07  19:39:14 A    32767     8  77
 0   3 249 92/12/07  19:44:14 A    32767     7  69
 0   3 250 92/12/07  19:49:13 A    32767     7  69
 0   3 251 92/12/07  19:54:13 A    32767     7  69
 0   3 251 92/12/07  19:54:13 A    32767     7  69
 0   3 252 92/12/07  19:59:13 A    32767     7  69
```

Figure 13: Sample output files (abridged) from CODAS editing system. The names of the three files are denoted between the arrows. These files are input for the CODAS programs that set flags in the ADCP database.

6.2.1 Setting Thresholds

The quasi-automated editing system is based on comparing variations about the mean of the ADCP variables (raw and derived quantities) bin-by-bin and profile-by-profile to a set of pre-defined thresholds. Each parameter for each bin/profile that exceeds the threshold is flagged. Then the operator judges whether the data were flagged due to natural "noise" or due to an error. This is the basic idea behind the editing system.

Several raw ADCP ensemble variables and derived statistics are analyzed by the editing system:

- **Amplitude.** The return strength as measured by the Acoustic Gain Control exhibits a sharp maximum when the depth is greater than about 50 meters yet within the range of the acoustic beams. One has to be careful to distinguish between bottom reflections and scattering layers, such as dense areas of fish.

- **Variance of the vertical velocity (W).** This variance is calculated for an entire profile. When the water depth is less than about 50 m the variance is large and thus is a good indicator of shallow depths. Moreover, this statistic is also useful for catching "wild" profiles caused by glitches in the DAS.

- **Error velocity.** This value is derived from the fact that two of the four acoustic beams measure vertical velocity; the error velocity is simply the difference. It is mainly used to detect interference of physical objects in the path of one or a few of the acoustic beams, such as during a hydrocast, and shows up on the plots as a peak covering several adjacent bins within roughly the top dozen bins.

- **Second derivatives of U, V, and W.** These statistics supplement the detection of winch wire (or the like) interference. If the second derivative is high for W and for at least one of the second derivatives of U or V, then it is a good indication of physical intrusion in the path of a few of the beams. This statistic also detects "spikes" in the relative velocities caused be strong shear among adjacent bins. These spikes could be real noise or could be due to glitches in the DAS or instrument.

The thresholds are set in a MATLAB script, `setup.m` (Appendix B.7). The user has the option of using the default values as provided with the CODAS, setting threshold values based on knowledge gained from previous cruises, or quantitatively determining the thresholds based on the ADCP data. It is recommended to do the latter as a reference point regardless. In many cases the thresholds are loosened or tightened during the editing stage as the operator views many profiles and gets a feeling for how well the thresholds perform.

The percent-good threshold sets the range of the ADCP profiles that are used for data processing and analysis. Bins with percent-good less than this threshold are not

edited other than for amplitude. i.e., bottom interference, and are not considered for most of the following processing and analysis steps such as referencing of absolute velocities, plotting of velocity contours, etc. Obviously, the threshold chosen for the editing should be maintained (or at least be the lower limit) in the following steps. The percent-good threshold for the demo cruise, and for most of the recent cruises, was set to the relatively low 30% because the transducer was configured to already edit out bad data prior to recording into the ping data files. In situations where this is not done, the percent-good threshold may be as high as 80% (Bahr et al., 1989).

For the amplitude threshold (`AMP_THRESHOLD`) a good starting point is a value of 10. If it appears that too many scattering layers are being detected, then the value could be raised to 20 or more. In most cases, it is safest to leave the value at 10, view the flagged values on the staggered profiles of the amplitude parameter, and to reject the flags. For the error velocity check, a relatively low value of 50 is suggested as a starting point, and can be raised depending on the particular cruise. The thresholds for the second derivative statistics and W variance are more cruise dependent; thus it is best to calculate a starting point based on the ADCP data. Regarding the W variance, we found that the top one or two bins always caused unexplained fluctuation in the vertical velocity component, and so the bin at which to start the W variance calculation is usually set to 3.

A CODAS program, `profstat`, is available for calculating cruise statistics which can be used for determining the thresholds, `profstat`. Two control files have been created for feeding information to program `profstat`: `profst00.cnt` and `profst00.cnt` (Appendix A.19). The former is used to obtain a general idea of the degree of variability in the vertical velocity while the latter is used to obtain second difference statistics on U, V, and W velocity components.

In the first control file, `profst00.cnt`, as seen in the first example in Appendix A.19, a `step_size` of N (i.e., considering only every Nth profile) may be selected to cover a wide data time range in a relatively short running time of the program. The `ndepth` parameter selects the maximum bin range considered by the program. If several data time ranges were specified in the time range list, the option `combined` will produce one output, while `separate` results in statistics for each individual time range. The variable list here includes only W. For the selected variable, a title ("W component"), a reference layer, the order of vertical differencing, and a scale is specified. Referencing subtracts the mean over the specified range from the profile. The output consists of a text file (`ademodf0.prs`) and a MATLAB-loadable file (`ademodf0.mat`) that is used for the threshold calculation.

The second control file, `profst02.cnt`, as seen in the second example of Appendix A.19, was used to obtain statistics on the second vertical difference of U, V, and W. It is very similar to the one described above, with the exception that the ASCII output for W is in the form of a histogram. The `style= n` refers to a normalized histogram. The corresponding output files are similar to the above.

To run these programs, type:

`profstat` [*profst00.cnt*]

and

`profstat` [*profst02.cnt*]

A MATLAB script, `threshld.m` is used to calculate the thresholds from the output of `profstat`. Update this MATLAB script to reflect the correct filenames. Then enter MATLAB and type:

`threshld`

The script will write threshold values for the various statistics to the screen. The user should make a note of the values for future reference. At this point the user can update the CODAS/Editing system `setup.m` script with these thresholds if this deems appropriate.

6.2.2 Controls within SETUP.M

Prior to initiating the CODAS/MATLAB system, one configures the `setup.m` script (Appendix B.7) with a text editor. The parameters are separated by function: retrieval, editing, and plotting. Each parameter has a default value already assigned as part of the demo and a description of what it is and how to use it.

Within the `retrieval parameters`, one first sets the file name and path to the ADCP database. The `DEFAULT_NPRFS` field is usually adjusted during the course of editing. If the ship is steadily steaming in deep water and the instrumentation is working adequately, then this field can be raised to 100 or more. Conversely, if the ship is frequently on station or is in an area of abrupt bottom depth changes in the range of the acoustic beams, then this attribute is more appropriately set to 25. The other fields are self-explanatory.

The `editing parameters` have mostly been explained in Section 6.2.1 concerning the thresholds. If the `EDIT_MODE` attribute is set to "1", then all marked bins/profiles that the operator accepts are placed in the appropriate output files of error lists. However, as a beginning step of editing, the user may want to leave the `EDIT_MODE` = 0 (the default) and view several consecutive staggered profile plots of various parameters to see if the thresholds appear adequate. For instance, if the error velocity flags are being set for many signals that are clearly natural noise and not interference by something like a winch wire during a CTD cast, then one edits `setup.m` and assigns a higher value to `EV_THRESHOLD`. Then one could begin the viewing of profiles again from the start of the cruise with the `EDIT_MODE` parameter set to "1", which allows the operator the option to accept or reject each flag. Accepted flags are placed in output files as a function of error type.

Within the `editing parameters`, special attention is given to identifying the maximum water depth, ie. the bottom. In addition to the `AMP_THRESHOLD` attribute

which was discussed in Section 6.2.1, two other parameters are utilized. The `BOTMPAS3` and `LAST_85` flags can be set if you intend to use these programs later after the bottom flagging or just want to see their effects. The botmpas3 program is used to "spread" the bottom around. That is, where profiles have been flagged for bottom interference at particular bins, the botmpas3 program looks at three such profiles in sequence, and assigns the shallowest bin to the middle profile. In this way, gaps of 1 or 2 profiles in the bottom flagging eventually get caught, without having to manually add them to the `bottom.asc` file. The `last_85` program reduces the last good bin by another 15%, to remove further contamination from side lobe effects (RDI, 1989).

Within the `plotting parameters` only the last three attributes are of importance. The `PLOTBAD` field is normally left at "1" (default) so the suspicious bins and profiles are marked on the plots. If one desires to keep several windows open while editing, which is very useful for cross-referencing, then the `PLOTNAV` and `PLOTUV` fields can be left at "1" (default). If your computer appears to be too slow to handle three windows at the same time or is configured only to plot one window at a time, then these last two parameters should be set to "0".

6.2.3 Interactive Controls

At last the user is ready to begin the CODAS/MATLAB interactive editing system. To initiate the system, enter MATLAB and type

```
setup
```

The `setup` feeds the default values to the CODAS editing system. It must be entered each time the CODAS/MATLAB session is activated. It must also be used each time parameters within `setup.m` are modified. Note that is not necessary to actually leave the CODAS/MATLAB editing system in order to update default settings of the `setup.m`; one can type the parameter and setting within MATLAB. For instance, to change the `DEFAULT_NPRFS` parameter, enter `DEFAULT_NPRFS=100` (or what ever desired value is wanted).

The interactive session is driven by a host of commands as shown in Figures 14 and 15. These commands are grouped by activity such as moving around within the database, plotting of desired parameters, and setting/unsetting flags.

The `movement` commands allow one to move around within the database. The `getp(`*blk,prf*`)` command must be entered as a starting point each time the `setup` command has been applied. For instance, `getp(0,0)` retrieves `DEFAULT_NPRFS` profiles from the database, starting from block 0, profile 0. It is also used when one wants to jump by a large number of profiles within the database. The `l` and `r` commands clear the present screen(s) of plot(s) and loads in the next batch of profiles immediately to the left or right, respectively, into the editing system. It is important to note that if the `EDIT_MODE=1` (list flags to output files) is activated, one must type `list` to place all desired flags in the output files before moving on to the next batch. As a safety

catch, the system will prompt you to save or not save the flags from the particular batch of profiles that are about to be cleared when the `l` and `r` commands are applied BUT NOT IF YOU ARE EXITING THE SYSTEM (in which case, use the `list` command prior to exiting). It does not hurt to `list` a given workspace more than once (e.g., you decided to throw in another bin for flagging, after doing a `list`). The main side effect is duplicate entries in the `*.asc` files, making them larger than necessary. To undo flagging after a `list` has been issued, you will have to edit the *.asc files manually.

Now the operator is ready to view plots of the various parameters (Figure 14 within `ADCP VARIABLES FOR PLOTTING`) that are automatically marked with flags. For instance, typing `e` plots the error velocity as shown in Figure 16. The Y-axis is bin number, negated. The X-axis is the variable with the selected offset applied to stagger the profiles. Various flags may appear on the plots (Figure 14, `FLAGGING SYMBOLS`) depending on which parameter is plotted. When the variables are loaded into the workspace, the command window displays the results of applying the editing criteria (Figure 17). These numbers help the operator judge whether to adjust the default threshold settings in `setup.m` and also aid in determining if the flagged value is real or erroneous.

Several special plotting features are available. For instance, one can highlight a particular profile or range of bins within a range of profiles with the `k` command. For examining fine detail, one can zoom in on a group of profiles with the `zi` command. This replots only the selected range of profiles as chosen by clicking the mouse. Upon examination, one can restore the plot to full-scale with the `zo` command.

The operator has control over which flags are accepted and which are rejected (Figure 15). These commands are straightforward and do not need more discussion.

The `pb` command is useful for toggling between on and off as explained in Figure 15. The `off` status (`pb=0`) only plots data that pass the access criteria (percent good, `last_good_bin` after application of by botmpas3 and `last_85`). Thus, it depicts the profiles as subsequential extractions are made after the editing stage is completed. Normally during the editing stage however, the `on` status (`pb=1`) is maintained while the operator decides whether to accept or reject the flags.

6.2.4 What to Look For

It has been our experience that it is best to walk through the database and examine the profile plots for all varieties of errors, rather than going through once just for bottom reflections, etc. This method works out best due to the nature of the examinations, which require continual cross-references.

The major sources of error that the operator wants to edit have already been discussed: bottom reflections, wire/CTD interference, and glitches. In this section, these ideas are elaborated upon and examples are given.

- **Bottom interference** Elimination of bottom interference is the easiest editing

CODAS/Matlab Editing System Commands

MOVEMENT

getp(*blk,prf*)	Retrieves DEFAULT_NPF profiles from ADCP database starting block *blk*, profile *prf*
l	Grabs set of DEFAULT_NPF profiles to the left
r	Same yet to the right

ADCP VARIABLES FOR PLOTTING

plotnav	Plot cruise track if lat/lon stored in NAVIGATION variable of ADCP database
a	For amplitude plots
p	Percent good
e	Error velocity
w	Vertical velocity (w) component of relative current
u	U horizontal component of velocity
v	V horizontal component of velocity
uv	For both U and V components in the same plot window

SPECIAL PLOTTING FUNCTIONS

k(*blk,prf*)	Key on block *blk*, profile *prf* by marking bins with '+'
k	Use mouse to click profile to key
k(*n*)	Move key +/- *n* profiles
k([ ])	Remove marked keys
zi	Zoom in using mouse to select area
zo	Zoom out to restore plot to full-scale (all profiles, all bins)

FLAG SYMBOLS

o (*yellow*)	Bins that fail the error velocity check
o (*magenta*)	Bins that fail the 2nd derivative test
* (*cyan*)	Bins that fail the amplitude check
* (*magenta*)	Profiles that fail the W variance check
+ (*yellow*)	Bins/profiles that the user marks as bad
x (*white*)	Bins/profiles that have already been edited out of the database; the edit commands do not affect these
+ (*white*)	Used as a "key"; not a flag due to suspicious error

Figure 14:

CODAS/Matlab Editing System Commands

UNFLAG BINS/PROFILES

ok	Unmark a region of bins with 'o' flags (2nd derivative and error velocity) using mouse to click upper-leftmost and lower-rightmost corners of plot area containing the undesired flags
okprf	Unmark entire profile(s) flagged bad from W Variance by clicking mouse on the profile(s)
sl	Unset a region of bins with '*' flag(s) (amplitude--ie. unflag a "scattering layer") by clicking mouse (as with "ok" command above)

FLAG ADDITIONAL BINS/PROFILES

zap	Add bad bin(s) by clicking mouse within the plot on the suspicious bin(s) (puts yellow '+' on flagged bin(s))
rzap	Mark a region of rejected bins using mouse to box area
bad(*b*, [*blk prf*], *npf*)	Mark a region of bad bins where *b* is a vector of bin numbers to flag, [*blk prf*] indicates the profile to start marking from, and *npf* is number of profiles to flag the same bins from
badprf	Add complete profile(s) to flagged subset by clicking the mouse on each (marked with yellow '+')
bottom	Mark each bin by clicking mouse where you think the bottom "maxes out"

MISCELLANEOUS CONTROL FUNCTIONS

setup	Initiate/re-initiate default settings in script 'setup.m'; this is the first command used in the system and can be re-applied as desired
pb	Toggle PLOTBAD variable (see setup.m); pb=1 means all data plotted with flags mark which allows you to see if flags are set correctly pb=0 means flagged data and data that do not meet the access criteria (percent good/last good bin) disappear from the plot but the marks remain which is useful to see if the current flagging is satisfactory or needs to be supplemented
undo	Undo previous command (ok, zap, bottom, etc.)
list	Records all flagged bins on current plot to output files (*.asc) after you are satisfied with the editing

Figure 15:

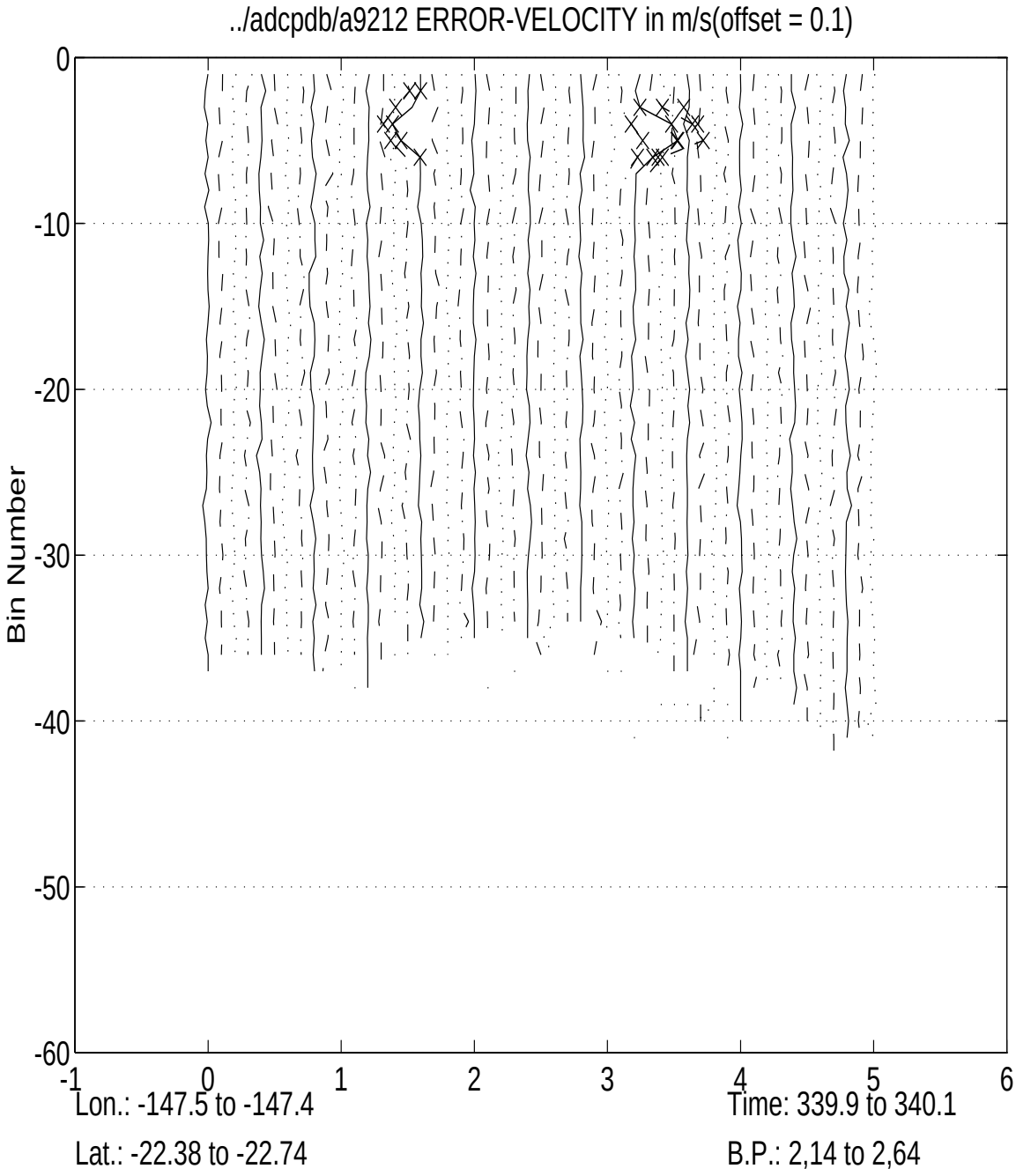


Figure 16: Error velocity plot depicting interference signals from a winch wire during a hydrocast as highlighted with the marks (x) which denote that these have been flagged in the database. This plot was made after the database was updated with the editing flags.

```
=================> ERROR VELOCITY <=================
Blk_Prf_Bin_EV =

 0     0     0     0     0     0     0     0     0     0     0     0
 5     6    12    12    36    44    46    47    50    51    52    57
 7     9     7    25     8     4     4     4     3     2     3     3
-59    54    60    59   -53    56    66    51    57    53    50    53

rows 1 to 4 respectively:  block number(s), profile number(s),
  bin number(s), and value for bins that fail the error velocity check.

=================> AMPLITUDE <=================
Blk_Prf_Bin_Z_Gap =

 0     0     0     0     0     0     0     0     0     0     0     0
 0     1    10    11    12    13    14    15    17    30    31    32
21    20    29    29    29    30    29    29    29    29    29    29
181   173   245   245   245   253   245   245   245   245   245   245
 0     0     8     0     0     0     0     0     1    12     0     0

rows 1 to 5 respectively:  block number(s), profile number(s),
  bin number(s), depth value at which the amplitude check failed, and
  gap indicator, the number of "unflagged" profiles between the previous
  and current column entries.

=================> W VARIANCE <=================
Blk_Prf_WVar =

 0     0     0
142   143   195
596   589   528

rows 1 to 3 respectively:  block number(s), profile number(s), and
  variance of flagged profile.

=================> 2ND DERIVATIVE TEST <=================
Blk_Prf_Bin_D2U_V_W =

 0     0     0     0     0     0     0     0     0     0     0     0
111   114   132   133   136   137   138   139   194   194   194   195
 6     7     7     7     2     2     2     2     5     6     7     4
 11   -91  -109  -133   -61    -4    -5   -20   286  -119  -211   178
-119     5    54    41    85   151   121   103   135  -140  -115    89
-43   -44   -49   -49   -53   -60   -53   -47   -50    60    62   -56

rows 1 to 6 respectively:  block number(s), profile number(s),
  bin number(s), and 2nd derivatives of U, V, and W respectively,
  for bins at which the 2nd derivative of W and the 2nd derivative
  of either U or V fail the editing check.
```

Figure 17: Command window display of results of applying editing criteria to various parameters.

task. The bottom is usually quite visible in a number of variables, and in addition it is normally known when the ship was in shallow water and thus when the ADCP might have encountered this problem.

Bottom interference is usually identified as a subsurface maximum in amplitude due to the strong backscatter from the solid ground (and typically noisy U and V-components). Corresponding to the ship's course over a changing bottom topography, the depth location of this maximum changes in time, typically increasing (decreasing) in strength as the bottom shoals (deepens) (Figure 18).

Bottom interference has to be distinguished from scattering layers in the water column caused by plankton or fish. While these "false bottoms" also show vertical migration due to the movements of the animals, the associated amplitude maximum often decreases in strength as it shoals when the critters disperse or the ship moves away from their largest concentrations (figure 19). Because RDI claims that scattering layers have no effect on measurements, bins marked as such are not edited out. However, plots of the U- and V- components should be examined, as scattering layers do affect velocity on rare occassions. It is advisable to print out any unusual plots and make a note of the bins where the scattering layer affects the velocity components, even if these bins are not edited out. At least a comment can be placed in the cruise summary documentation to alert users of the anomalous signal.

Another possible source of confusion is a very quickly changing bottom topography, i.e., large depth changes within one profile. The amplitude maximum would be "smeared out" and might be hard to detect. Increase in the vertical variance below the depth of the actual bottom is helpful in identifying such a situation.

Another editing tool to use is the command display window echoes within the CODAS/MATLAB session. As shown in Figure 17, (AMPLITUDE), the last row is an indication of any gaps in the flagging, i.e., the number of "unflagged" profiles between the previous and current column entries. Since it is normal for the bottom interference to contaminate a continuous sequence of profiles, it is important to detect where that may not be the case. Sometimes, the water becomes so shallow, so that there is really no good data left, but the amplitude signal will not exhibit the behavior that the flagging algorithm is searching for (Figure 20). In other cases, the "kink" in the amplitude is barely perceptible or has a peculiar shape that it escapes the automatic flagging. In both these cases, the operator may need to manually specify the bin at which those profiles' amplitude "max out".

- **Intrusion of object(s) in the path of the beam(s)**

  The error velocity flags often indicate interference by an object drifting into the view of the ADCP, such as winch wire during hydrocasts or deployment of

moorings. The example in Figure 16 shows a peak in the error velocity signal over several consecutive profiles while the ship was on station performing a hydrocast. Anomalous behavior is usually seen in vertical velocity and percent good plots (not shown). As previously mentioned, if the second derivative is high for W and for at least one of the second derivatives of U or V, then it is a good indication of physical intrusion in the path of a few of the beams. If the velocity components are not affected, the cause is likely to be a scattering layer. Please note that the above example, picked for illustrative purposes, is certainly on the more clear end of a range that reaches all the way to barely detectable problems.

- **Random glitches and problems in the upper bins**

  Occasional random glitches caused by instrument failures, configuration errors, or acquisition system disruptions are sometimes caught by the second derivative and W variance flags. It is important to distinguish between real shear and erroneous shear. Without notes from the ship log prepared by the DAS operator, it may not be possible to uncover the source. Flagging the database is left to the judgement of the editing operator.

  If turbulence is exceptionally high due to rough seas, then the first good bin may have to be adjusted. Usually this problem is taken care of by the percent good criteria during acquisition of the database.

6.2.5 Apply Editing Results to the Database

The final task of editing is applying the results to the ADCP database. At this stage, we assume that you have gone through the entire editing session and have the *.asc files (Section 6.2). The first program to run is

`badbin` *path to database* `badbin.asc`

where *path to the database* could look like `../adcpdb/ademo`, for example. This program simply sets the `PROFILE_FLAGS` variable in the database to indicate that particular bins have been tagged as bad. The original velocity values remain intact. During later data access with CODAS programs `adcpsect` or `profstat`, the user can specify whether to consider these tags during access by using the `FLAG_MASK` option. By default, this option is set to `ALL_BITS`, meaning only data for bins at which `PROFILE_FLAGS` are zero will be used. Use showdb, option 16 (`GET DATA`), variable ID 34 (`PROFILE_FLAGS`), to see the effects of running badbin on the database.

If there are entire profiles that need to get thrown out you need to run:

`dbupdate` *path to database* `badprf.asc`

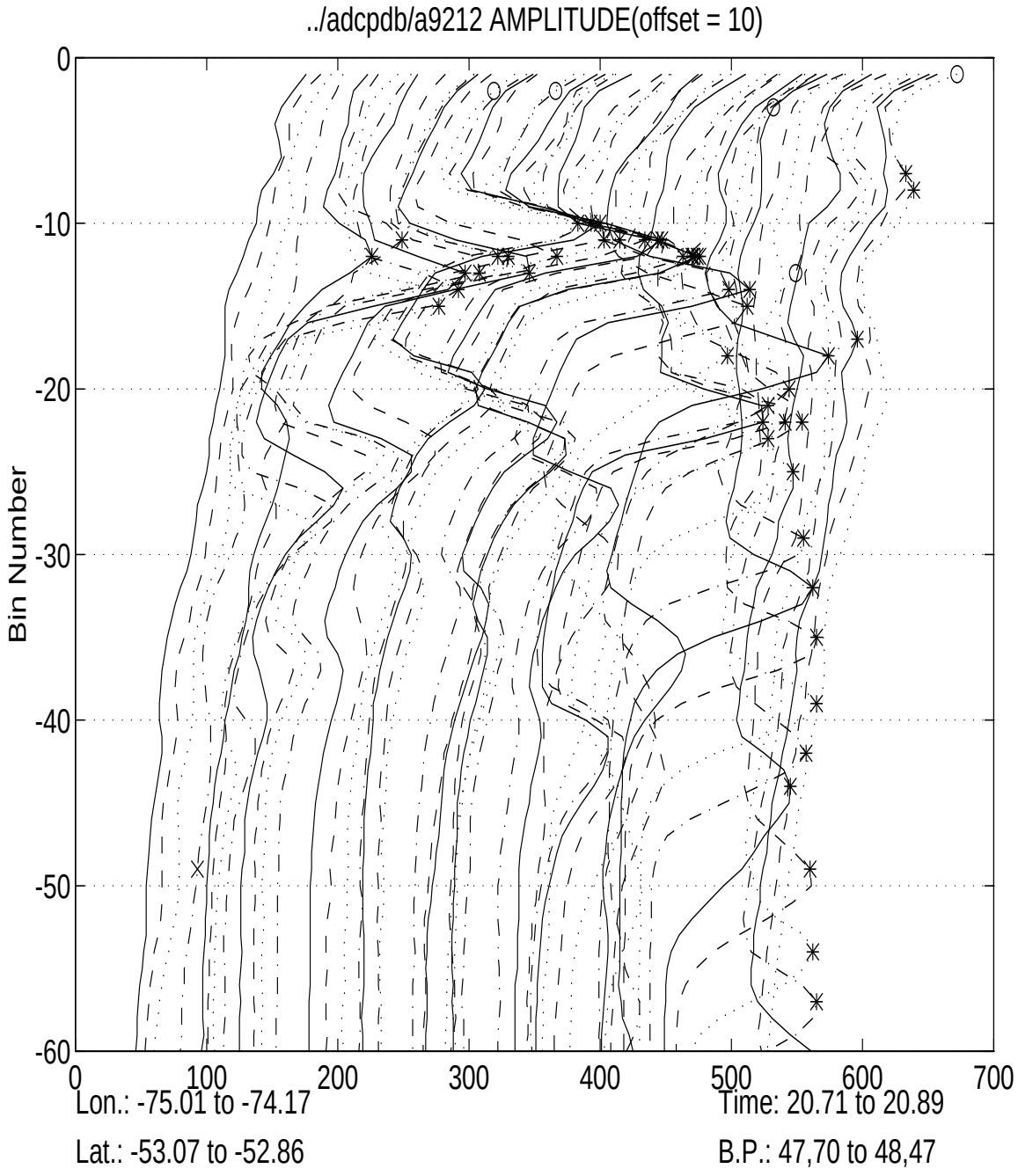


Figure 18: Bottom reflection (*) depicted by sharp maximum in the Amplitude signal. Note how the signal strength remains high at the maximum amplitude as the bottom deepens.

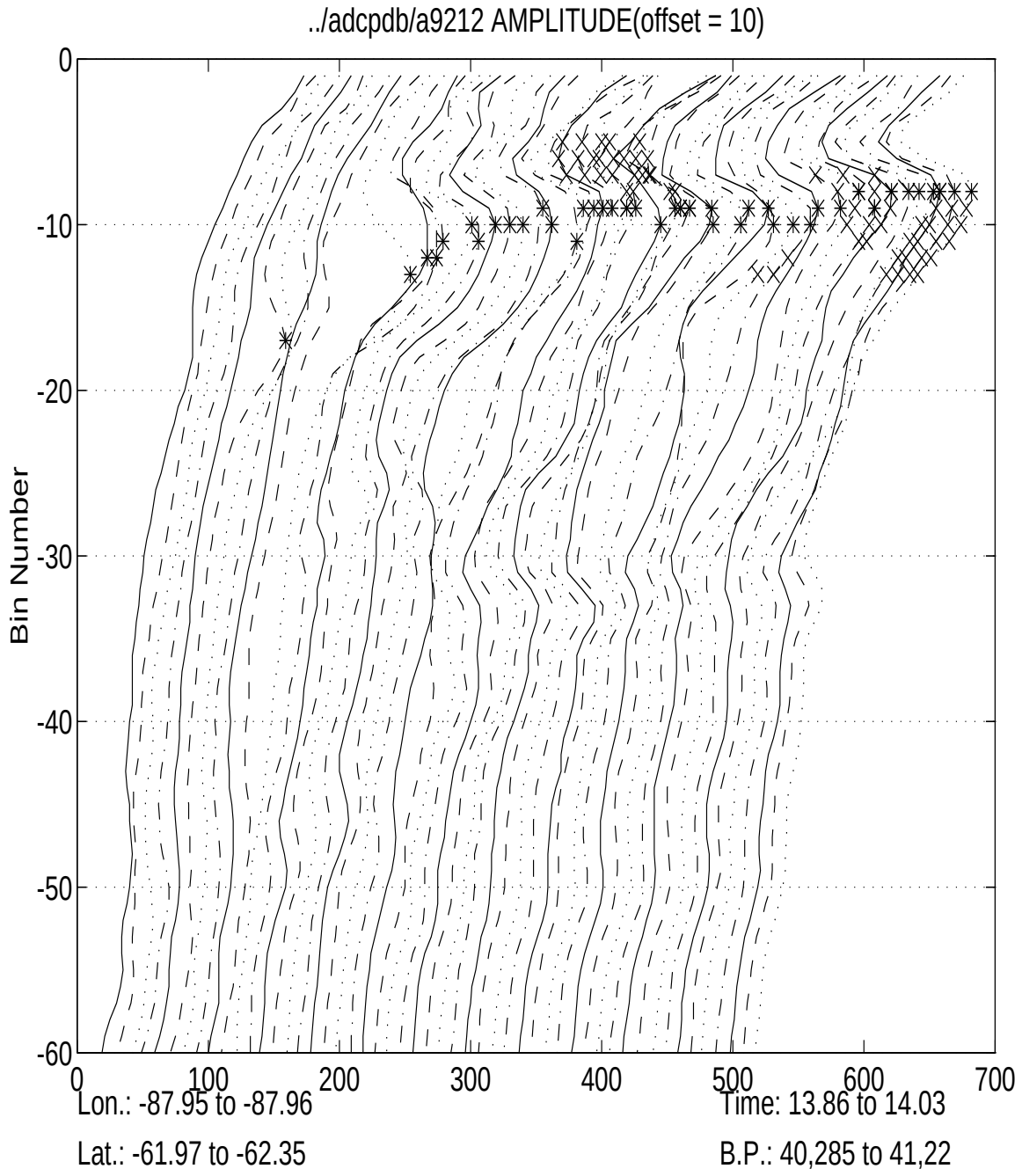


Figure 19: Scattering layer (*) depicted by sharp maximum in the Amplitude signal. This is a particularly dense layer. The ship was on station and peaks could be seen in the U and V velocities at about the same level. Since the ship was in deep water and since the U and V signals look OK below this level, it is determined to be a scattering layer. Another indication is that the strength of the amplitude maximum varies.

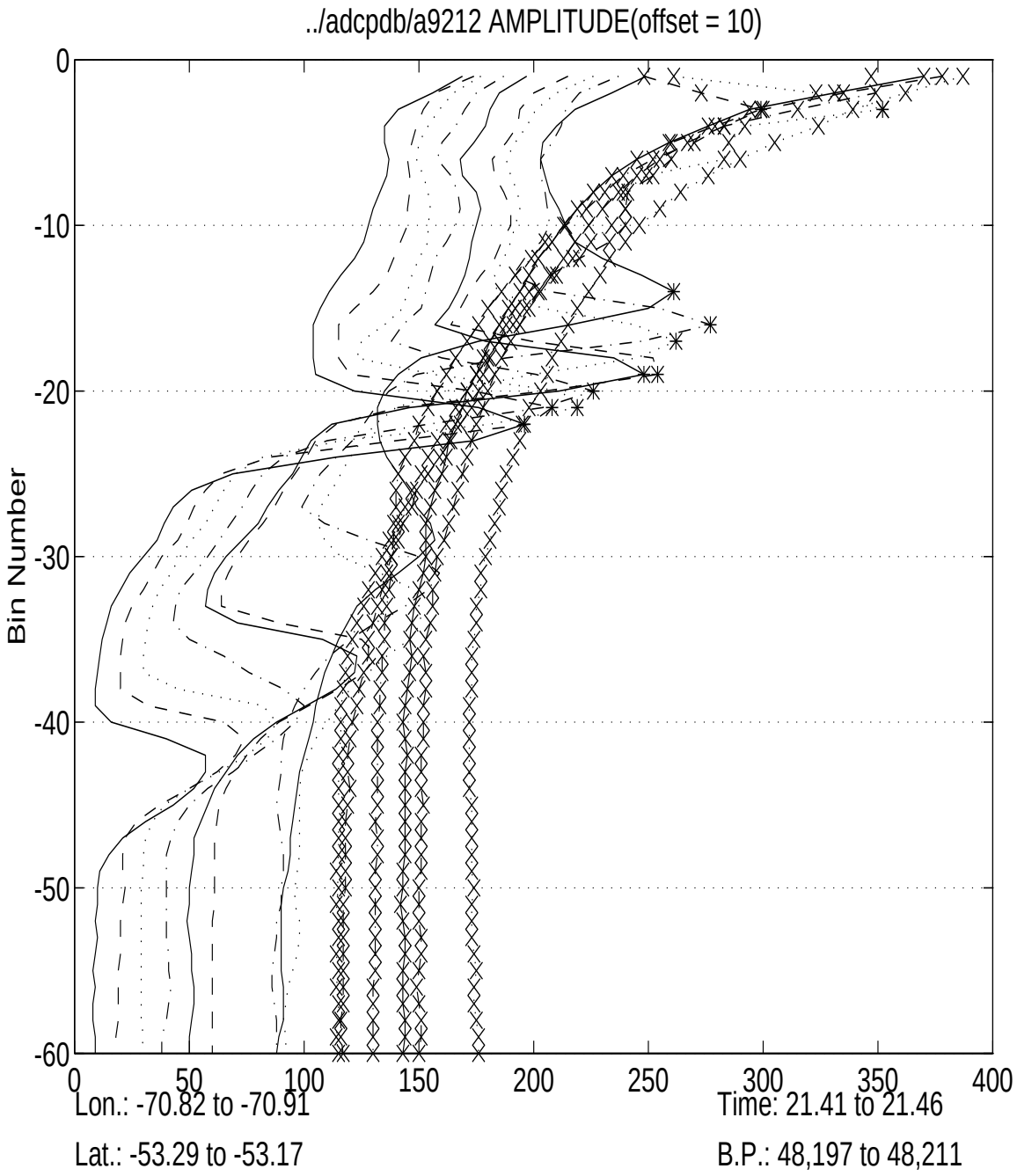


Figure 20: Bottom reflection (*) depicted by sharp maximum in the Amplitude signal as the ship enters port, but when the depth becomes very shallow (less than about 50 m), the amplitude signal gets very smooth. In this case, the entire profiles in the shallow water were edited out as indicated with the x's.

This sets the ACCESS_VARIABLES.1_last_good_bin to -1 to indicate that the entire profile is not to be used during subsequent access.

The next group of programs are used for establishing the bottom. First run:

dbupdate *path to database* bottom.asc

This sets the database variable ANCILLARY_2.max_amp_bin to the bin indicated in the file. Note that it does NOT set ACCESS_VARIABLES. last_good_bin, which access programs like adcpsect use during retrieval to determine the "good" portion of a profile (in addition to other editing criteria, of course, like percent good).

And to spread the bottom signal out among adjacent profiles, run:

botmpas3 *path to database*

Again, use showdb option 16, variable ID 38 and 39 to see the effect of botmpas3 on the database. The program updates ANCILLARY_2.last_good_bin and ACCESS_VARIABLES.last_good_bin as needed.

Finally, the last step, last_85, in bottom flagging is optional. During plotting, you should have been able to determine whether you want to do this or not. This program raises the bin of maximum amplitude (i.e. the bottom) by 15% to account for an error source due to the geometry of the beams relative to the bottom. To run this program, enter:

last_85 *path to database*

If you care to check all the effects of this editing at this point, you can run through the stagger plots one more time, this time setting all flagging criteria in setup.m to [], except for the access criteria PGOOD_THRESHOLD and USE_LGB. Set PLOTBAD=1 and EDIT_MODE=0. Just plot uv from beginning to end to check that you haven't missed anything.

Discussion of editing of fixes will be presented in the next Chapter.

7 Calibration

The basic methods of calibration are explained by [3] and [4]. In summary, there are two methods: 1) comparing the ship's displacement measured from bottom tracking with that determined from navigation data, and 2) comparing the acceleration relative to the water, measured with the ADCP, to the acceleration over the ground, calculated from navigation. These two methods should give identical results for the transducer orientation (relative to the gyro compass), but they can give scale factors that differ by something like 0.5%. Therefore the bottom track method is a useful supplement to the water track method, but cannot replace it. For both methods, the quality of the

satellite fixes and the gyrocompass should be examined, and if necessary, corrected prior to the calibration process.

Different authors have chosen various definitions of the calibration parameters. We have selected a convention which we find relatively easy to visualize and use. Express the velocity vector (u, v) as a complex number $U = u + iv$ and let subscripts u and c indicate uncorrected and corrected velocities, respectively. The calibration correction is then

$$U_c = Ae^{i(\phi\pi/180)}U_u.$$

A is the amplitude factor, a dimensionless number typically in the range 0.98–1.02, and ϕ is the counterclockwise angle in degrees (or misalignment angle) from the gyrocompass forward axis (which should be aligned with the ship's keel) to the transducer forward axis. When the ship is underway, errors in A and ϕ cause errors in the calculated absolute velocity that are proportional to the ship's speed. Expressed as a percentage of the ship's speed, a 0.6° error in ϕ causes a 1% error in the athwartships component of velocity, and a 1% error in A causes an error of the same magnitude in the forward component.

In the ideal case, calibration is done in two steps. First, apply a correction to the gyrocompass using GPS heading data. Then use water track and/or bottom track methods to calculate the net transducer offset.

In general, we start off with trying to calculate the transducer offset directly, mainly for benchmark purposes. If the cruise track was not especially complicated (not too many turns; no north-south speed changes), and the gyrocompass compensators were kept up-to-date, one may be able to find a good enough estimate of the transducer offset at this point, and not have to perform the gyro correction.

7.1 Examination of Ship Position Fixes

Nowadays, the ship position fixes are almost entirely obtained from the Global Positioning System (GPS) navigation systems. Other methods have been used in the past. The LORAN system is adequate but is limited to near-shore areas. Transit has been widely used up until recently yet the system will soon be discontinued. At the same time, the accuracy of the public-domain GPS data has improved as the military relaxes its selective availability restrictions and as new GPS technology arises.

The ship position fixes are an essential element of the ADCP database. They are used as a comparative tool during the calibration stage and are an integral component in the calculation of absolute current velocities. Thus, we examine and correct these important data prior to the calibration and navigation stages. The basic steps are:

- Extract the ship position fixes.
- Apply an automated editing program if applicable.

- Make plots of absolute currents which are a tool for identifying spurious fix values.
- Perform a final edit of the spurious values.

7.1.1 Extract Fixes

- **Case where fixes have been stored with the ADCP profiles.** The position data are stored within the CODAS database in the `NAVIGATION` structure, which can be viewed for a given ensemble with `showdb` option `16` (`GET DATA`, then enter either `68` or `NAVIGATION`). Optionally, position fixes may have been stored in a user buffer (Chapter 5.1) by use of a `user exit` program during acquisition on ship. These fixes are also loaded into the the CODAS database. Moreover, often the GPS data are logged independently and can be obtained as a separate file. Regardless, an important point is to know if the fixes are instantaneous positions or average positions over an ensemble. If they are averages, then it is best to assign the time of the fix as the midpoint of the ensemble.

  The first glance at the ship position data was done in Chapter 5.2 during the viewing of the cruise track, which provides a gross check on the quality of the fixes. Thus, within the `nav/` subdirectory, a file(s) of fixes should exist as output from either program `ubprint` or `getnav`. (Or if Transit navigation was taken, an ASCII file (`*.trs`) within `scan/` subdirectory could have been created during the `scanping` step (Chapter 5.1.) At this point, the program `ubprint` will be reviewed and greater detail on setting the parameter, `variables:`, in the control file `ubprint.cnt` will be explained (Appendix A.29).

  The parameter `variables:` controls the output options and depends on the source and type of the fix data. Within the `variables:`, the options of interest are the `*_summary` variables, as these provide the results in the appropriate format. Which of the `*_summary` variables to choose from depends on the type of user buffer written by the user-exit program. For 1320-type user buffers, the `avg_GPS_summary` option is typically chosen. Preferably, the fixes were recorded as close to the end of the ensemble time as possible. If fixes were recorded at the end of the first ensemble and beginning of the next ensemble, then these are averaged to obtain the best fix for the first ensemble. This option is set in with the `avg_GPS_summary` parameter. For 1280- and 1281-type user buffers, the `GPS_summary` option is appropriate. In this case the fixes are instantaneous positions at or near the end of the ensemble. The `TRANSIT_summary` is usually already obtained when the `scanping` program is run, so is no longer requested in the `ubprint` control file shown.

  Examples of several ship position files are shown in Figure 21. The first three columns are time in decimal days, longitude, and latitude, which were utilized

for the cruise track. The remaining columns vary among the file types as set by `variables:` option in `ubprint.cnt` and are used to assess the fix quality.

- **Case where fixes are stored in NMEA GPS-formatted files.**

  The installations that do not use an user-exit program similar to the UH system but generate NMEA-formatted GPS files or use the RDI program `navsoft` to store these in the user buffer structure can use the program `nmea_gps` to convert from that format to a text file with columns suitable for use with the succeeding steps. If the RDI `navsoft` program was used, the program `scanping` can be used to extract the user buffer ASCII messages to a text file, which can then be run through the `nmea_gps` program to produce columnar output of GPS fixes. See **??** on how to use the `scanping` program.

  The `nmea_gps` program uses a control file (Appendix A.18) that requires editing as follows. The `YMD_BASE:` parameter specifies the year, month, and day of the first observation in the time range requested for output. It is used to convert the NMEA file times to decimal days. The `TIME_RANGE:` specifies the time range over which to extract data. Setting it to `ALL` converts the entire input file(s). For navigation purposes, the `OUTPUT_ASCII_FILE:` must be specified (root only). This output file will have the extension `.gps` and can be used for the remainder of the navigation calculations. The `INPUT_GPS_FILES:` parameter precedes the list of NMEA GPS input files to be read from.

  The program is then run by typing

  `nmea_gps` *nmea_gps.cnt*

  A few lines from a sample output file are displayed in Figure 21. The first three columns provide the required decimal day, longitude and latitude information. The remaining columns provide quality indications: number of satellites, quality, horizontal dilution of precision (hdop), north component dilution of precision (ndop), east component dilution of precision (edop), vertical component dilution of precision (vdop), and altitude. This ASCII file can then be used in place of the `ubprint` `.gps`, `.ags`, or `.trs` files.

- **Other cases.** If the options above are not applicable for your source of ship position data, you will need to write a program to reformat position fixes from whatever source to text files, where the first three columns of each line provide the fix time in decimal days, and longitude and latitude in decimal degrees. If quality parameters are available, they could be recorded following the formats describe above for GPS or Transit Summary output files so they can be used with the `edfix` editing program in the next step. Otherwise, they could also be hand edited directly, and used with the post-editing steps described below.

```
==========> variable: avg_GPS_summary (*.ags) <=====================
  DD            LO             LA        NS  Q  HD    ALT
 338.7837616  -149.5695583   -17.5372167   3  1   4    16.00
 338.7872222  -149.5694000   -17.5369250   4  1   1    16.10
   .
   .

==========> variable: GPS_summary (*.gps) <=======================
  DD            LO             LA       NS Q HD ND  ED  VD  ALT
 190.7958333  -157.8863133    21.3161917  5 1  1  1   1   1  150
 190.7975694  -157.8864850    21.3160817  4 1  1  1   1   1  150
   .
   .
==========> variable: TRANSIT_summary (*.trs) <==================
  DD            LO             LA         EL NI   DR   F  DT
 190.8604630  -157.7488861    21.1823694   43  2  0.57  1  18
 190.9175579  -157.5449083    21.0662833   52  2  1.00  1  18
   .
   .
==========> NMEA_GPS output <====================================
  DD            LO             LA       NS Q HD ND  ED  VD  ALT
 191.7958333  -159.8863133    31.3161917  6 1  1  1   1   1  150
 191.7975694  -159.8864850    31.3160817  4 1  1  1   1   1  150
   .
   .
Key to columns:
DD:  decimal day
LO:  longitude
LA:  latitude
NS:  number of satellites
Q:   quality flag as part of the GPGGA message, 0 = GPS not available
                  1 = GPS available
HD:  HDOP, horizontal dilution of precision
ALT: altitude
ND:  NDOP, North dilution of precision
ED:  EDOP, East dilution of precision
VD:  VDOP, vertical dilution of precision
EL:  elevation of satellite
NI:  number of iterations
F:   flag (1 if fix accepted by navigator, 0 otherwise)
DT:  time difference in sec between PC and navigator
```

Figure 21: Sample ship position fix files (abridged).

The next step is to edit out bad fixes from the fix file(s). The preferred way of implementing this is by inserting a `%` character in the first column of the bad fix entry (thereby commenting out the entry as opposed to deleting the line). One may also wish to fill in data gaps from a particular fix source with data from an alternative source.

As mentioned previously, the **first** gross check on the fix quality is done by producing the cruise track. The **second** method is to use the program `edfix` to automatically edit out bad fixes. This method is limited due to the constraints on the input format; its applicability ends with the 1320-user buffer as it presumes certain columns. The 720/2240 user-buffer does not provide the same columns that edfix expects. However, the quality of the GPS data from these latter user buffer types is better, and in many cases does not require program `edfix`, and can be assessed in the following. The **third** method involves calculating and plotting absolute currents.

7.1.2 Editing the Fix file(s) with Edfix

The program `edfix` is used to automatically edit out bad fixes based on the quality criteria. The `edfix.cnt` control file is used to specify what type of editing is to be performed and whether or not the `.trs` file is to be merged into the `.gps`file (Appendix A.9).

One must first edit the `edfix.cnt` control file. The `output:` parameter specifies the output filename. The `transit_file:` parameter specifies the name of the Transit fix file (`.trs`) or can be set to `none` if the user does not want the Transit fixes merged into the `.gps` file. The next five lines define the editing criteria for Transit fixes (or are ignored if the user specifies `none` for the `transit_file:` option). The `min_elevation=` and `max_elevation=` parameters define the acceptable range of satellite elevation (we use 7°to 10°). The `max_iterations=` specifies the maximum number of iterations in the fix calculations (we use 3). The `max_dr=` specifies the maximum distance of the fix position to the dead reckoning position. The `time_since_gps` criterion edits out Transit fixes that occur too close to a Transit fix (we use .006 decimal days or under nine minutes roughly).

The `gps_file:` parameter specifies the name of the input `.gps` file. It is followed by the editing criteria for GPS fixes. The `max_hdop=` criterion specifies the maximum acceptable horizontal dilution of precision (we use an upper limit of 6). The `time_since_3` criterion specifies the maximum time elapsed since a three-satellite fix occurred and is calculated from the `number of satellites` and `time` columns. This is useful in the case where the GPS receiver has a timing device such that it can use two satellites and the clock to calculate a fix. We usually start with a value of .25 decimal day, meaning that a two- satellite fix will be edited out if the last three-satellite fix occurred less than 6 hours before.

After setting up the control file, the program is run by typing

`edfix` *edfix.cnt*

The output file will have an extension `*.edf`.

Regardless of whether `edfix` has been run or not, it is a good idea to visually scan the final fix file for remaining outliers. It should be noted that `edfix` edits based on the quality indices alone, and does not examine the fix values themselves for reasonableness. One may also choose to override some of `edfix`'s decisions if they result in too large a gap where data look reasonable enough. That is one reason why bad fixes are merely commented out rather than completely omitted from the output file.

7.1.3 Editing the Fix File Using Plots of Absolute Currents

Another useful way to assess the quality of the fix data is by calculating and plotting absolute currents for a reference layer. The steps involved are the same steps used in the Navigation stage of ADCP data processing, which is explained in detail in Chapter 8. For now, an explanation is given on how absolute currents can be used to gauge the fix quality and briefly on how to perform the steps, with the majority of the description of these routines postponed until later.

In a nutshell, the ADCP measures relative velocities between the ADCP transducer mounted on the hull of the ship and scatterers that are advected by the currents. In order to obtain absolute currents, one must subtract out the velocities of the ship relative to the earth coordinate system, which are determined by the fixes. Thus, assuming the ADCP data editing has been satisfactory, spurious features in absolute currents are attributed to erroneous fix values.

The general steps involved are as follows:

- **Obtain relative ADCP velocities of a reference layer.** One extracts from the database the ADCP measurements of the ship relative to a reference layer, which is an average between bins in the roughly 50 to 150 m range for each ensemble. In the `nav/` subdirectory, provide the usual parameters (path to database and database name, output files names, year base, choice of reference layer, time span, etc.) to the control file `adcpsect.cnt` (see Chapter 9 and Appendix A.1). Run in the usual way, (program name followed by control file). The ASCII output file of relative velocities will have extension `.nav`.

- **Calculate absolute current velocities of reference layer.** Fix data are used to calculate the ship velocity relative to the earth, which in turn are subtracted from the ship velocity relative to the reference layer[11] (output of `adcpsect`, `*.nav` file) to yield absolute reference layer current velocities at each profile time. This is done with program `refabs`. Edit the control file `refabs.cnt` (Appendix A.21) and supply the `reference_file:` (output of

[11] The ship velocities relative to the reference layer are the ADCP velocity relative to the reference layer NEGATED.

`adcpsect`, `*.nav` file), the `fix_file:` (fix positions), and `output:`, which should have file extension `.ref`. Run in the usual way.

- **Smooth the absolute reference layer current velocites.** Apply program `smoothr` to smooth the absolute velocities. This program does more than smooth the velocities as explained later in Chapter 8. Edit the control file `smoothr.cnt` (Appendix A.26). Provide `reference_file:` (output of `adcpsect`, `*.nav` file), `refabs_output:` (`*.ref`), and `output:` with file extension `.sm`. Leave the other parameters at their default values or refer to Chapter 8 for explanations of the options. Run the program in the usual way. The program produces a binary file with extension `.bin` that is used in the following step.

- **Plot the smoothed and unsmoothed absolute velocities.** Edit MATLAB script `callrefp.m` and specify the name of the binary output file from program `smoothr` (`.bin`) and the name of the ASCII output file from program `refabs` (`*.ref`). Enter MATLAB and type

  callrefp(dd)

  where dd is the day. This displays two days for each dd given (Figure 22). From the plot, we check the raw absolute reference layer velocity (thin lines) for spikes that may skew the smoothed estimates (thick dark lines). The '+' at the bottom of the reference layer velocity plots indicate gaps in the ADCP reference layer velocity data. If we do see suspicious spikes that obviously skew the smoothed values in both the u and v components, we edit the bad fixes out of the fix file by putting a `%` in front of the record, and verify its correction by redoing the steps above.

At this point, we should have a clean set of ship position fixes. Yet prior to performing the calibration, we first examine the gyrocompass data if possible.

7.2 Examination of Gyrocompass Data

The gyrocompass information is essential for rotating the ADCP velocities from transducer to earth coordinates. This rotation is done as the ADCP data are logged into the ping files by integrating the gyrocompass with the ADCP DAS. Thus, long-term or short-term drifts in the gyrocompass data lead to errors in the ADCP velocities. A few sources of these errors and ideas for correction are given below.

A simple gyrocompass points not due north, but along the axis of the total rotation vector: the sum of the earth's rotation and the rotation of the compass due to its relative motion on the earth's surface. The difference from due north is typically of order 1° for a ship, and is a function of latitude and of the north component of

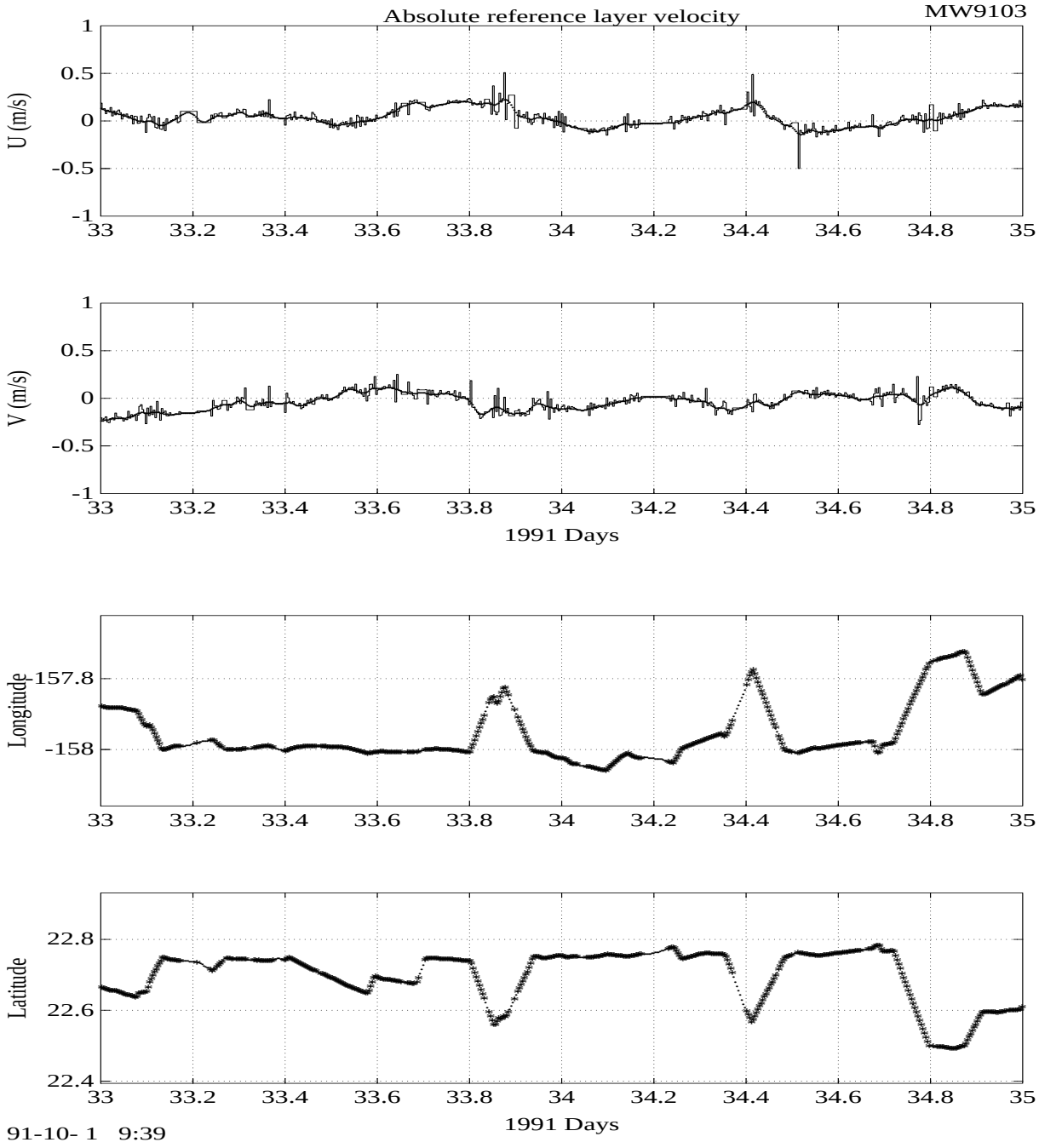


Figure 22: Sample `callrefp` plots. In this instance, there seem to be a few bad fixes that still need to be eliminated; the fix file was edited and the `refabs` and `smoothr` programs rerun.

the ship's velocity. Most gyrocompasses can compensate for this error, given correct settings of two dials, one for latitude, one for speed. Experience has shown that the latitude adjustment is most important and need only to have been correct to a few degrees. The ship captain usually is responsible for maintaining the compensation. However, it can easily be neglected and in some cases, untouched throughout the cruise. At the processing stage, it is difficult to know how and when the gyrocompass was compensated.

Another potential source of error from the gyrocompass is the Schuler oscillation. Literature indicates that the oscillation is excited mainly by the meridional component of acceleration of the ship on the time scale of the oscillation (84-minute period). If so, then calibrations should be more accurate on zonal cruise tracks than on meridional tracks, and on the latter should be more accurate when the ship stops than when it gets underway. It is not yet clear that these effects are detectable in the data we have looked at. Also, it is not clear to us how the amplitude of the oscillation should vary from one compass to another; the oscillations reported by [4] are larger than we seem to see on other ships and larger than we expect based on the calculations of [**?**].

The assumption underlying most discussions of ADCP calibration is that gyrocompass error may drift with time over a period of days to weeks, but at any time is independent of heading (apart from the short-term Schuler oscillation). This assumption has been demonstrated false in at least one case: a standard Sperry Mark 37 compass has been shown to have a difference of 1.5° between the errors on north versus south headings. We do not know yet whether this dependence of the error on heading is a symptom of compass failure, how long it has been occurring, or how commonly it might occur on other compasses.

The best possible means of verifying and correcting the gyrocompass data is to utilize heading information from an independent source, such as a dual GPS system or an Ashtech system. The systems consist of pairs of antennas in fixed locations aboard the ship. The heading data are relative to the geometry of the array, which in most cases is not exactly aligned with the keel of the ship. Thus, these supplementary heading data can monitor drifts in the gyrocompass data, but are not used for the absolute determination of ship heading.

The basic principles in using the independent GPS antenna array system is as follows. The gyrocompass data are compared to the antenna array system to obtain a file containing a correction for gyro drift as a function of ensemble time. This correction is applied to the ADCP velocities in the database with program `rotate`, which is explained later in this chapter[12]. Thus, during the calibration stage, the phase ϕ coefficient will be the angle between the transducer and the antenna system, which is a single constant for a cruise assuming neither was disturbed.

[12]The rotation of the database velocities can wait until after calibration. To see how the gyro correction affects the the calibration, one can choose to rotate only the output of `adcpsect`: the ship velocities relative to the reference layer (for water track calibration) or the ship velocities relative to the bottom (for bottom track calibration).

Another method of correcting gyrocompass errors is with a model. This model makes assumptions of how the compass was compensated for latitude and speed during the cruise. After each model run, the compensation versus heading data are plotted to see if they appear reasonable. If so, a file of time and offset angle is created and program `rotate` is used to correct the ADCP velocities. The model data are much less reliable than the antenna array system, but may be adequate for filling gaps if the latter is disrupted.

The software associated with the antenna array and gyro model have not been incorporated into the CODAS package at this time, but are available on a case-by-case basis. Please contact the authors for more information.

7.3 Water Track Calibration

In this method accelerations relative to the water as measured by the ADCP are compared to the accelerations over the ground measured with high-resolution navigation such as GPS. *The absolute water velocity is assumed to remain constant through the calibration period.* We typically use an average of two to three 5-minute ensembles on either side of a ship's speed or course change, separated by a similarly long time period centered on the time of the change. Any substantial acceleration during good GPS coverage can be used, so usually most calibration points are obtained in the course of a normal hydrographic cruise when the ship stops on station and then gets back underway. Additional calibrations can be obtained at very low cost in ship time by slowing the ship for about 20 minutes any time between stations or on deadheads.

Good-quality GPS is essential. It is also important that the fixes be obtained within a very few seconds of the end of each ADCP ensemble. If fixes are not being logged with a user exit program, then they should be logged externally very frequently (up to once per second), and the ADCP time (which comes from the DAS PC clock) must coincide with GPS time. (The ADCP time can be adjusted in postprocessing. If necessary, this adjustment can be determined to within a few seconds from the GPS fixes and the ADCP data alone.)

The quality of the calibration is dependent on the quality of the fix and gyrocompass data as described in the previous sections.

To calculate GPS calibration estimates, proceed along the following steps:

7.3.1 Obtaining a File of Position Fixes

If it has not already been done, run `ubprint` to get a GPS fix file, and `edfix` for automatic fix editing based on the fix quality messages (see section 7.1.1 for a detailed description).

7.3.2 Extracting Relative Ship Velocity

Run `adcpsect` to get a file containing only the ship's velocity relative to a reference layer as a function of time (see Chapter 9 and Appendix A.1). The relevant output file will have the extension `.nav`. The `step_size` must be `1`. The parameter `ndepth` must equal or exceed the deepest reference layer bin. A `pg_min` (for "minimum percent good") threshold usually provides adequate editing for this purpose. Typical values are 30 to 50. Don't go below 25. Include only `reference_bins` in the list of `navigation` options. The range should be chosen to be the thickest for which percent good is generally high (say 90 or above), except that one may wish to exclude the top few bins if they are likely to have higher variance than deeper bins. A typical reference layer bin range is 5–20. If the depth range of the ADCP was poor, one might want to use a shallower layer such as 2–10 or 2–15. Alternatively, the program can be run several times with different bin ranges (and different output file names!) so that any possible change in calibration amplitude factor (scale factor) with depth can be investigated. The `statistics` option generates an additional output file with the extension `.sta`.

7.3.3 Estimating Amplitude and Phase for each Acceleration

Using the output of `adcpsect` and `ubprint` (and `edfix`) as input, run `timslip` (Appendix A.27) to produce a single output file containing calibration parameters and quality indications for each ship acceleration that meets your specifications and for which GPS fixes are available. The output file name is completely specified by the user, but the extension `.cal` is customary.

The original function of `timslip` was to detect ship accelerations from the ADCP and compare them to the GPS record, sliding them around in time to find a time offset that provides a best fit. Hence the name "time slip". This function was needed because for some datasets the ADCP recorded time could be in error by as much as a few minutes, and with navigation data recorded independently, this offset caused substantial errors in the calculation of absolute reference layer velocity.

The control file parameters should be specified as follows:

- `fix_file_type` should always be `simple`; `HIG` refers to an archaic format that was the only one available for some early UH data sets.

- `min_n_fixes` will normally be the same as `n_refs`, and both should be an odd number, 5 or larger. They determine the time span, measured in ADCP ensembles, within which accelerations are detected and the "time slip" calculation is done. The only problem with specifying a long span is that it will cause the accelerations to be missed in the case of a brief CTD station, for example. With 5-minute ensembles we normally use 7, but occasionally go to 5 in order to catch the maximum number of accelerations.

- `i_ref_xx` indices should be chosen as (1 1 4 4) or (1 2 3 4) for `n_refs` of 5; (1 2 5 6) or 1 3 4 6) for `n_refs` of 7; (1 3 6 8) or (1 4 5 8) for `n_refs` of 9; and so on. These indices determine the ranges of ensembles that are used in calculating the calibration coefficients. We think the results are generally a little better with the first of the alternatives in each example above, because it excludes the two ensembles nearest the time of the acceleration.

- `up_thresh` and `down_thresh` should be about half the normal underway speed of the ship. The algorithm for detecting significant accelerations looks for runs of `n_refs`/2 consecutive ensembles for which the magnitude of the speed difference between the current ensemble and the ensemble `n_refs` ahead exceeds `up_thresh` for an acceleration and `down_thresh` for a deceleration. For a turn, the criterion to be satisfied similarly is that the magnitude of the course difference exceeds `turn_thresh` and the speed (for each ensemble involved) exceeds `turn_speed`. For the latter two parameters, values of 60° and 2 m/s are reasonable.

- `dtmax` should be just a few seconds more than the ensemble length. It is used to detect data gaps, so that only continuous data are used in the calibration.

- Leave `tolerance` at around 5.e-5 days (about 5 seconds); it specifies the tolerance for the iterative "time slip" calculation.

- `grid` should normally be `ensemble`. It determines whether all interpolations are to the grid of fix times or ensemble times. These times are nearly coincident for GPS fixes recorded with the UH user exit program, so the setting of `grid` is not critical.

- `use_shifted_times` should initially be set to `no`, again assuming the ADCP data have been loaded with time corrections so that ADCP times and fix times are both correct to within a few seconds. The `yes` setting causes the calibration factors for each acceleration to be calculated after locally adjusting the ADCP ensemble times for a best fit (using the "time slip" calculation). If the ADCP ensemble times and fix times are not actually in error, this adjustment is a response to noise, not signal, and therefore will tend to make the calibration factors less accurate. A third alternative is to set the variable to an integer number of seconds of shift. For GPS fixes collected through the Magnavox port A and listed via the `GPS_summary` option in `ubprint`, the usual value is 5 to 7 seconds. This is the typical amount by which the recorded fixes effectively precede the end of the ensemble. It is estimated by the mean value in the column labelled "shift" in the `timslip` output, and summarized under the label "navpc" in the `adcpcal` output.

The `timslip` output will normally include quite a range of values including several obviously wild points. These can be edited out manually if desired, but it is easier to just let `adcpcal` do that automatically, as described below.

7.3.4 Determining Phase and Amplitude

The basic methodology is as follows:

- **Prepare output of `timslip` for MATLAB.** Use a text editor to extract the tabular information of the `timslip` output file, that is, everything but the commented out header lines and blank lines, into another file. You may call this file whatever you like, but let's assume you called it `cal`. The reason for this step is that this file will be read into MATLAB as an ASCII file, and MATLAB does not allow any form of comments or text in such files.

- **Define clipping criteria for calibration points.** The routine `adcpcal` is configured by editing the section of the MATLAB script M-file `adcpcal.m` (Appendix B.1). The default values given in the demo version of `adcpcal.m` are good starting points, and there may be no need to change them at all. The idea of the editing is simply to remove outliers and points that contain no good information about the calibration factors. Poor calibration data, typically because of GPS deficiencies, is indicated by excessive values of "time slip" (set `clip_dt`), and by poor agreement between the velocity from GPS and that from the ADCP (set `clip_var`). Note that the variance criterion checks the column `var` rather than `min_var`; this is because `timslip` is usually run with no or only fixed time shift. The program still lists, however, the time by which it *would* shift. Outliers are directly indicated by large deviations of the calibration angle and amplitude from their medians (set `clip_ph` and `clip_amp`, respectively). When in doubt, set the editing parameters loosely at first, then look at the `adcpcal` output to see the standard deviations of `phase`, `amplitude`, and `nav-pc`, and the mean and standard deviation of `var`. Reasonable clipping points would then be 2 to 3 standard deviations of the first 3 of these, and the mean plus 2 to 3 standard deviations of `var`.

- **Perform clipping and plotting calibration routine.** Run MATLAB, load `cal` (type: `load cal.`; you do need the dot if the filename lacks an extension), and run the function `adcpcal`. The only compulsory arguments are the name of the array (which is the filename less extension) created from the file, and a text string that will serve as part of a plot and output label. Typically one would type something like: `adcpcal(cal, 'MW9004 cal 1A')`.

  The result will be several plots on the screen (hit a key after each to continue), which will be saved automatically (Figure 23). For each run, a summary of the calibration statistics will be appended to a file called `adcpcal.out` (Figure 24).

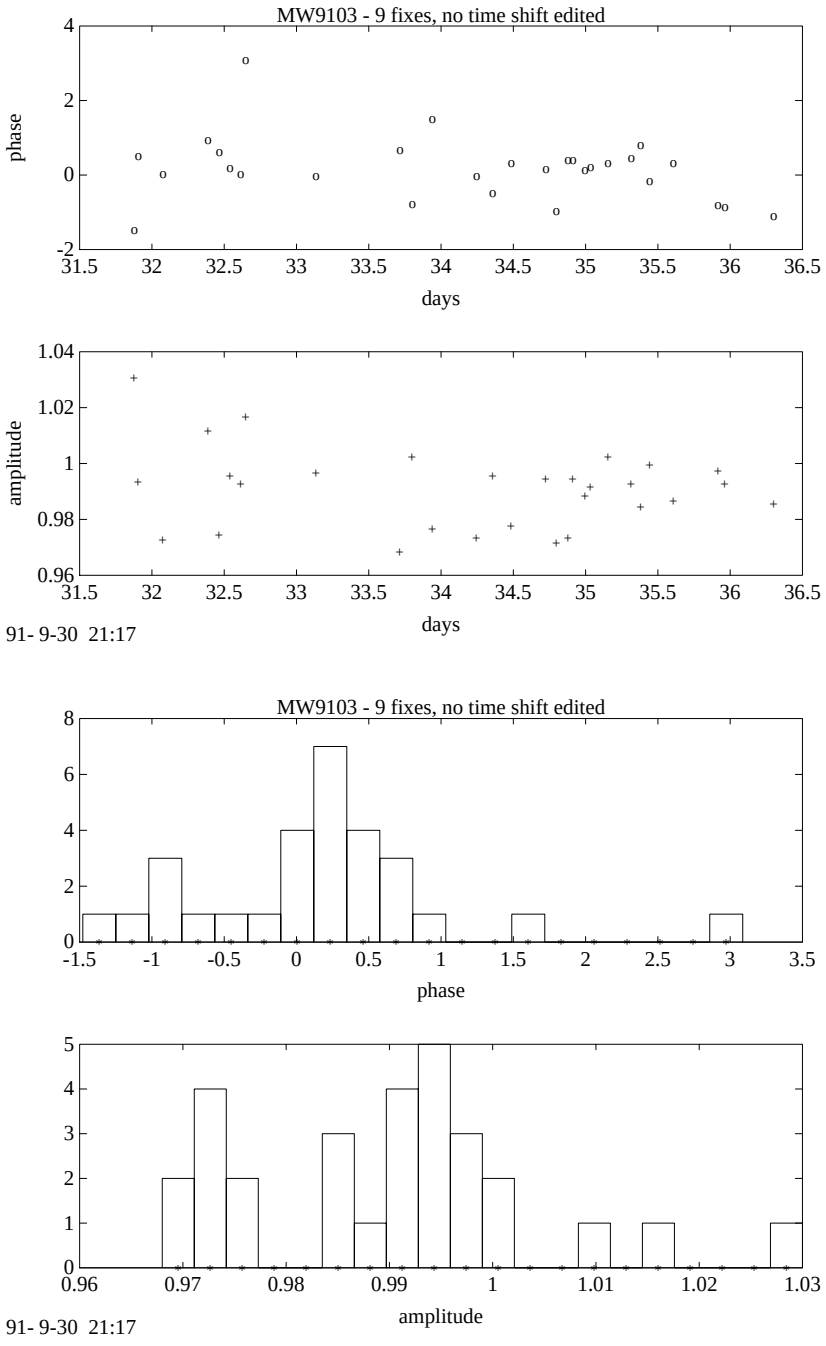


Figure 23: `adcpcal` example plots.

```
MW9302 (GPS_Gyro_Cor_Applied)- 7 fixes, no time shift
 Time range  61.94 to  88.25
 Calculation done at 93- 4-16  11:25
 delta-u min = -100.00, max = 100.00;
 delta-v min = -100.00, max = 100.00
 clip_amp = 0.04,  clip_ph =  3.0
 clip_dt =   60,  clip_var = 0.050
Number of edited points:  67 out of  79
 amp   = 1.0016  + 0.0001 (t -  76.4)
 phase =  -1.05  + -0.0018 (t -  76.4)
            median     mean      std
amplitude   1.0020   1.0016   0.0156
phase      -1.0770  -1.0500   0.9306
nav - pc    4.0000   3.8358  13.6766
var         0.0130   0.0154   0.0089
min var     0.0120   0.0135   0.0080
delta-u     0.1300   0.1837   3.9889
delta-v     0.0300  -0.0725   3.0223
```

Figure 24: Sample output file for `adcpcal.m`

This `adcpcal.out` summary repeats the clip parameters, gives a least squares linear fit of amplitude and phase as functions of time, and lists basic statistics for the columns of the `timslip` output file: `nav-pc` refers to the calculated time shift (in seconds) for best alignment of GPS and ADCP accelerations. A large number would indicate a PC clock error that was not properly corrected during loading. In that case, go back and correct your times! (There is a program `chtime` to perform a constant offset correction to the profile times in a CODAS database.) A small number (such as a consistent 7-second difference for earlier Moana Wave data) may just result from some delay in the system. It can be corrected by rerunning `timslip` with a specified time shift (the "third alternative" described above). This is preferable over letting `timslip` calculate the time shift for each fix, since the latter would most likely be more noisy. `var` stands for the variance of the difference between the GPS and ADCP reference layer velocities, i.e., the variance of the absolute reference layer velocity. `min_var` gives the same variance *after* the time shift has been performed. `delta_u` and `delta_v` stand for the velocity difference across the acceleration. For a section with many course changes, their mean values are fairly meaningless.

Typical results for standard deviation of phase and amplitude after editing are 0.6–1.2° and 1–1.5%, respectively. This level of jitter in the calibration calculation comes from GPS error plus actual variations in the ocean reference layer velocity during each calibration acceleration interval. In addition, phase tends to be associated with larger velocity variance than amplitude because of errors in the gyrocompass, both short-term Schuler oscillations and long-term drift. Feel free to try out whether different settings of the `timslip` parameters such as longer or shorter groups of ensembles reduce the standard deviations.

Further, the `up/down` column from the `timslip` output could be used in MATLAB to differentiate station arrivals (-1) or departures (1). Calibration points from arrivals are less affected by Schuler oscillations, since here the ship is stopped when the compass oscillations occur so that the ADCP profiles are not contaminated by "leakage" from the ship's velocity. However, in one dataset we looked at (US–PRC 8) the arrival calibrations turned out to be slightly more noisy than the departure results. This could have been due to the sharper speed change on station departure when the Chinese ship practically "jump-started", while the station arrivals occurred as prolonged drifts to zero speed after the engines were shut down.

Ideally, over the course of a month-long cruise, one obtains 50 or so calibration points. The plot of phase and amplitude versus time may suggest a gradual drift in phase of up to about a degree, and perhaps a very small drift in amplitude. One can fit a curve by eye or numerically. For the latter, one method we have used is to divide the calibration points into groups, perhaps weekly, average them separately, and use spline interpolation to run a smooth curve through the averages. In addition, one can incorporate the results of bottom track calibrations, usually at the start and end of the cruise.

7.4 Bottom Track Calibration

In the bottom track method, the calibration parameters are derived by comparing the ship's track over the ground, as determined by navigation, to the track, as integrated from the ADCP bottom track velocity. While the more frequent and accurate GPS navigation is preferable, Transit fixes may be adequate for a long run in shallow water (50–500 m).

We have found bottom track calibration amplitudes to be biased toward larger values compared to water amplitudes. This bias arises from the non-orthogonal angle between the ADCP beam and the scatterer (the bottom). Those sections of the bottom that are closer to the transducer will produce a stronger echo and thus dominate the return signal. Their corresponding along-beam velocity component is smaller than that for the outer, farther areas. With the signal processing filter centered on the maximum amplitude, a bias toward smaller Doppler shifts, i.e., smaller ADCP velocities and thus larger amplitude factors is the result. The CODAS package allows for separate implementation of water track and bottom track calibrations.

7.4.1 Obtaining a File of Position Fixes

If it has not already been done, run `ubprint` to get a GPS fix file, and `edfix` for automatic fix editing based on the fix quality messages (see section 7.1.1 for a detailed description).

7.4.2 Extracting Bottom Tracking Velocity

Edit `lst_btrk.cnt` (Appendix A.14) and run program `lst_btrk` to retrieve the bottom velocities from the ADCP database. The `step_size` should be `1` to select every profile. The time range is not critical as long as it overlaps the time of bottom tracking. The program is executed in the usual way:

`lst_btrk` *lst_btrk.cnt*

In the example output in Figure 25 (file extension `.btm`), the first four profiles should be treated with caution (eliminated) because of too shallow water (1st) or due to the bottom obviously changing very rapidly during the five–minute ensembles. The bottom track time interval should overlap the satellite-fix time interval (i.e., you have a longer bottom track interval available), not vice versa. If not, eliminate (mark with %) the fixes preceding the start of bottom tracking before continuing.

The output file from `lst_btrk` should be split into pieces based on continuity (no gaps) and homogeneity of ship heading and bottom depth. This is accomplished with a text editor resulting in separate files for each continuous, homogeneous section of bottom tracking. The bottom track calculation routines are repeated over each such piece to come up with an estimate of the transducer offset.

```
%Bottom Track time, u, v, depth for :
%,  dbname: a9212,  output: a9212.btm,  step_size= 1,  year_base= 1992
%

%
 338.797535   -2.491   -1.165     14
 338.801007   -3.950    1.790     23
 338.804502   -1.554    1.953    327
 338.807975   -1.101    1.494    458
%
 386.723831    2.326    5.561    414
 386.727303    2.073    5.715    132
 386.730787    2.152    5.559    104
 386.734248    2.181    5.508    124
 386.737720    2.163    5.526    120
 386.741204    2.130    5.657    113
 386.744664    2.156    5.653    107
   .           .        .        .
   .           .        .        .
   .           .        .        .
```

Figure 25: Sample output file of `lst_btrk` (first few lines only). Columns represent time (decimal days), zonal and meridional bottom speed, multiplied by -1 to indicate ship velocity over ground, and bottom depth in meters.

7.4.3 Calculating Absolute Ship Velocity

Run `refabsbt`, a modified version of the program `refabs` used for estimating absolute currents, to match up the bottom track velocity and satellite fix and to calculate the zonal and meridional displacements from each information source (Appendix A.22; The control file parameters should be clear from the comment section of the control file; the `fix_file_type` should be simple. The `gap_tolerance` specifies the length of small interruptions that will not be treated as gaps. Execute the program in the usual way:

`refabsbt` *refabsbt.cnt*

The output file (Figure 26) contains the columns time, zonal and meridional displacements between two consecutive fixes in meters, zonal and meridional displacements calculated from bottom track, zonal and meridional gap velocity (cumulative), and gap time. The last three columns should show only zeros, i.e., there should be no interruptions in the bottom track record. The `refabsbt` program would (probably) interpolate over gaps, but that may influence the calibration values in an unwanted manner. If ADCP interruptions had occurred during bottom tracking, one must calculate separate calibration factors for each continuous record (i.e. the reason for creating separate files out of the `lst_btrk` output).

7.4.4 Determining Phase and Amplitude

Before invoking MATLAB to calculate amplitude and phase, edit the MATLAB script M-file `runbtcal.m` (Appendix B.5.

Now you are ready to invoke MATLAB and:

- Assign the value 1 to the variable $first$;
- Run `runbtcal`, which in turn calls `btcal`. This calculates a complex calibration factor aa by least-squares fitting complex arrays (in the form of $lat + i * lon$) of bottom track (bt) and satellite fix (nav) displacements according to the equation:

  $$nav = aa * bt + error$$

  . Amplitude amp and phase pha are obtained as

  $$a = abs(aa)$$

  ;

  $$ph = angle(aa) * 180/pi$$

  and written out to a text file called `btcal.out`, together with fit residuals $res = nav - aa * bt$ (Figure 27). The function also plots the residuals (Figure 28 and Figure 29).

```
  TIME            FZD       FMD        BZD       FMD     ZGV   MGV    GT
 386.7273380       595      1678        646      1668    0.00  0.00   0.0
 386.7308102       650      1678        646      1668    0.00  0.00   0.0
 386.7342824       637      1680        654      1652    0.00  0.00   0.0
 386.7377431       629      1613        647      1653    0.00  0.00   0.0
 386.7412269       659      1671        641      1703    0.00  0.00   0.0
 386.7446875       654      1698        645      1690    0.00  0.00   0.0
 386.7481713       618      1694        652      1689    0.00  0.00   0.0
 386.7516435       655      1697        611      1676    0.00  0.00   0.0
 386.7551157       307      1645        352      1669    0.00  0.00   0.0
 386.7585764       338      1654        315      1658    0.00  0.00   0.0
   .
   .
   .
KEY TO COLUMNS:
TIME:    decimal days
FZD:     zonal displacement between two consecutive fixes (meters)
FMD:     same for meridional
BZD:     zonal displacement determined by bottom tracking (meters)
BMD:     same for meridional
ZGV:     zonal gap velocity (cumulative) (should be 0)
MGV:     same for meridional
GT:      gap time
```

Figure 26: Sample output file for `refabsbt` (abridged).

```
ADCP KNORR 9212 (deleting none); amp=  0.9947, ph=   0.187
       day       dx       dy
  386.72734      -20      -22
  386.73081       -7       -5
    .             .        .
    .             .        .

ADCP KNORR 9212 (deleting 8); amp=  0.9948, ph=   0.181
       day       dx       dy
  386.72734      -16      -20
  386.73081       -3       -3
    .             .        .
    .             .        .

ADCP KNORR 9212 (deleting 8,11); amp=  0.9948, ph=   0.180
       day       dx       dy
  386.72734      -13      -19
  386.73081       -1       -2
    .             .        .
    .             .        .

ADCP KNORR 9212 (deleting 8,11,21); amp=  0.9951, ph=   0.193
       day       dx       dy
  386.72734      -14      -15
  386.73081       -2        1
    .             .        .
    .             .        .

ADCP KNORR 9212 (deleting 8,11,21,10); amp=  0.9951, ph=   0.190
       day       dx       dy
  386.72734      -11      -14
  386.73081        2        2
    .             .        .
    .             .        .

ADCP KNORR 9212 (deleting 8,11,21,10,16); amp=  0.9952, ph=   0.200
       day       dx       dy
  386.72734      -11      -14
  386.73081        2        2
    .             .        .
    .             .        .
```

Figure 27: Sample output file for `btcal`. The columns dx and dy are in meters. For each run, the rows (time) are abridged. As one sees, the amplitude does not change greatly, although the phase changes slightly (0.02) as points are edited out.

- Examine the plot of residuals. If it contains obvious outliers, run `runbtcal` again. The fix corresponding to the largest outlier will be eliminated, and amplitude and phase will be calculated again based on the smaller number of fixes and bottom track displacements (the bottom track segments around the eliminated fix are combined to provide integration between the remaining fixes). The results will be appended to the ASCII and output file.

- Repeat the above step as many times as there are outliers. The amount by which the calibration results change gives you an indication of how influential the outliers were. Usually a point is reached very quickly beyond which the reduction of residuals does not significantly change the results.

7.5 Rotation

When you have analyzed all available calibration information and decided what calibration parameters to use, the program `rotate` is used to perform the calibration on the velocities in the database.

Appendix A.23 shows an example of a control file for the `rotate` program.

The `DB_NAME:` parameter specifies the name of the CODAS database to be rotated. The `LOG_FILE:` specifies the name of a text file that the rotation parameters are recorded into. The next parameter may either be a `TIME_RANGE:`, `BLOCK_RANGE:`, or `DAY_RANGE:`, whichever is most convenient for specifying the range of profiles to be rotated. The parameter may be set to `all` if all profiles will be rotated. This is followed by an `OPTION_LIST:` that indicates which track is to be rotated (`water_track`, `bottom_track`, or both `water_and_bottom_track`). This permits specification of different rotation parameters for water and bottom track velocities. For the specified track, the user must then specify the rotation parameters: `amplitude=` of the rotation and a constant angle `angle_0`. In the case of heading-related calibration, the user may also specify values for `angle_sin_H=` and `angle_cos_H=`, the coefficients of, respectively, the sine and cosine of the ship heading. In the case of a time-varying rotation, the user may also provide values for `angle_1`, `angle_2`, and `angle_3`, as needed for linear, quadratic, and cubic time terms, as well as supply the mean profile time `time_0` (in decimal days) and the `year_base` used to express `time_0` in decimal days. The rotation of the horizontal and/or bottom track velocities is performed as described by the following equation:

$$U_c = Ae^{i(\phi\pi/180)}U_u$$

where

U_c is the corrected u-component of the velocity

U_u is the uncorrected u-component of the velocity

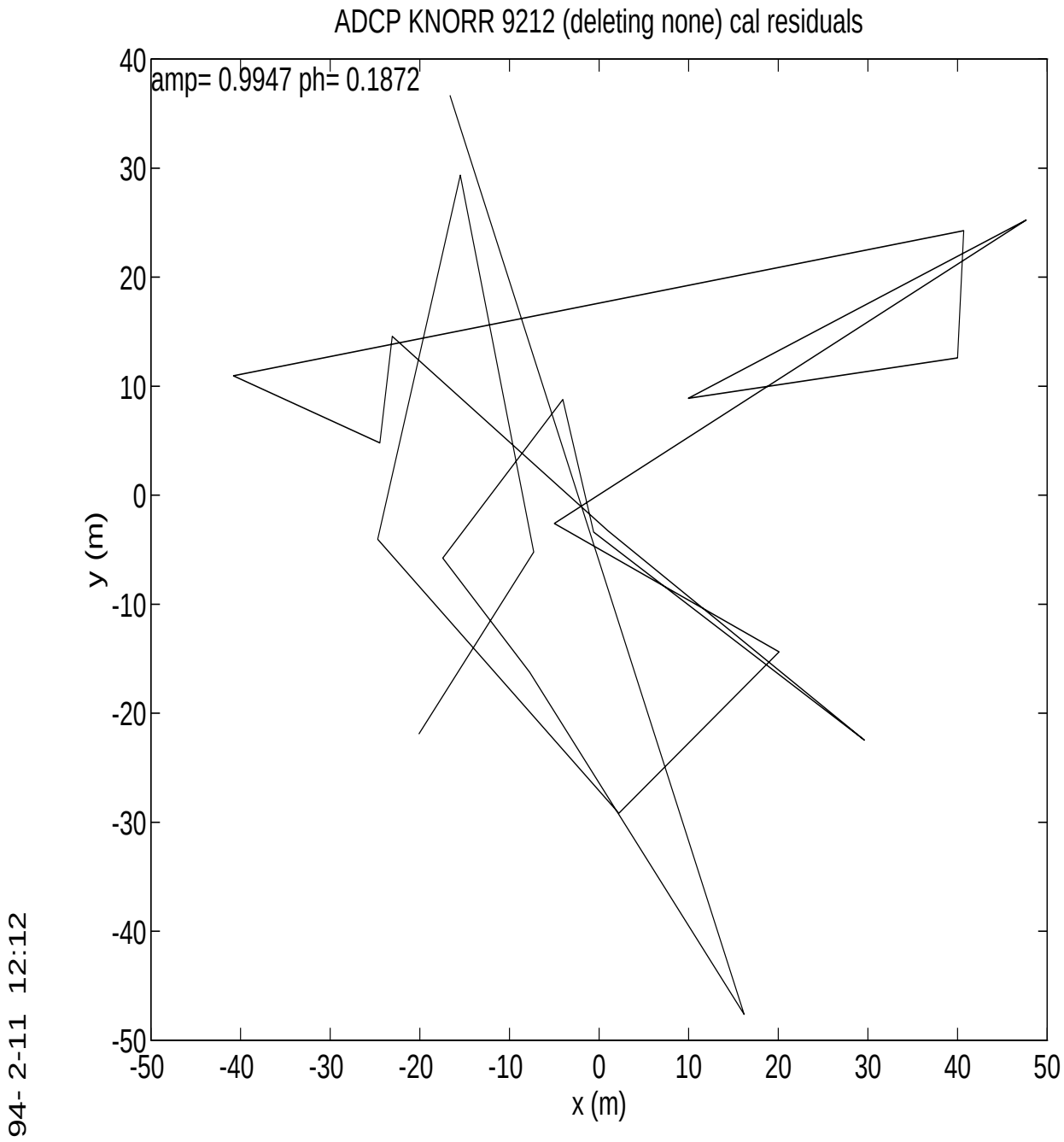


Figure 28: Initial run of `runbtcal` showing plot of residuals. The points are already within 50m which is considered good.

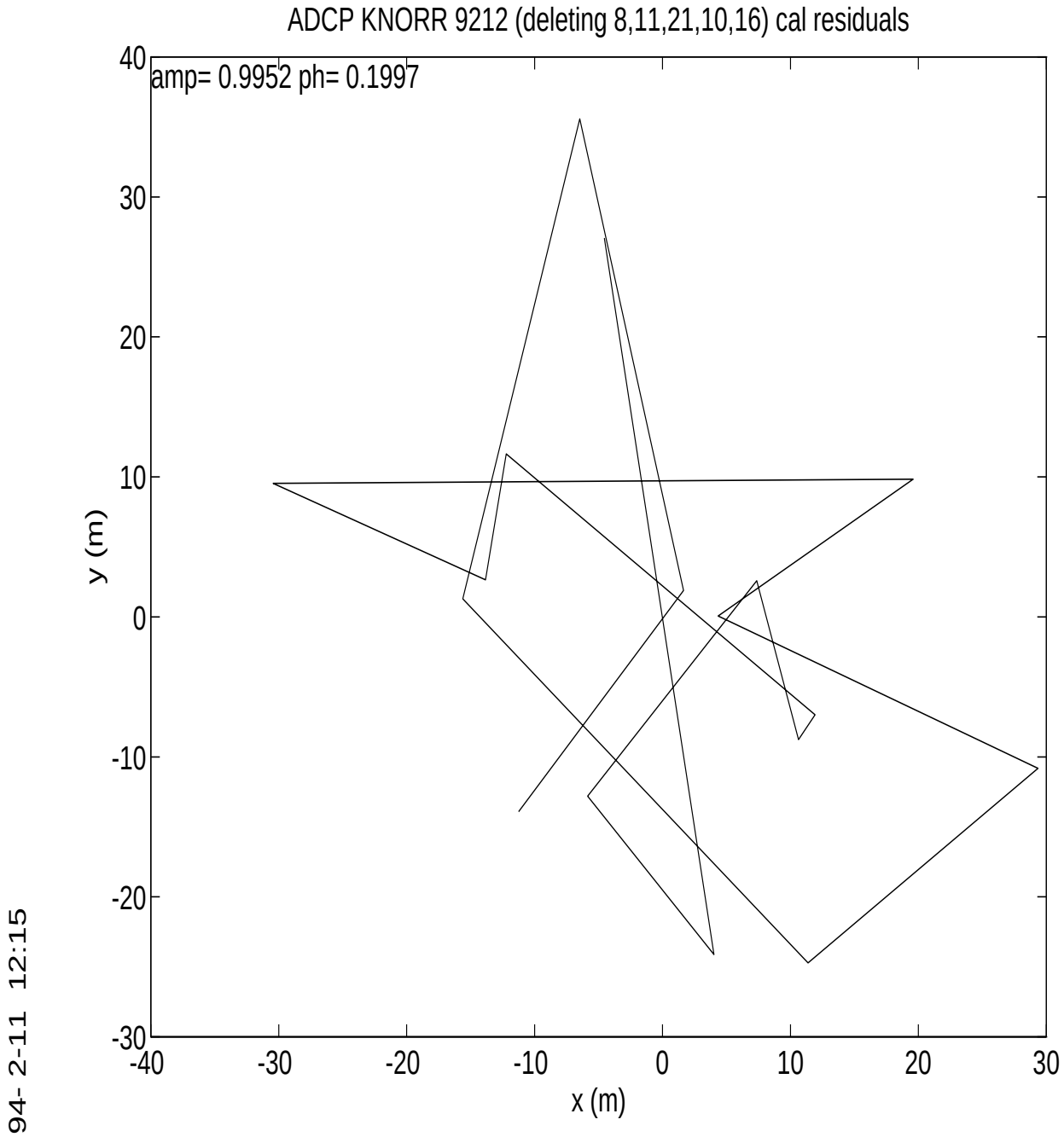


Figure 29: Subsequent run of `runbtcal` showing plot of remaining residuals after several outliers were removed. The clipping of outliers is not changing the phase and amplitude significantly so the calibration run has stablized. This calibration iteration is performed for each "piece" of continuous, homogeneous bottom tracking.

A is the `amplitude` factor

angle is given by

$$\begin{aligned}
&\texttt{angle}_0+\\
&\texttt{angle}_1 * (profiletime - \texttt{time}_0)+\\
&\texttt{angle}_2 * (profiletime - \texttt{time}_0)^2+\\
&\texttt{angle}_3 * (profiletime - \texttt{time}_0)^3+\\
&\texttt{angle}_{\texttt{sin}_H} * sin(shipheading)+\\
&\texttt{angle}_{\texttt{cos}_H} * cos(shipheading)
\end{aligned}$$

Failing to specify a parameter results in the following default value for the omitted parameter:

`amplitude=` 1.0

`angle_0=` 0.0

`angle_1=` 0.0

`angle_2=` 0.0

`angle_3=` 0.0

`angle_sin_H=` 0.0

`angle_cos_H=` 0.0

`time_0=` 0.0

`year_base=` the year of the earliest profile in the database

8 Navigation

This chapter describes how to use the UH ADCP Processing System to calculate: 1) the ship's position at the end of each profile, and 2) the average velocity of the ship during the profile. This information will then be stored with the CODAS database for use as a reference for calculating absolute currents in geographical coordinates.

Navigation for our purposes requires three quantities:

1. ship's heading
2. ship's velocity relative to some vertical average of the water column (reference layer), and

3. position fixes.

Heading is needed only for transforming the ship's velocity into geographical coordinates. It is normally provided by the ship's standard gyrocompass. In this manual, we assume that the transformation has been directly applied by the DAS to the ping data. Any further corrections required due to inadequate compensation or other gyrocompass error sources will have been applied during the calibration stage (see Chapter 7).

The reference layer should be chosen so that it is expected to be as smoothly varying as possible along the cruise track. Hence one wants the thickest layer that usually contains good data, and that perhaps omits the first few bins near the surface. We normally use ADCP bins 5–20, corresponding to about 50–170 m with 8-m bins. If bad weather or poor ADCP range is expected, we might reduce the range to bins 5–15, or perhaps 3–15. However, there is nothing to be gained by frequent fiddling with the bin range.

Satellite navigation is the primary source of position fixes for cruises outside the range of Loran C. It is preferable that the fixes be instantaneous rather than averaged fixes, recorded as near as possible to the ADCP ensemble times. In addition to fix information, other quality data, such as horizontal dilution of precision and the number of satellites are also desirable as a reference when editing fixes.

To facilitate matching up the ADCP ensembles with the fix information, we have integrated the satellite navigators into the ADCP system by writing a "user-exit" program, as explained in Section 5.1. This program allows the satellite information to be recorded in the user buffer during data acquisition. (This program is available upon request.) Such integration is *not* absolutely essential to proceed with the navigation calculations described below.

The navigation calculations begin with obtaining:

1. a text file of position fixes from which bad fixes have been edited out

2. a text file of ship velocity relative to the reference layer from which questionable sections have been edited out (e.g., calibration runs over which velocity is recorded relative to the ship coordinates, or periods of malfunction in the ADCP heading data recording).

The basic idea behind the navigation step is as follows. Given these two files above as input, the absolute reference layer velocities are calculated between navigation fixes, then smoothed and interpolated to the ADCP profile times. The results are plotted using MATLAB and reviewed to determine whether different smoothing settings should be used or if any other bad fixes need to be edited out from the fix file. The appropriate programs are rerun and the reference layer velocities are replotted until the results are satisfactory. The resulting absolute ship velocity and profile positions are then stored as part of the CODAS database, and applied during data extraction to derive the absolute current velocities. The database (relative) velocities remain unaltered.

8.1 Generating and Editing a Fix File

The user should start by deriving an ASCII file of fixes, where the first three columns are 1) time in decimal days,[13] 2) longitude and 3) latitude in decimal degrees. All that is required format-wise is that the columns be delimited by white space. The source of such a file would be installation-dependent. In general, we can distinguish among three sources:

1. the navigation variable recorded with the raw (ping) data files. The navigation variable contains a longitude and latitude field, but no time, so the fix usually "inherits" the ensemble time. In the case of installations using the UH user-exit program, the fix is grabbed right before the ensemble is written to disk, so the discrepancy between the actual fix time and the ensemble time will normally be under ten seconds. Installations using RDI's `navsoft` program use the navigation structure to store a fix that has been averaged over the ensemble, in which case, it may be more appropriate to subtract half the ensemble length from the ensemble time when assigning the fix time.

2. the user buffer recorded with the raw (ping) data files. The user buffer, if used, may also contain more fixes, and will probably be raw fixes. In the case of the most recent version of the UH user-exit program `ue3` (the one that yields either a 72-byte or 224-byte user buffer), a couple of other fixes will be available from the user buffer, along with the satellite time, and are grabbed at the start and middle of the ensemble. In the case of RDI's `navsoft` program, the user buffer, if used, would normally store one or more NMEA messages that have been grabbed early in the ensemble. (For users of this program, we normally recommend being selective about which messages to record ($GPGGA is good), and using a user buffer size equivalent to twice the length of the expected messages in order to increase the chances of getting at least one complete fix message per ensemble; then we can eliminate the need for using the averaged fixes stored in the navigation structure.)

3. an external file recorded independently of the ADCP. Finally, an external file will probably have all the information that is required—time, longitude, and latitude—and all that may be needed is to transform it into columnar format of the right sequence and delimited by whitespace. Hopefully, the fixes will be raw rather than averaged. The main concern then is whether the ADCP ensemble times are accurate and will line up neatly with the fix file. For installations that use the user buffer to store satellite fixes that include time, we normally use this information to correct the ADCP ensemble times and thereby achieve synchronization either at the data acquisition stage (`ue3` installations with the

[13] Again, we emphasize that decimal days are counted from 0.0 on 00:00:00 of 1 January of the base year

`correct_clock` option enabled in the configuration file `ue3.cnf`), or during the loading stage (`loadping`, using the `time_correction` option).

It is not at all unusual to use a fix file that may be some combination of the above. A receiver failure causing an extended gap may make it necessary to patch in fixes from an external file into the user buffer record.

While not necessary, it is a good idea that the fix file be sampled as close as possible to the ADCP ensemble times. For installations that use the UH user-exit program, this is automatically the case. External GPS sources, on the other hand, would typically use a much shorter sampling interval; much of the noise in the subsequent navigation plots can be eliminated by resampling closer to the ADCP ensemble interval.

Below we describe several programs that have been written for the purpose of extracting and editing the fix file.

8.1.1 Extracting Fixes from the Navigation Structure

When fixes need to be retrieved from the navigation variable recorded with the raw data files and loaded into the database, the program `getnav` is used. This program outputs the ADCP ensemble time, and the longitude and latitude fields of the navigation structure. The user must set up the control file `getnav.cnt`, an example of which is copied by the `adcptree` script into the `nav/` directory. The control file parameters are self-explanatory (see Appendix A.11). Note that if the fixes stored in the `NAVIGATION` are averages over the ensemble, the time column in the `getnav` output file should probably be reduced by half the ensemble interval.

8.1.2 Extracting Fixes from the User Buffer

There are two programs that can be used for extracting fix information from the user buffer, depending on the format type.

User buffers that contain ASCII strings should be extracted during the `scanping` stage, using the `UB_OUTPUT_FILE` option together with `USER_BUFFER_TYPE` specified as `ascii`. The resulting file will contain all available text from the user buffer. This would generally need to pass through an intermediate step to convert it into the columnar format required for subsequent processing steps. An example of such an intermediate program is `nmea_gps`, which can parse one or more ASCII files containing NMEA-0183 messages. The user will need to develop his or her own means for converting other types of messages to columnar format.

User buffers that have been generated by one or another version of the UH user-exit program can be extracted anytime after loading the database using the program `ubprint`. To run the program, the user must set up the control file `ubprint.cnt`. An example of this control file is copied to the `nav/` directory by the `adcptree` script and is given in Appendix A.29.

The `dbname` parameter specifies the name and path, if needed, of the CODAS ADCP database from which to read the user buffer.

The `output` parameter gives the root to use for the output filename(s). The appropriate extension(s) is (are) appended, depending upon the specification of the `variables` parameter.

The `step_size` parameter is used to specify a sampling interval. We normally use a value of 1 for deriving fix files to be used for calibration and navigation calculations.

The `year_base` parameter is used to establish the base year in calculating decimal days. It is normally set to whatever the year is for the first observation in the dataset.

The `variables` keyword precedes a list of one or more output options terminated by `end`. Only the `*_summary` options deliver columnar output. All the rest are more verbose and useful only for detailed checking.

- `TRANSIT_summary` option extracts Transit fixes from UH user buffer types 1020 through 1320 to an ASCII file with extension `.trs`.
- `GPS_summary` option extracts GPS fixes recorded at the end of each ensemble in UH user buffer types 720, and 1280 through 2240 to an ASCII file with extension `.gps`.
- `avg_GPS_summary` option prints the average of the GPS fix at the end of the ensemble and the fix at the beginning of the next ensemble, which should normally provide a position closer to the ADCP ensemble time. This option works only with UH user buffer types 720 and 2240.

It is possible to request for more than one variable on a single run, and the appropriate extension will be appended to the different output files. When possible, the `avg_GPS_summary` is recommended. Otherwise, the `GPS_summary` is used. When necessary (i.e., major gaps occur in GPS coverage), the `TRANSIT_summary` output can be used to supplement the GPS fix file for older datasets. The merging of the GPS and Transit fix files can be done using the program `edfix`, described below (Section 8.1.5).

8.1.3 Converting NMEA-Formatted Fixes to Columnar Format

The program `nmea_gps` was specifically written to facilitate the conversion of an ASCII file containing NMEA-0183 messages to columnar format compatible with the subsequent navigation and calibration steps. (It also features an option for generating MATLAB Mat-files.) An example of the control file is given in the `codas3/cntfiles/` subdirectory and printed in Appendix A.18. The control file requires editing as follows:

The `YMD_BASE` parameter specifies the year, month, and day corresponding to the first observation in the first input file. Since NMEA messages track time only as hour, minute, and second, this parameter is used to establish the day of the year. Note that

the program simply increments this parameter by one each time a day boundary is crossed. So files that may include a gap of more than a day should be processed on separate runs to keep this day counter straight.

The `TIME_RANGE` parameter specifies the time range of interest, so the program limits extraction to this subset. Setting it to `ALL` converts the entire input file(s).

The `OUTPUT_ASCII_FILE` parameter must be specified to be other than `none` so an ASCII file with columns time, longitude, latitude, etc. will be generated. Only the filename root should be specified, since the program automatically appends the extension `.gps`.

The `OUTPUT_MAT_FILE` and associated options are not necessary, unless the user wishes to use MATLAB as a tool for editing and/or sampling the fixes, and then save the results to another ASCII file.

The `INPUT_GPS_FILES` keyword precedes the list of NMEA GPS input filenames from which the fixes are to be extracted.

The program is then run by typing

`nmea_gps` [*nmea_gps.cnt*]

A few lines from a sample output file are shown in Figure 30. The first three columns supply the time, longitude and latitude information required in subsequent navigation and calibration steps. The remaining columns are quality indications that can be used for manually or automatically assessing the quality of the fix and editing the file.

8.1.4 Editing Fixes

For datasets collected when GPS coverage was inadequate, we usually employ a program that automatically edits the fix file and optionally merges in Transit fixes to supplement the GPS. For more recent datasets, we usually perform a casual, manual editing of the fix file as a first pass. This usually means editing obviously spurious fixes observed from a simple plot of the cruise track (which can range from none, to a few random glitchy fixes, to missing negative signs on all latitudes between 0 and 1°, as can happen with particular types of GPS messages).

For both automated and manual editing, the fix file is not quite finalized until one or more passes through the entire navigation calculation have been done. When the smoothed results show reasonable variation in the reference layer velocity, then the fix file is considered satisfactory. Otherwise, the bad fix regions are identified from the plot and edited out of the fix file, and the entire calculation is redone and the results are replotted to see if the problem regions have been adequately corrected.

Editing of the fix file, both manual and automated, is best done by prepending a `%` symbol in the first column of the line. This makes it easy to restore a fix if it appears that it has been unnecessarily edited out.

```
1) ubprint output for variable: avg_GPS_summary (*.ags)
   or GPS_summary (*.gps) for 72- & 224-user buffer

     DDAY          LON           LAT     NSAT QUAL HDOP ALT
  338.7837616  -149.5695583   -17.5372167   3  1   4    16.00
  338.7872222  -149.5694000   -17.5369250   4  1   1    16.10
       .
       .
       .

2) ubprint output for variable: TRANSIT_summary (*.trs)
                                          E  I     D  F
                                           L  T     D  L
                                            E  E     R  A
     DDAY          LON           LAT         V  R     P  G  DT

  190.8604630  -157.7488861    21.1823694   43  2  0.57  1  18
  190.9175579  -157.5449083    21.0662833   52  2  1.00  1  18
       .
       .
       .

3) nmea_gps output and ubprint output for variable: GPS_summary (*.gps)
   for 128- & 132-byte user buffer
                                        N Q  H  D   E   V
                                         S U  D  D   D   D    A
                                          A A  O  O   O   O    L
     DDAY          LON           LAT       T L  P  P   P   P    T

  191.7958333  -159.8863133    31.3161917  6 1  1  1   1   1  150
  191.7975694  -159.8864850    31.3160817  4 1  1  1   1   1  150
       .
       .
       .

Key to columns:
--------------
DDAY:  decimal day
LON:   longitude
LAT:   latitude
NSAT:  number of satellites
QUAL:  quality flag as part of the GPGGA message, 0 = GPS not available
                                                  1 = GPS available
HDOP:  horizontal dilution of precision
ALT:   altitude
NDOP:  North-south dilution of precision
EDOP:  East-west dilution of precision
VDOP:  vertical dilution of precision

ELEV:  elevation of satellite
ITER:  number of iterations
DDRP:  distance to dead-reckoned position (in nautical miles)
FLAG:  flag (1 if fix accepted by navigator, 0 otherwise)
DT:    time difference in sec between PC and navigator
```

Figure 30: Columnar output of `ubprint` and `nmea_gps` programs.

Below we describe how to use the automated program for editing and merging GPS and Transit fixes when processing older datasets collected using earlier versions of the UH user-exit program (those that generated user buffer sizes of 102 through 132 bytes).

8.1.5 Automated Editing of the Fix File

This program works only with the `.gps` files generated by either `ubprint` or `nmea_gps`, and the `.trs` file generated by `ubprint`. It uses the number of fields in a line to distinguish between Transit and GPS fixes, and accordingly assumes what each field's contents are.

To run the program, the user must first set up the control file. An example, `edfix.cnt`, is copied by the `adcptree` script to the `nav/` directory, and is shown in Appendix A.9. The program functions according to the user-specified parameters.

The `output` parameter gives the filename to use for the output. As a convention, the `.edf` extension is recommended.

The `transit_file` parameter can either be set to `none` if no Transit fix file is available or if the user does not want it merged with the GPS fixes, or to the name of the file containing the Transit fixes. Note that such a file should have all the columns as specified in the `.trs` output file from `ubprint` (see Figure 30 for the column designations).

The parameters `min_elevation`, `max_elevation`, `max_iterations`, `max_dr`, and `time_since_gps` are used for editing the Transit fixes if the `transit_file` has been specified as other than `none`. The `min_elevation` and `max_elevation` parameters define the acceptable range of satellite elevation in degrees. We usually use the values 7 and 70 degrees, respectively. The `max_iterations` specifies the maximum number of iterations in the fix calculation (we use 3). The `max_dr` parameter defines the maximum distance of the fix position to the dead-reckoned position (we use 4 m). The `time_since_gps` criterion is used to edit out Transit fixes that occur too close to a GPS fix (we normally define "too close" as being less than roughly 9 minutes or 0.006 decimal day apart).

The `gps_file` parameter specifies the name of the input GPS fix file. It is followed by the parameters that define the editing criteria for GPS fixes.

The `max_hdop` criterion establishes the maximum acceptable horizontal dilution of precision (we normally use 6).

The `time_since_3` criterion is used to reject two-satellite fixes that occur long after a fix calculated from three or more satellites was available. This is possible with receivers that have a timing device that enable them to estimate the fix from just two satellites. We normally use a value of 0.25 decimal day (6 hours).

Once the control file is set up, the program is run by typing

`edfix` [*edfix.cnt*]

Note that `edfix` just blindly applies the user-specified parameters to the columns in the input files. The user may choose to override `edfix`'s editing at anytime, restoring fixes if the automated editing unnecessarily results in too large a gap (simply by deleting the % symbol prepended by `edfix`) or commenting out more fixes if subsequent navigation calculations show this to be necessary (again, by prepending a % symbol to the appropriate line in the fix file).

8.2 The Navigation Calculation

The navigation calculation is performed once calibration is complete and the fixes have been cleaned. Absolute currents over a fixed depth range (reference layer) are obtained by subtracting the average of the ship velocity relative to a reference layer (i.e. ADCP velocities) from the absolute ship velocity over the ground (from navigation, i.e., GPS). The raw absolute current velocities relative to the reference layer are smoothed to reduce the effects of noise in the position fixes and combined with the navigation data to obtain the best estimates of ship positions and velocities, which are stored into the data base. Thus, absolute currents at any depth can be determined from the final ship navigation data and the relative ADCP measurements. The following sections provide instructions for accomplishing these tasks.

8.2.1 Extracting the Ship Velocity Data

The purpose of this step is to extract data on ship velocity relative to the reference layer from the CODAS database. The program used is `adcpsect` (Appendix A.3; for brevity, the control file structure has been omitted and is presented in Appendix A.1). It is no different from that described in the water track calibration section (Chapter 7) to which the reader is referred for details. The only reason to run it again is that the velocities will have been altered during the rotation.

Upon completion of this step, one should have a `.nav` file—a text file with three columns: profile time in decimal days, and the u- and v-components of ship velocity (in m/s) relative to the reference layer. An example is shown in Figure 31.

8.2.2 Finding Average Ship Velocity Between Fixes

The purpose of this step is to generate initial estimates of the reference layer velocity. Averaged between fixes, this is simply the difference between the velocity of the ship over the ground, as determined from the fix file, and the velocity of the ship relative to the reference layer, as taken from the ADCP profiles.

The program `refabs` is used to perform this calculation. A sample control file is shown in Appendix A.21.

The parameter `fix_file_type` recognizes two formats of the fix file: `simple`, where the first three columns correspond to decimal days, longitude and latitude (as in what has been described previously); and the now rarely used `HIG` format, the

```
%        day   reference_bins
   98.001759   -0.151  -0.087
   98.005231   -0.164  -0.116
   98.008704   -0.156  -0.100
   98.012176   -0.213  -0.080
   98.015648   -0.303  -0.124
   98.019086   -0.411  -0.271
   98.022593   -0.210  -0.083
   98.026065   -0.175  -0.038
   98.029537   -0.215  -0.078
   98.033009   -0.247  -0.084
   98.036481   -0.283  -0.111
   98.039965   -0.283  -0.092
   98.043426   -0.262  -0.116
   98.046875   -0.285  -0.157
   98.050370   -0.253  -0.069
   98.053854    3.075   0.879
   98.057315    4.075   1.246
   98.060787    3.999   1.578
   98.064259    4.008   1.721
     .
     .
     .
   99.998056    5.448  -4.109
```

Figure 31: Sample output file for `adcpsect` navigation option

intricacies of which are not useful to discuss here. The `reference_file` parameter specifies the name of the `.nav` output of `adcpsect`. The `fix_file` parameter specifies the name of the fix file. The `output` specifies the name to use for the output file, which typically is given an extension `.ref`. The `year_base` is used for converting time to decimal days. The `ensemble_length` specifies the length of an ensemble in seconds. Finally, the `gap_tolerance` defines a "gap" as one that exceeds the specified number of seconds, and generates some statistics relating to such gaps. The program is run by typing

`refabs` [*refabs.cnt*]

The program generates a text output file. An example and description of this file is given in Figure 32.

8.2.3 Smoothing the Reference Layer Velocity

The next step is to smooth the initial estimate of the reference layer velocity. The program `smoothr` employs a Blackman window function $w(t)$ of width T:

$$w(t) = 0.42 - 0.5 * cos(2\pi t/T) + 0.08 * cos(4\pi t/T)$$

The choice of filter width depends on the characteristics of the data: the quality of the fixes and the expected amplitude and time scales of the currents being surveyed. As an example, when GPS coverage is good and there are small-scale current features of interest, T might be as short as 15 minutes. In the worst case of poor Transit quality and no GPS, T can be up to 12 hours.

The control file for `smoothr` is shown in Appendix A.26. The `reference_file` specifies the name of the file containing data on ship velocity relative to the reference layer (`adcpsect`'s `.nav` output). The `refabs_output` refers to the output file of the `refabs` program, containing the initial estimates of average reference layer velocity between fixes. The `output` parameter gives the text output filename which typically is given an extension `.sm`. Its root plus extension `.bin` will be used for the filename of the binary version of the same. A file with the same root and extension `.log` will contain log information. The `filter_hwidth` specifies half the filter width T in days. The `min_filter_fraction` is a value between 0 and 1 that indicates how much of the filter area must have data. The `max_gap_ratio` limits gap acceptance to within the specified ratio; a value of 0.05 is suggested. The `max_gap_distance` specifies the maximum gap distance (in meters) over which data will be interpolated. The `max_gap_time` is the maximum gap time (in seconds) over which data will be interpolated. Anything exceeding these gap thresholds will start off a new segment. The suggested values are 2000 meters and 3600 seconds (one hour). The `ensemble_time` is the length of an ensemble in seconds. The `max_speed` specifies the maximum speed of the ship in m/s. The `min_speed` indicates the expected short-term variability in ship speed on- or off-station. The `iterations` parameter specifies how many passes

```
98.0017708  -91.7888   13.0144  -0.060   0.159    4.98   0.00  0.00   0.0
98.0052315  -91.7894   13.0145  -0.043   0.198    5.00   0.00  0.00   0.0
98.0087037  -91.7899   13.0148  -0.108  -0.042    5.02   0.00  0.00   0.0
98.0121875  -91.7908   13.0144   0.019   0.207    4.98   0.00  0.00   0.0
98.0156481  -91.7916   13.0147  -0.140   0.094    5.03   0.00  0.00   0.0
98.0191435  -91.7931   13.0142   0.014   0.006    5.02   0.00  0.00   0.0
98.0226273  -91.7937   13.0140  -0.069   0.103    5.00   0.00  0.00   0.0
98.0260995  -91.7943   13.0142  -0.014   0.096    5.00   0.00  0.00   0.0
98.0295718  -91.7950   13.0142  -0.063  -0.088    5.00   0.00  0.00   0.0
98.0330440  -91.7958   13.0137   0.042   0.135    5.00   0.00  0.00   0.0
98.0365162  -91.7965   13.0138  -0.133   0.028    5.00   0.00  0.00   0.0
98.0399884  -91.7977   13.0136   0.022   0.063    4.95   0.00  0.00   0.0
98.0434259  -91.7983   13.0135  -0.116  -0.010    5.00   0.00  0.00   0.0
98.0468981  -91.7994   13.0130   0.053   0.099    5.05   0.00  0.00   0.0
98.0504051  -91.7999   13.0131  -0.126   0.004    5.00   0.00  0.00   0.0
98.0538773  -91.7917   13.0155   0.042   0.100    5.00   0.00  0.00   0.0
98.0573495  -91.7803   13.0192  -0.076  -0.052    5.00   0.00  0.00   0.0
98.0608218  -91.7695   13.0233  -0.123   0.142    5.00   0.00  0.00   0.0
98.0642940  -91.7588   13.0284   0.042   0.036    5.00   0.00  0.00   0.0
  .
  .
  .
99.9911111  -89.2137   11.7037   0.013  -0.125    5.02   0.00  0.00   0.0
```

Figure 32: Sample output file of `refabs` program. The first column of the output file is the fix time in decimal days. This is followed by the position fix (longitude and latitude in decimal degrees). The next two columns are the u- and v-components of the absolute reference layer velocity in m/s. The sixth column gives the time difference between two consecutive fixes in minutes. The last three columns accumulate the u- and v-components of velocity (in m/s) and the time span of a defined gap.

will be made to optimize positions. On a PC, two passes should do. On a faster machine, a third pass could easily be added. Finally, the `fix_to_dr_limit` is the threshold, in degrees, for printing fix times and deviations.

The program is run by typing

`smoothr` [*smoothr.cnt*]

The program generates two output files, an ASCII version (`.sm`) for the user to examine, and a binary version (`.bin`) for quick loading into MATLAB. Both files contain the smoothed reference layer velocity estimates. An excerpt of the ASCII version is shown in Figure 33.

8.2.4 Reviewing the Reference Layer Velocity Estimates

This step is done by using MATLAB to plot the unsmoothed and smoothed reference layer velocities, together with the position fixes. From these plots, the user can decide whether the smoothing parameters are adequately set and whether the fix file has been sufficiently edited for bad fixes.

To create the plots, MATLAB M-files and two programs have been provided to plot one or more days' worth of data at a time. To use these files, the user must first edit the uppermost section of the file `callrefp.m` as shown in Appendix B.2.

Specify the `input files` as the name of the binary output file from program `smoothr` (`.bin`) for `smoothrfile` and the name of the ASCII output file from program `refabs` (`*.ref`) for `refabsfile`. The user has the option to plot both or either of these files by setting the appropriate switches in the control file. If the `smoothrfile` is to be plotted, the `max_gap_ratio, min_speed, and max_speed` should be set to match the values in the control file `smoothr.cnt`. The PostScript output of the plots will be appended on `outfile`. The next three parameters control the axes. The `dt` defines the time interval to be plotted at a time (the default value of 2 days or less is recommended). The variables `y0` and `dy` define the y-axis (u- and v-components of velocity in m/s) range; the values provided are adequate for most datasets. Finally, the `cruise` value is used for the plot title. The `year` defines the base year.

Once everything is set, the user can start up MATLAB and type

`callrefp`(*t0*)

where *t0* is the decimal day at which to start the plot. If more than `dy`'s worth of data need to be plotted, the user will have to run `callrefp` several times with increasing values of *t0* (one could easily incorporate the `callrefp` invocation into a MATLAB `for` loop to automate this process.

The result of this step is a set of plots based on which the user must decide whether the navigation calculations are satisfactory and the results can thus be stored as part of the CODAS database; whether the smoothing parameters need to be revised, in which case the `smoothr` control file should be edited and the process repeated starting

```
 98.001759   -0.059   0.134   -0.210   0.047    -91.788487    13.014472  0.50
 98.005231   -0.059   0.124   -0.223   0.008    -91.789103    13.014495  0.69
 98.008704   -0.058   0.114   -0.214   0.014    -91.789695    13.014533  0.84
 98.012176   -0.056   0.104   -0.269   0.024    -91.790440    13.014598  0.94
 98.015648   -0.053   0.093   -0.356  -0.031    -91.791425    13.014512  0.98
 98.019086   -0.049   0.081   -0.460  -0.190    -91.792685    13.014002  1.00
 98.022593   -0.044   0.069   -0.254  -0.014    -91.793394    13.013963  1.00
 98.026065   -0.039   0.059   -0.214   0.021    -91.793987    13.014019  1.00
 98.029537   -0.037   0.051   -0.252  -0.027    -91.794683    13.013946  1.00
 98.033009   -0.037   0.047   -0.284  -0.037    -91.795469    13.013844  1.00
 98.036481   -0.039   0.045   -0.322  -0.066    -91.796358    13.013666  1.00
 98.039965   -0.040   0.045   -0.323  -0.047    -91.797255    13.013537  1.00
 98.043426   -0.041   0.044   -0.303  -0.072    -91.798089    13.013344  1.00
 98.046875   -0.041   0.044   -0.326  -0.113    -91.798984    13.013040  1.00
 98.050370   -0.041   0.045   -0.294  -0.024    -91.799801    13.012973  1.00
 98.053854   -0.041   0.046    3.034   0.925    -91.791381    13.015490  1.00
 98.057315   -0.041   0.048    4.034   1.294    -91.780261    13.018987  1.00
 98.060787   -0.041   0.048    3.958   1.626    -91.769315    13.023396  1.00
 98.064259   -0.039   0.045    3.969   1.766    -91.758339    13.028184  1.00
   .
   .
   .
 99.998056   -0.057  -0.140    5.391  -4.249    -89.184177    11.680669  0.31
```

Figure 33: Sample `smoothr` output file. The first column is the ADCP ensemble time in decimal days, followed by two columns of the zonal and meridional reference layer velocity (m/s) and two columns of the zonal and meridional ship velocity (m/s). The sixth and seventh columns are the profile positions represented by longitude and latitude, respectively, in decimal degrees. The last column denotes filter fraction which is the fraction of filter area with data.

from Section 8.2.3 above; whether the fix file needs further editing, in which case the user must edit the fix file and rerun both `refabs` and `smoothr`.

A set of plots is shown in Figure 34. In this instance, the smoothed reference layer velocities look fine, so no further editing/iterations are needed.

Once satisfactory, the plots that result from this step also serve to document important aspects of the particular ADCP dataset: spatial coverage, quantity and quality of position fixes, and the degree to which major current features have been resolved by the smoothed reference layer estimate.

8.2.5 Storing the Reference Layer Velocities

This step describes how to store in the CODAS database the smoothed reference layer velocities, the final ship positions for each ensemble, and the average speed of the ship over each ensemble. The program used is known as `putnav` and a sample control file is shown in Appendix A.20.

The `dbname` specifies the name of the destination database to store the data into. The `position_file` specifies the name of the smoothed reference layer velocities input file (output from `smoothr`). The `year_base` is used to convert from decimal days to profile times. The `tolerance` specifies the maximum allowed discrepancy between the fix time and the ADCP profile time. The `navigation_sources` is followed by a list of sources for the fix data terminated by `end`. This information is recorded in the database as a set of bit flags.

The program is then executed by typing

`putnav` [*putnav.cnt*]

During runtime, the warning `BAD DATA AT TIME` :... may appear several times, indicating profile times for which gaps in the ADCP reference layer velocity data or extended periods without good navigation make it unreliable to derive profile positions and absolute velocities.

The effect of `putnav` can be verified using `showdb`. The user can type

`showdb` [[*path*]*dbname*]

where *dbname* is the five-character name under which the database was loaded (preceded by the path to the location of the block files if not in the current directory) and select 16 for GET DATA. The ship velocity stored in variable 39 (ACCESS VARIABLES) and the profile positions are printed in the header section of each screenful of data.

9 Data Extraction and Analysis

CODAS provides various utilities for data extraction and analysis. The primary tool for pulling current velocities from the database is program `adcpsect`. This

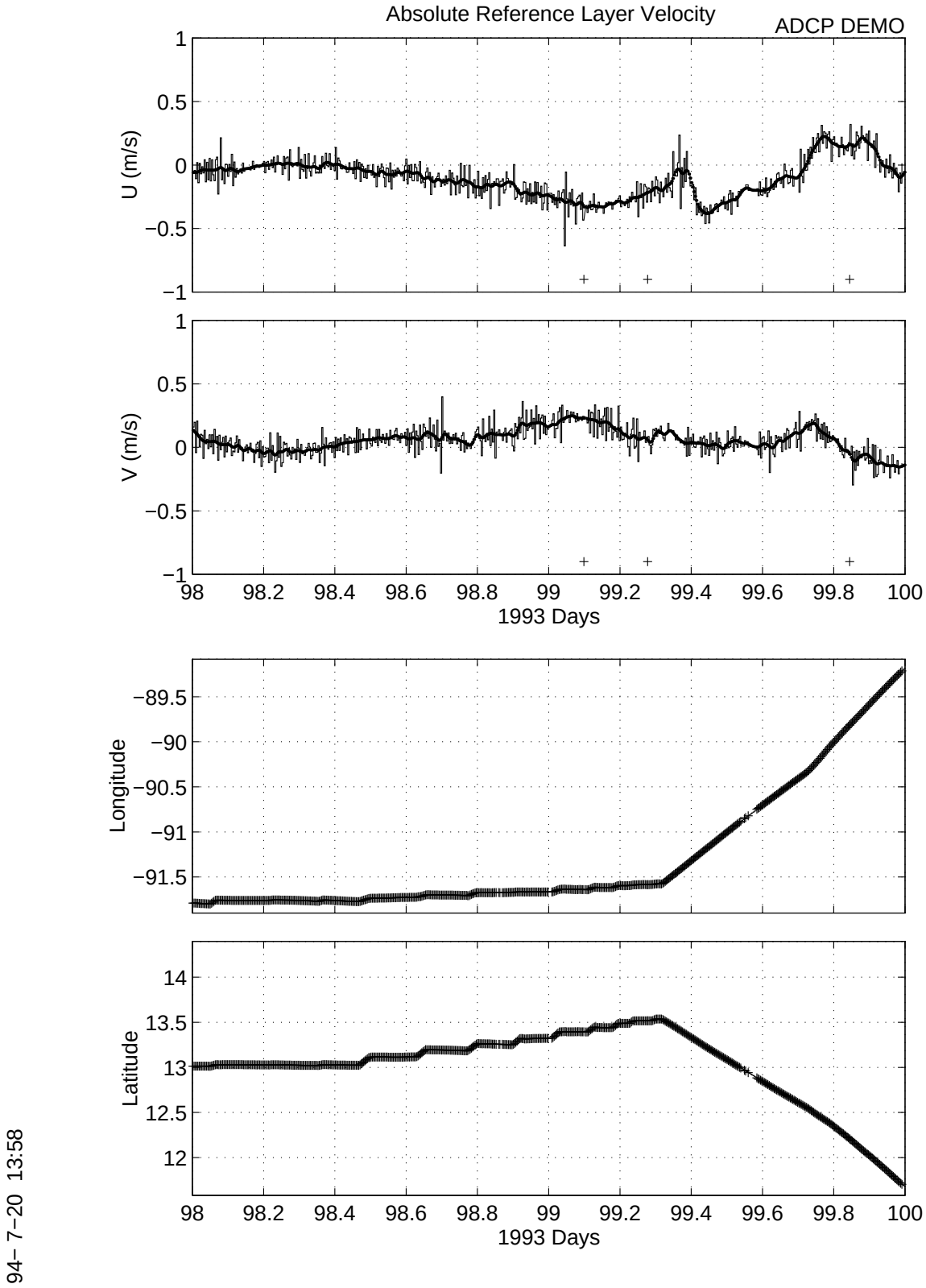


Figure 34: Reference layer velocity plots generated by `callrefp.m`.

program will be the focus of greatest attention in this manual. One of the important applications of `adcpsect` is preparing input files to two common plotting routines: `vector` and `contour`. These application programs will also be described below. A secondary extraction utility, `profstat`, facilitates access to the ancillary data, such as echo amplitude and percent good. This program has other useful features that will be explained.

We begin with a brief review of important terms from previous chapters and an introduction to some of the commonly used terms associated with the various extraction and graphical tools.

block and directory files A CODAS database consists of two types of binary files: block files and a directory file. CODAS is a hierarchial system in which a directory file summarizes the contents of each block file. Typically the directory file and its associated block files pertain to a single cruise, although blocks from several cruises may be combined under a single directory file if the cruises do not overlap in time. The block files hold current velocity profiles, ancillary measurements, quality and processing status flags, and verbal metadata on cruise notes and the processing, calibration, and navigation steps.

profile This refers to the recorded ensemble-average, usually at 5-minute intervals, of single-ping profiles. The profiles are partitioned in the vertical into equally-spaced depth bins. Binned-arrays of various parameters such as U-, V-, and W-components of relative currents, echo amplitude, etc. are stored in the block files.

control file Most CODAS programs are executed in conjunction with a control file which sets user-defined options. A control file is an ASCII text file containing explanatory comments and the keyword/entry pairs that the user sets. Most CODAS software is not interactive. The basic procedure for using the CODAS tools is to modify (with a text editor) a copy of an existing control file, which is provided with the software, and then to execute the program associated with the control file.

The following are critical CODAS utilities.

mkblkdir Creates a CODAS database from a set of block files and, if necessary, performs binary format conversions. The NODC stores the CODAS block files in a SUN/SGI compatible binary format. If the user's host machine is different, then `mkblkdir` will perform the binary format conversion. This utility can also be used to change the name of the database or to merge data block files from several cruises into a single database, assuming no overlap in time occurs.

showdb This interactive utility is used to browse through a CODAS database. Once the directory and block files have been made local to your machine, this tool

will verify that the format conversion was successful and will give a quick view of which parameters are available.

adcpsect This is the primary extraction tool for the current data. It allows a variety of averaging and regridding capabilities as well as selective extraction based on data quality criteria.

profstat This is a statistical tool for data extraction as well as a means of obtaining ancillary parameters stored as binned-arrays such as echo amplitude and percent good.

lst_temp This utility obtains the transducer temperature.

lstblock Obtain time range and number of profiles per block.

lst_prof Lists profile times and positions for a given time range.

asc_dump This tool performs a complete ASCII dump of CODAS block files. It expands the block file by a factor of 5 in size. It will be used mostly by the NODC to provide data sets to users with machines upon which CODAS has not yet been made compatible.

Time is the primary access key of the `adcpsect` and the `profstat` programs. To facilitate the input of time domains into these programs, several supplementary tools are provided:

llgrid A program to generate a text file of time ranges that roughly correspond to longitude/latitude grid crossings. The grid is specified as a starting position and an increment for each dimension. For instance, if one wants to generate a vector plot of the near-surface currents along the cruise track with one vector representing each 1°latitude/longitude box, then `llgrid` would generate a list of time ranges of each box for input to the `adcpsect` program.

timegrid A program to break up a given time range into smaller time ranges of a specified interval. For instance, one can break a cruise up into hourly intervals.

arrdep A program to generate a text file of time ranges for profiles that occurred while the ship was on-station or underway. This is useful in preparation of performing statistical analysis of on-station versus underway data using `profstat`.

These CODAS programs have common usage: 1) modify the sample control file pertinent to the given program for pointing to input and output files and for setting your options and 2) run the program (referred to as the CODAS execution fashion) by entering

```
program-name control-file-name
```

The output files of the extraction programs are in ASCII format. `adcpsect` and `profstat` also generate Matlab matfiles with essentially the same information as the ASCII file, for those users who prefer to use Matlab for plotting and data manipulation. Matlab is available for personal computers and workstations. It provides easy plotting and interactive calculations (see Section 1 for an address of the company). We highly recommend the use of Matlab.

9.1 Preparing Time Ranges using CODAS Utilities

Time is the primary key for accessing the database. The control files require time ranges for the desired data extractions. These time ranges can be entered by hand, cut and pasted from other control files or from block directory headers as seen with `showdb`, or prepared using supplementary programs such as `llgrid`, `timegrid`, or `arrdep`. These programs in turn require the use of control files as described below. The output of these programs consist of files containing time ranges that are subsequently appended to the control files of the data extraction utilities, `adcpsect` and `profstat`, which can produce averaged velocities or statistics for each time range separately or can combine all time ranges to obtain a single average or set of statistics. The user is typically doing some averaging while extracting the data. However, if the user desires every ensemble, then time ranges corresponding to each ensemble can be prepared as explained below in Section 9.1.2.

9.1.1 Converting Spatial Domains into Time Ranges: llgrid

The extraction tool `adcpsect` allows averaging and regridding in depth and space. However, the program can not explicitly access the database by spatial domains. Thus, time ranges are defined which correspond to the period during which the ship was within the area of interest. `llgrid` translates a user-specified longitude-latitude grid, which is specified as a starting position and an increment for each dimension, into the corresponding time range.

First, obtain a copy of the control file, `llgrid.cnt` (Appendix A.12). We recommend renaming the copy to a name related to the chore; for example, `llg_10.cnt` for a control file set to increment by 10ths of a degree. The control file is edited to set the options depicted in Figure 35.

The keywords `dbname:` and `output:` represent the input and output respectively. For the former, prefix the name of the database with the path to its location if the program is executed from a directory other than where the database resides. For the latter, provide a filename that makes sense relative to the chore. In many cases, especially in preparation of the vector or contour plots, one may iterate between the programs `llgrid` (define time ranges), `adcpsect` (extract, average, and regrid), and `vector` or `contour` until the desired spacing of the data points is obtained. Thus to keep track of the various versions it is up to the user to be creative with the naming of the intermediate files.

```
      CONTROL FILE STRUCTURE:  LLGRID.CNT

dbname:       < CODAS database name >
output:       < output filename >
step_size:    < profile sampling rate > { 1 = every profile }
latitude
  origin:     < decimal degrees >
  increment:  < decimal degrees >
longitude
  origin:     < decimal degrees >
  increment:  < decimal degrees >
time_ranges:  { list of YMDHMS time pairs }
  < yy/mm/dd hh:mm:ss > to < yy/mm/dd hh:mm:ss >
```

Figure 35: Leading comments in the `llgrid.cnt` control file

The `step_size:` keyword pertains to the density of profile ensemble times to be considered. A `step_size:` set to 1 denotes every profile to be taken while setting it to N denotes every Nth profile time to be recognized.

The `latitude` and `longitude` have two options which must be set: `origin:` and `increment:`. The former refers to where the output data will be centered in relation to the grid boundaries. The latter refers to the spatial interval between grid points. The `origin:` is usually set to the negative of half the `increment:`, so the output data will be centered on the grid boundaries. For large spatial domains to be analyzed or plotted, the `increment` is typically increased to spread out the grid points, while for examining fine detail within a smaller area, `increment` would be decreased.

The `latitude` and `longitude` parameters are both set if the cruise track is neither entirely zonal or meridional, such as in preparation of vector plots. If one prefers to analyze the data along zonal or meridional sections, such as in preparation for contour plots, then the axis that remains roughly constant is deactivated by setting the `increment:` parameter to a large value, such as 10000.

Finally, the `time_ranges:` refers to the time range(s) within the desired database. One convenient method of obtaining the time range is to use `showdb` for viewing the block directory header (Figure 10). Note that the format for year within `time_ranges:` is two-character (e.g. 93 for 1993).

Once the control file is modified, run the program in the CODAS execution fashion.

9.1.2 Partitioning to Equally-spaced Time Ranges: timegrid

This program simply breaks the input time range into smaller time ranges of a specified interval, such as hourly. A sample control file is shown in Appendix A.28 with

the keywords shown in Figure 36.

```
        CONTROL FILE STRUCTURE:  TIMEGRID.CNT

output:          < output filename >
time_interval:   < in minutes >
time_range:
  < yy/mm/dd hh:mm:ss > to < yy/mm/dd hh:mm:ss >
```

Figure 36: Leading comments in the `timegrid.cnt` control file

It does not require an input file. An `output:` keyword must be set to the name of the desired output file. The `time_interval:` keyword is set to the interval in minutes of the resultant equally-spaced time ranges, e.g. 60 for hourly. The `time_range:` is the range that will be partitioned. If one is subsequently planning to use `adcpsect` to perform an average of each time range, then the user may want to set the `time_range:` field in `timegrid.cnt` such that the average falls on a standard point, such as on the hour. In this case, the beginning and ending times `time_range:` field in `timegrid.cnt` would be set to 30-minute increments such as

93/02/01 19:30:00 to 93/02/28 05:30:00.

Once the control file is modified, run the program in the CODAS execution fashion.

9.1.3 Creating Special Time Ranges: arrdep

For diagnostic analysis, it is helpful to compare the velocities measured by the ADCP while the ship was on-station (quasi-stationary) versus underway. The program `arrdep` finds the time ranges for use in control files of programs such as `profstat`.

Two sample control files are provided with the package: `arrdepos.cnt` and `arrdepuw.cnt`, for the cases of on-station and underway, respectively, both of which should be separately modified and executed with program `arrdep`. Examples are shown in Appendix A.6. The parameters are shown in Figure 37.

The keyword `reference_file:` refers to the input file, which is a navigation file (velocity of the ship relative to a reference layer at each ensemble) created by program `adcpsect` as described in Section 9.2. The keywords `output_file:` and `year_base=` are adequately explained in Figure 37.

The `nrefs=` refers to the width of the window used in the calculation while the `i_ref_xx=` refer to the indices `xx` that are used to determine if the ship is steaming or on-station. The default settings in the demo control files are usually adequate. The `range:` parameter is set to `on_station` or `underway`. The other options listed within

```
        CONTROL FILE STRUCTURE:  ARRDEP.CNT

reference_file: < input file of ship velocity relative to reference layer >
output_file:    < output filename >

year_base=      < base year for decimal days >

n_refs=    <13>  { number of ensembles to use in calculation }
i_ref_l0=   <6>  { index of first ensemble used before change in velocity }
i_ref_l1=   <6>  { index of last ensemble used before change in velocity }
i_ref_r0=   <8>  { index of first ensemble used after change in velocity }
i_ref_r1=   <8>  { index of last ensemble used after change in velocity }

range:    < on_station | underway | all | whole_station | whole_underway >
up_thresh=    <2.5> { m/s, for jump detection, about 1/2 underway speed }
down_thresh=  <2.5> { m/s, for jump detection, about 1/2 underway speed }

margin=       <10>  { seconds to subtract from first time, add to second,
                      to allow for rounding errors and ensure that the
                      time range brackets the ensemble times }
```

Figure 37: Leading comments in the `arrdep.cnt` control file

`range`: are special cases that are rarely used. The `up_thresh=` and `down_thresh=` parameters are normally set to 2.5 and refer to the threshold (in m/s) for detecting jumps in ship velocity. Finally, the `margin=` is explained in Figure 37.

Once the control file is modified, run the program in the CODAS execution fashion.

9.2 Primary Extraction Tool: adcpsect

Many analysis and plotting steps involve a single multi-purpose program called `adcpsect` (from `ADCP sect`ions). It is primarily used to obtain either ship or current velocities. For access to ancillary parameters, see program `profstat` (Section 9.3) and special utilities such as `lst_temp` (Section 9.4). As with most CODAS programs, it is driven by a control file, `adcpsect.cnt`, in which the operator sets the desired output options. The primary key for extracting data from CODAS is time; the easiest method for preparing the desired time ranges is through application of programs `llgrid` and `timegrid` as discussed in Section 9.1.

The user has a variety of options for averaging and re-gridding in all dimensions (x,y,z,t). Output files for the many possible user-defined combinations of time, longitude, latitude, and u- and v-components (absolute, relative, or ship velocity) are available as ASCII flat files and as matlab-formatted (binary) files. The output also includes an ASCII file of statistics. The multitude of options and output files which are controlled by `adcpsect.cnt` are described below.

The recommended approach to preparing a control file for `adcpsect` is

1. Find a sample control file that closely resembles the task at hand. A variety of examples are shown in Appendix A.1 and loaded with the software with naming convention `as_xxx.cnt`, where xxx can be nav (navigation), vec (vector), con (contour), sub (standard ascii subset), and trn (transport).

2. Rename the working control file to a name that reflects the task. In that way, one will build more sample control files that are readily identifiable for future use.

3. Using the examples at hand and the explanations below, set the parameters to your specifications.

4. Prepare the time ranges with `llgrid` or `timegrid` and append these to the `adcpsect.cnt`.

We now describe the various options within the control file.

9.2.1 Preliminary Parameters

The control file starts with the mandatory parameters (Figure 38). Many of these are common to other CODAS control files and have been discussed.

```
dbname:       < CODAS database name >
output:       < output filename >
step_size:    < number of profiles to advance >
ndepth:       < number of depth bins to read and process >
time_ranges:  < combined or separate >
year_base=    < base year for decimal days >
```

Figure 38: Leading comments in `adcpsect` control file: mandatory parameters

- `dbname`: The data base name with path if necessary. The path can be absolute or relative to the directory from which `adcpsect` is ran.
- `output`: Specifies the output file name(s), without extension. The extension for the output files is added by `adcpsect`, according to the content of the output file (Figure 39). The types of output files created depend on various options set in the control file. For example, it can extract velocities for distance versus depth sections, such as equatorial transects, that can be plotted as contours (output file with `.con` file extension as set with the `contour` option). Or longitude, latitude, U and V sections can be extracted for plotting vectors (`.vec` file as set with the `vector` option). Another output option is to extract ship velocity relative to a reference layer that creates output with a `.nav` extension. The `.nav`, `.con`, and `.vec` are the three most commonly used ASCII output files. More details on these output options and other output options are discussed below.
- `step_size`: 1 means you consider every profile, 2 for every other profile, and so on. This parameter, usually left at 1, may be set to large values if you want to sample a large dataset with minimal program execution time.
- `ndepth`: Specifies the maximum number of bins to be considered by `adcpsect`. If you know that you will use a limited bin range, say 5–20, you could save running time by specifying `ndepth: 20`.
- `time_ranges`: Time is currently the primary access key to the CODAS database. Thus you specify which profiles you want to work with by listing (at the end of the `adcpsect` control file) one or more time ranges that these profiles fall into. These time ranges can be entered by hand, cut and pasted from other control files, or created by `llgrid` and `timegrid` as previously discussed (Section 9.1). In the case of several time ranges, you might want to calculate, for example, basic statistics for all profiles within these time ranges `combined`,

FILE EXT.	FORMAT	APPLICATION EXAMPLE	COMMENT
.nav	ASCII	calibration,	ship velocity (negative of ADCP velocities) as u-, v-components versus time; the user sets parameters in the control file to relate (or not) the velocites to either a reference layer, the bottom, etc.
.con	ASCII	contour or stick plots	x-axis as time, latitude, or longitude; y-axis as depth; u- and v-components of current velocity, which can be absolute, relative, or other as set by parameters in the control file
.vec	ASCII	vector plots	longitude, latitude, then one or more pairs for each layer in depth grid: u- and v-components of velocity, which can be absolute, relative, or other as set by parameters in the control file
.sub	ASCII	subset	same as ".vec" above except includes time, transducer temperature, and ship velocity
.sta	ASCII	statistics	velocity component statistics such as the mean, variance, extrema, normalized extrema, and the bins at which the extrema occurred
.trn	ASCII	transport	transport streamfunction along cruise track; for each depth range, the along track integral of udy, -vdx, their sum, and the vertical integral of each of these three
_mtr.mat	matlab	transport	matlab version of above
_muv.mat	matlab	numerous applications	velocity components
_mxy.mat	matlab	numerous applications	position and time arrays

Figure 39: Various file types created by `adcpsect`. EXT denotes file extension.

or calculate statistics for each time range `separately`. These two cases are distinguished by this parameter. The option `separate` would then give you several sets of statistics. The `combined` option is rarely used with `adcpsect` but often used with `profstat`.

- `year_base`: In most of the output files, time is given in decimal days, which is equivalent to the Julian day less 1. This parameter defines the origin for the decimal day (noon on January 1 of the `year_base` year will be day 0.5).

9.2.2 Optional Parameters

The following discussion describes the options which allow the user to selectively extract data based on data quality or other criteria and the options which specify exactly which parameters are extracted, how they are to be averaged and gridded, and if any special calculations must be made. In most cases, many of the options are left out. Some options contain embedded options in which one may choose one or several parameters. The user is referred to sample control files in Appendix A.1 for examples. We begin with a brief overview of the various option types.

Overview

- **Selective Criteria–** These options allow one to selectively extract data based on quality flags or other specific criteria.
  - `underway`: Allows selection based on ship speed.
  - `pg_min`: Selection based on percentage good of pings per ensemble.
  - `AGC_margin`: To limit noise bias in older data sets, reject data with AGC levels within this range of the minimum AGC. Profile penetration is cut off to limit noise bias based on where the AGC level falls to a threshold above the noise floor.
  - `skew_limit`: This feature is used only with older data sets. It is designed to limit the skew bias.
  - `flag_mask`: Selection based on flags set during editing and processing.
- **Data Adjustments and Reduction–** These parameters allow for choosing velocities types (relative versus absolute), for averaging and regridding in space and time, and for rotating the coordinate axis.
  - `reference`: The velocity components are stored in the database relative to the ADCP. During extraction, these can be obtained as shear profiles or adjusted to absolute profiles based on a variety of referencing schemes.

  - `ctd`: Adjust depths for sound speed profile based on CTD data (if available).
  - `regrid`: Various regridding and averaging schemes in space and time. Allow for the vertical axis to be a quantity other than depth, such as potential density.
  - `rotate`: Rotate the axes of the velocity components; for instance, one may want the V-component along a major current (such as the Gulf Stream off Hatteras toward the NE) with the U-component orthogonal to the flow.

- **Specific Products**– Various options cater to special applications.
  - `navigation`: The ships speed can be obtained from the relative ADCP velocities. This option allows for a variety of output styles based on several referencing schemes for determining the ship speed.
  - `contour`: With either time, latitude, or longitude as the abscissa and depth or density as the ordinate, contours of the current velocities can be prepared. This option facilitates the use of the `contour` program as well as providing output that is useful for other applications.
  - `vector`: With longitude as the abscissa and latitude as the ordinate, the current components are added at each point for a given depth to produce vectors, which can be plotted using the CODAS program `vector`. This is an excellent way of looking at the synoptic features.
  - `transport`: Transport stream function along the cruise track can be calculated. Statistics are also created for each depth so that an assessment can be made of the quality.
  - `ascii`: This option was set up to produce the NODC standard subset that contains output columns for time, longitude, latitude, and U- and V-component pairs for specified depths and temporal sampling.
  - `statistics`: A variety of statistical parameters can be determined.
  - `sequential_cor`: This was a special feature used only for detection of big compass jump errors.

Selective Criteria

CODAS maintains the original data set as collected by the instrument on the ship. During the editing stage, flags are set if data are suspicious or if the bottom is encountered, but the original data are left intact. As data are extracted from the

CODAS database, the user can specify whether to recognize the flags or not. These and other options for selective extraction are discussed below.

- `verbose`: prints lots of debugging and auxiliary output
- `underway`: the choices `yes` (`no`) select only those profiles for which the ship speed exceeds (is less than) the velocity threshold `limit` in m/s. Omit this option if you want to use both underway and on-station profiles.
- Additional editing can be done as the data are extracted. Four editing options are provided by the following:

  1. `pg_min`: The simplest is rejection of all depth bins for which the percent good pings falls below some threshold, for instance, 30%. For data collected with firmware versions 16.26 and later, we usually apply only the percent good editing using a threshold of 30%, and let the `flag_mask` default to `ALL_BITS`.
  2. `flag_mask`: This specifies one or more bit-masks that cumulatively specify which bits in the CODAS database variable `PROFILE_FLAGS` to check. The variable `PROFILE_FLAGS` is an array of bytes, one byte per profile bin, the individual bits of which are set according to whether the data for that bin pass certain bin editing criteria. The bits are set during the editing stage of ADCP processing. For example, bins of profiles that were contaminated by CTD wire interference during data acquisition have the `GLITCH_BIT` set. During the `adcpsect` run, the user can select whether only certain editing criteria will be applied to the data upon extraction; by default, all such editing criteria are applied (i.e., the default bit-mask is `ALL_BITS`).

     The next two editing options are based on the analysis of deep bias [2], to which the reader is referred for terminology and details. Normally use only for data collected with firmware earlier than version 16.26.
  3. `AGC_margin`: To limit noise bias, the profile is cut off below the depth where the AGC level falls to a threshold above the noise floor. The latter is defined as the average AGC level of the bottom few bins. A typical threshold is 7 counts (where one count is approximately 0.45 dB).
  4. `skew_limit`: Skew bias is limited by cutting off the profile below the point where the velocity has changed by more than a threshold (typically 50 cm/s) from its value at a shallower threshold AGC level (100 counts, corresponding to roughly 130–150 m) at which the signal processing filter in the profiler ceased to track the signal.

Data Adjustments and Reduction

The following parameters allow various adjustments of the data in terms of reference frames, grid patterns, and temporal averaging.

- `reference`: ADCP profiles are stored in CODAS as profiles of velocity relative to the transducer. They are converted into absolute velocities as needed when plotting and analyzing the data. (Note that only ONE way of referencing is possible). The options as they appear in the control file are shown in Figure 40.

```
[ reference:       { choose only one of the following: }
    < reference_bins <5> to <20> |
      none |
      final_ship_ref |
      nav_position |
      nav_velocity |
      bottom_track ]
```

Figure 40: Options for the `reference:` parameter.

Several reference methods are available:

1. `reference_bins`: Each profile is referenced to the mean velocity over the specified bin range. A bin range over which good data are available is normally selected, such as bins 5 - 20 (for 8 m depth bins representing the layer from roughly 50 to 170 m.
2. `final_ship_ref`: The ship speed calculated from combined ADCP and navigation data. We consider this our best estimate of the ship speed.
3. `nav_position`: In this and in the following option, we use navigation data that were recorded together with the raw ping data in the pingdata file (variables loran.lat, loran.lon, loran.vel and loran.dir in the navigation structure). For `nav_position`, the reference velocity is calculated by differencing the position data. Note this option is rarely ever used and only makes sense if navigation positions were recorded at the end of the ensembles.
4. `nav_velocity`: Uses loran.vel and loran.dir for referencing. It is rarely used.
5. `bottom_track`: Uses the recorded bottom track data.

- `rotate`: Allows for rotation of the velocity components. There are two possibilities: (1) the data may be rotated by a specified angle (the sign convention is such that the the coordinate system is rotated counterclockwise by the specified

amount); (2) the data are rotated into ship's coordinates, with the zonal (x) direction becoming starboard, the meridional (y) direction becoming forward. For example, this would facilitate analysis of the Gulf Stream off Hatteras if the ship is steaming with the current and the y-axis points in the forward direction.

The navigation output (the `.nav` file) is not affected by the rotate parameter.

- `ctd: dbname`: In this option, the default ADCP bin depths calculated from the default sound speed (1470 m/s) will be recalculated based on the in situ sound speed profile. The sound speed is calculated from the specified CTD database; the CTD profile on or after (in time) to the current ADCP profile is selected. `dbname`: refers to the name of the CODAS database of CTD data. A separate manual for loading CTD data into a CODAS database is available upon request from the authors.

  A CTD database must also be specified if any vertical regridding based on density intervals is selected (see below).

- `regrid`: Refers to regridding the ADCP profile in the vertical. This option contains many suboptions as show in Figure 41. Choose one option from each suboption delimited by greater-than and less-than signs (`<..>`).

```
[ regrid:
    < integrate | interpolate | centered_average | average |
      per_grid_interval >
    < depth | svan | pot_svan | sigma_theta | sigma_t >
      { one of the last 4 only if the ctd: option has been selected }
    < grid      number= <50> origin= <16> increment= <8> |
      grid_list number= < 5> boundaries: <20> <40> <60> <80> <100> > ]
```

Figure 41: Options within the `regrid:` parameter.

1. **statistical method for regridding in the vertical**

   `< integrate | interpolate | centered_average |`
   `average | per_grid_interval >`

   Several of the regridding schemes are based on vertical integration, such as `integrate`, `average`, `centered_average`, and `per_grid_interval`. For the last three options, the integral is divided by a suitable length. For averaging, this length is always the depth interval over which the average was taken. For `per_grid_interval`, the length is the grid variable interval. If the variable is depth, then this is the same as the average. If the variable is other than depth (i.e. one of the density options) `per_grid_interval`

divides the depth integral by the grid variable interval so the result is the transport per unit width with respect to the grid variable.

`average`, `per_grid_interval`, `integrate`, and `integrate_top` all start by integration from one grid point to the next. For `integrate_top`, the first grid point is set to 0m, so it extrapolates to the surface. `average` then divides this depth integral by the depth interval over which the average was taken.

`centered_average` and `interpolate` put the grid point in the middle of the increment while the rest average over the increment. `centered_average` is the average value in a region surrounding the grid point. `interpolate` is simple linear interpolation from the original grid to the user-specified grid.

Examples of `regrid:` options can be seen as used with `contour` (Appendix A.2) and with the `vector` (Appendix A.5) output options.

2. **vertical-variable**
   `< depth | svan | pot_svan | sigma_theta | sigma_t >`

   The `depth` is the bin depth array data. One of the last four options can only be chosen if the `ctd:` option was applied.

3. **vertical regridding scheme**

```
< grid      number= <50> origin= <16> increment= <8> |
grid_list number= < 5> boundaries: <20> <40> <60> <80> <100> >
```

   You specify the new vertical grid either by the three values `grid number`, `origin`, and `increment`, or by enumerating the `boundaries` and specifying how many as `grid_list number`.

Specific Products

The choice of output parameters from the database (time, latitude, longitude, depths, velocity components) varies depending on the application in mind. Also, the selection of the averaging and regridding techniques dictate different styles of presenting the output. Moreover, special products such as statistics can be produced. This set of options pertains to the various styles of output files as tabularized in Figure 38.

- `navigation`: This option produces an output file with the extension `.nav` used (for example) in the calculation of ship's speed. It is mostly geared toward the processing navigation step during which the final ship's speed and position at each ensemble point are determined. A list of the options is shown in Figure 42.

```
[ navigation:      { output in .nav text file }
                   { choose one or more of the ff. then 'end' }
    [reference_bins <5> to <20>] {bin range; first bin is 1}
    [final_ship_ref]
    [nav_position]
    [nav_velocity]
    [bottom_track]
    end ]
```

Figure 42: Options within the `navigation:` parameter.

The different output file versions are:

1. `reference_bins`: under this option, the three columns of the output file contain time in decimal days for each profile, and the zonal and meridional velocity (relative to the transducer) averaged over the specified bin range. Thus, this file gives the difference between absolute current and ship speed. The velocity vector is multiplied by -1, so that it shows the ship's motion relative to the reference layer rather than vice versa.
2. `final_ship_ref`: here, the second and third column show the final estimate of zonal and meridional ship speed, respectively. Obviously, this output can be obtained only *after* the navigation calculations have been completed.
3. `nav_position`: lists a velocity based on differencing the loran.lat and loran.lon from the navigation structure; see also `reference` above.
4. `nav_velocity`: lists zonal and meridional velocity converted from loran.vel and loran.dir; see also `reference` above. The output under `nav_velocity` is similar but not identical to that from `nav_position`. The input to the navigation structure is generated by the user-exit program.
5. `bottom_track`: outputs the recorded bottom track velocity. This option can be used for the bottom track calibration, but has recently been superceded by the program `lst_btrk` which provides additional information on the bottom depth.

- `contour`: Generates ASCII output files that can be used by graphics programs to generate velocity contour plots. A list of the options is shown in Figure 43.

  ADCP profiles are averaged over each time range, and output in four columns to a file with extension `.con`: the first column contains the horizontal coordinate, which can be either latitude or longitude or time, followed by the vertical coordinate, the averaged zonal and the averaged meridional velocity. Specifying

```
[ contour:          { output in .con text file }
    < latitude | longitude | time >
    < middle | mean >
    minimum_npts= <10>
    units=        <0.01> ]
```

Figure 43: Options within the `contour:` parameter.

`units= 0.01` provides cm/s. The `minimum_npts` limits acceptance to average velocities to those with the specified minimum number of points within the time range. Choose this value such that you eliminate cases where the average would be based on a small fraction of the theoretically possible number of profiles in a time ranges. `middle` and `mean` distinguish whether the horizontal coordinate is defined as the middle position between the two end profiles of the time range, or as the mean position of all profiles within the time range. They would differ if the profiles are not evenly distributed in space.

Note: the `ascii:` option allows similar output except it includes simultaneous output of longitude, latitude, and time, instead of having to select just one of these under each run.

- `vector:` Similar to `contour:`, but here the ASCII output is used to generate vector plots. The output columns are longitude, latitude, and one or more pairs of u- and v-velocity components. The `minimum_npts` is the same as explained for `contour:`

- `transport:` Generates output of a transport streamfunction along the cruise track. A list of the options is shown in Figure 44.

```
[ transport: { output in .trn text file and _tr.mat Matlab Mat-file }
    minimum_npts= <10>
    units=        <1.e6>                          { for Sverdrups }
    direction=    < all | left | right | along > ]
```

Figure 44: Options within the `transport:` parameter.

Again, the ADCP profiles within the specified time ranges are first averaged, and the result is integrated trapezoidally in the horizontal to generate the stream-

function (note that the integration needs at least two points; thus there is no transport output for the first time range). Transport is defined as positive to the right of the cruise track, namely as $udy - vdx$.

Four output files are generated: one, `.trn`, is an ASCII file listing for each depth range the integral (along the cruise track) of udy, $-vdx$, their sum, and the vertical sum of these three. The other three are Matlab Mat-files with the last four characters of the root file name as `_mxy`, `_muv`, and `_mtr`, and with extension `mat` listing the same information in more compact format for easy plotting with Matlab. The first gives position and time arrays, the second the averaged velocities, and the last transport. The statistics option (see below) generates an output file `.sta`. Within the `.trn` and the `.sta`, a field is given that lists how many time ranges were used for the integration at each depth range. This number might be smaller for the deepest depth intervals used, indicating that there were gaps in the integration. Such gaps would not be readily apparent in the transport integral itself, so it is strongly advised to check the statistics file! Note, there is a lack of interpolation across gaps.

`minimum_npts` are defined as before; specifying `units= 1.e6` provide transport in Sverdrup. Specifying `direction` as `left` will consider only flow toward the left of the cruise track; `right` only that toward the right. For example, if the ship went south to north, then the `right` option would give the cumulative eastward transport without subtracting any westward flow. Selecting `all` would show the net transport.

- `ascii`: This parameter creates an ascii file with extension `.asc` that contains a column for time, longitude, latitude, and pairs of U- and V-components. A `minimum_percent=` parameter must be set (default is `.50`) which denotes the percentage of ensembles in each time range with good data. Normally one would use the vertical regridding options (see `regrid`: above) with the `origin=` set to the shallowest good bin, the `increment=` set to your liking, and `grid number=` set to the number of bins such that one does not penetrate the depths where the percent good drops below some assumed limit, such as 50 %.

- `sequential_cor`: This was a special feature used only for detection of big compass jump errors. The output appears similar to that from the navigation options: three ASCII columns showing profile time, correlation with the previous time after rotating the profile to provide optimum correlation, and the correlation angle used. For large correlations, the rotation angle should be small under normal conditions; a large rotation angle is only significant if the correlation itself is high. Choose a bin range over which the data are expected to be fine (e.g., 5–20).

- `statistics` provides ASCII output of statistics for each time range in a file with extension `.sta`. They include mean, variance, extrema, normalized extrema,

and the bins at which the extrema occurred. Selecting `all` gives all possible output. Units of 0.01 will again list results in cm/s.

Note: The current version of `adcpsect` produces a `.sta` file whether or not the user specifies the `statistics` option. A planned revision will make sure that the `.sta` file, as well as the matfiles, will be produced only if requested by the user.

9.3 Secondary CODAS Data Access Tool: profstat

The `profstat` program is similar to `adcpsect` in many respects: it extracts data from the CODAS database with time as the primary access key, provides statistical summaries, and allows selective extraction based on the flag masks. `profstat` provides several more statistical tools than `adcpsect`, such as first and second differences and histograms. One advantage that `profstat` has is that it allows extraction of not only the ADCP velocities, but also ancillary variables such as `percent good` and `error velocity`. As usual, a control file is modified to set your options. As with `adcpsect`, the programs `llgrid`, `timegrid`, and `arrdep` facilitate the creation of time ranges which are appended to the control file. An example of `profstat.cnt` is seen in Appendix A.19. The control parameters are also shown in Figure 45.

The control file parameters begin with the same preliminary parameters as used by `adcpsect.cnt` (Figure 38). Please refer to Section 9.2.1 for an explanation.

The remaining parameters of the control file are completed for each variable that is desired for extraction. The `< variable name >` can be selected from the DATA LIST (option 7) of the `showdb` utility. For instance, `U, V, W`, or `AMP_SOUND_SCAT` are possible choices, to name a few. Only the binned-arrays can be selected for extraction, which are variables on the DATA LIST screen with `vt_y` greater than 0 and less than 11.

The parameter `< "label" >` denotes a header or title will be placed in the output file for readily identifying each output variable which are placed in the same file. For example `"AMPLITUDE, ADCP DEMO"`. The quotes must be included.

The `reference:` keyword denotes a layer from `start bin` to `end bin` which is used for calculating a mean that subsequently is subtracted from the profile. The `difference:` keyword allows none, first, or second differences to be calculated. The `statistics: scale=` signifies that statistics will be calculated at a scale with convention `0.01` for centimeters and `1.0` for meters.

The keyword `histogram:` signifies that a histogram will be calculated with the characteristics determined by the parameters as shown in Figure 46.

The `flag_mask:` is the same as explained for `adcpsect` in Section 9.2.2. The `time_ranges:` are most readily prepared using the programs `llgrid`, `timegrid`, and `arrdep`.

The `profstat` program is ran in the CODAS execution fashion.

```
CONTROL FILE STRUCTURE:

dbname:       < CODAS database name >
output:       < root for output filename >
step_size:    < number of profiles to advance >
ndepth:       < number of depth bins to read and process>
time_ranges:  < combined | separate >

{ one or more variable names, followed by various arguments as shown }
[ < variable name >
    < "label" >                    { in quotes, for labelling the output }

    [ reference:   < start bin > to  < end bin > ]
    [ difference:  < 0 | 1 | 2 > ]     { for no, first, second difference }
    [ statistics:  scale= < n > ]  { 0.01 for output in cm, 1.0 = m, etc. }
    [ histogram:
        nbins=     < n >
        origin=    < n >
        increment= < n >
        style=     < h | c | n | cn > ]      { n = normalized display
                                                 c = cumulative display
                                                cn = cumulative & normalized
                                                 h = neither }
    [ flag_mask:
        < bit-mask >  { bit-mask is any one or more of those defined in
        .               profmask.h; these cumulatively indicate which criteria
        .               to consider in flagging profile bins as bad prior to
        .               calculations.  Default is ALL_BITS. }
        end ]
end

{ list of YMDHMS time pairs: }
< yy/mm/dd hh:mm:ss > to < yy/mm/dd hh:mm:ss >
.
.
```

Figure 45: Control parameters used by `profstat`.

```
[ histogram:
    nbins=     < n >
    origin=    < n >
    increment= < n >
    style=     < h | c | n | cn > ]  { n = normalized display
                                        c = cumulative display
                                       cn = cumulative & normalized
                                        h = neither }
```

Figure 46: Control parameters used by `profstat` for the histogram.

9.4 Specific Product CODAS Data Access Tools

A variety of utilities extract specific information other than current velocity from the CODAS database. The following CODAS programs create ASCII output files.

- lst_temp This utility is used to retrieve temperature and sound speed data for selected profiles within the given time range(s) from a CODAS database. It is mostly geared for the editing stage when one wants to verify if the transducer temperature appears reasonable.

```
CONTROL FILE STRUCTURE:

dbname:           < input CODAS ADCP database name >
output:           < output filename >
step_size=        < profile sampling rate >
year_base=        < base year for decimal days >
time_ranges:      { list of YMDHMS time pairs }
  < yy/mm/dd hh:mm:ss > to < yy/mm/dd hh:mm:ss >
  .
  .
```

Figure 47: Control parameters used by `lst_temp`.

A control file must be modified for setting the various options. A sample control file `lst_temp.cnt` is provided with the package. The control file parameters are shown in Figure 47.

Within the control file, the `dbname:`, `output:`, `step_size:`, and `year_base=` are the same as described in Section 9.2.1. The `time_ranges:` can be most easily completed using `showdb`.

- lst_prof The `lst_prof` program produces a list of times and positions of selected profiles within the given time range(s) from a CODAS database. One application is to create a list of profile time ranges for each ensemble which can be appended to the `adcpsect` program to allow extraction of velocity data for each ensemble. A control must be edited for setting options. An example of a control file is shown in Appendix A.16. The program is ran in the CODAS execution fashion.

- lstblock This utility retrieves the number of profiles and the time range of each block file. The program is interactive and does not require a control file. It is executed simply by typing

  lstblock

- lst_hdg This utility extracts the ship's mean and last heading from the CODAS database for each profile time. A control file must be edited to set the desired options. An example is shown in Appendix A.15. The program is ran in the CODAS execution fashion.

- lst_btrk This utility extracts the bottom track velocity from a CODAS database for each profile time. A control file must be edited to set the desired options. An example is shown in Appendix A.14. The program is ran in the CODAS execution fashion.

9.5 ASCII Dump of the CODAS Database: asc_dump

A mirror image of the binary CODAS block files can be obtained in ASCII form using program `asc_dump`. The size of the ASCII file is roughly five times the size of the binary file. A sample control file, `asc_dump.cnt` is provided with the CODAS package. Only one control file parameter must be set: `BLOCK_FILES:`. One simply provides a list of block files for conversion, with paths if the program is not executed in the same directory as the block files. The program is ran in the CODAS execution fashion. Details on the output format are described in Appendix D.

10 Graphical Application

The CODAS package offers several useful graphical tools. The plots not only provide scientific insight into the ocean current structure, but also allow another means of quality controlling and assessing the data. Within this section, graphical applications are explained for vector, contour, stick, and on-station-versus-underway plots. The latter two programs require Matlab (see Section 1). As with the other CODAS programs, control files (or Matlab script files) are prepared for setting options prior to running the programs. We begin with a brief overview:

vector This program plots current velocity vectors on user-specified depth levels (or layers) along the cruise track on a latitude/longitude grid. Files containing land boundaries can be overlayed to clarify the geographical region. The program allows multiple plots to appear on a single page. The input velocity data are normally created with programs `llgrid` and `adcpsect`. The program generates PostScript plot files.

contour This standard FORTRAN IV program plots depth cross-sections of current component velocities against latitude, longitude, or time. It is obtained separately from the CODAS software and should be applicable for most host machines. The input velocity data are normally created with programs `llgrid` and `adcpsect`. The program generates PostScript plot files.

stick This Matlab script provides several time series analysis routines for analyzing the mean, trend, tidal, inertial, and residual nature of the high-density data sets. It is best suited for time periods when the ship is on-station for extended periods of time. The input velocity data are normally created with programs `arrdep` and `adcpsect`. The output is plotted within Matlab.

on-station-versus-underway-assessment This Matlab script plots depth sections of the mean and standard deviation of not only any of three velocity components, but also for most of the ancillary parameters, such as error velocity, amplitude echo, etc. The input is normally created with programs `arrdep` and `profstats`. The output is plotted within Matlab.

10.1 Vector Plots

Vector plots of the current velocities are useful for viewing the synoptic features at various depth layers or levels over the course of a cruise; for example, see Figure 48. It can also be used to plot cruise tracks or coastlines. The usage is as follows. First, one obtains a file of current velocities using programs `llgrid` and `adcpsect`. Secondly, a control file is edited to set input, output, and plot options. Finally, the program is ran in the CODAS execution fashion, namely,

```
vector [ control file ]
```

10.1.1 Input Files

The primary input file to the `vector` program is the ASCII file of current velocity components, which are easily pulled from the database using the extraction tool `adcpsect` (see Section 9.2). First one defines the gridding pattern as a function of time with use of the `llgrid` program (see Section 9.1.1). The output of `llgrid` consists of a set of time ranges, which are concatenated into the control file `adcpsect.cnt`.

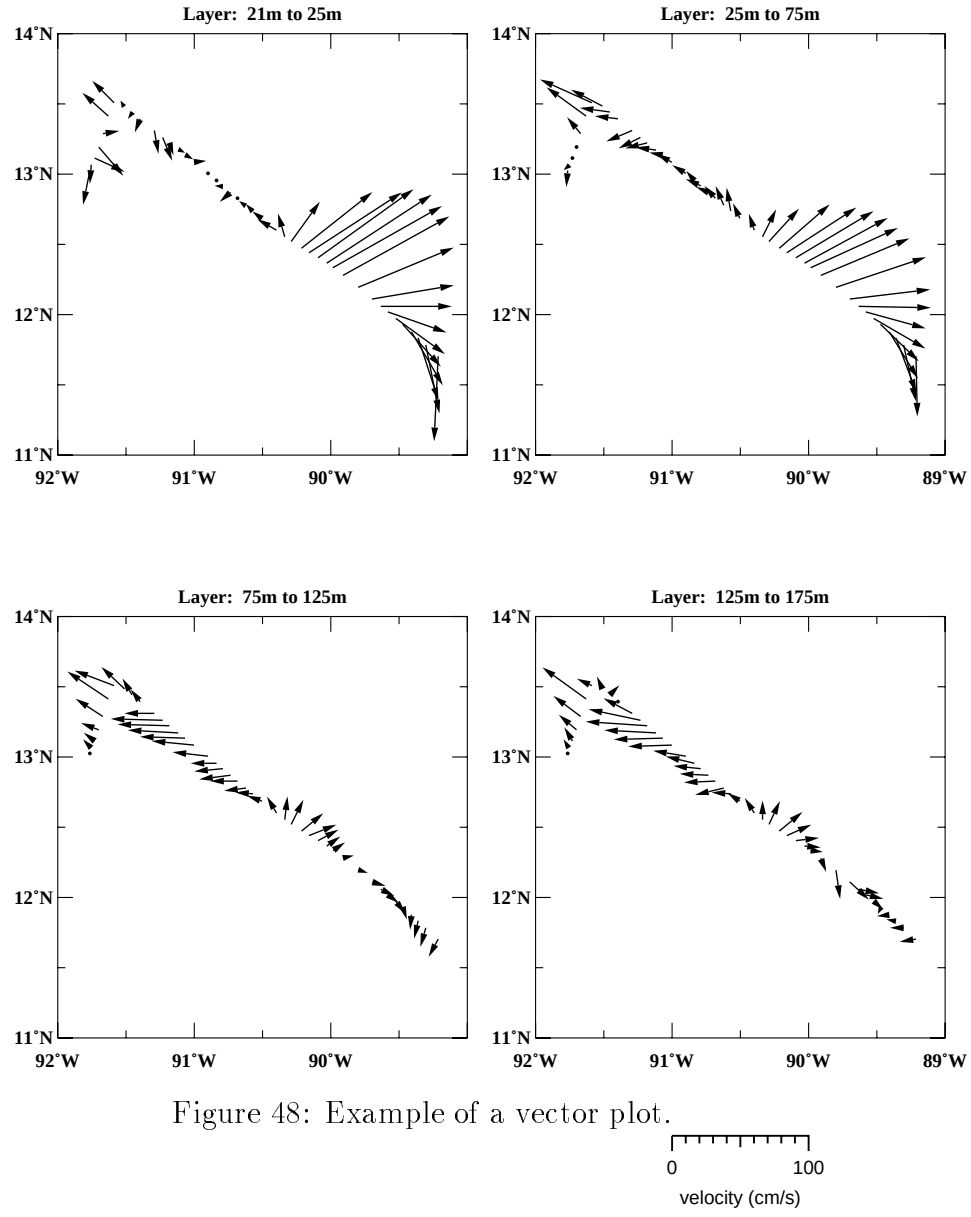


Figure 48: Example of a vector plot.

For each time range (i.e. grid box), the ensembles are averaged in the vertical for each layer as defined by parameters in the `adcpsect` control file. The `.vec` output of `adcpsect`, which consists of longitude, latitude, u- and v-components of velocity for each layer then serves as the input to the `vector` plotting program.

A secondary optional input file to the `vector` program is a land boundary file. Several `.map` files are provided with the CODAS package (Figure 49). The file name explicitly denotes the region. This is just an arbitrary collection of what has become available to us. The resolution varies from rough (`world.map`) to fine (`solomon.map`).

```
1)  westpac.map (West Pacific)

    resolution:  medium
    lon_range :  120E to 160E
    lat_range :   10S to  15N

2)  solomon.map (Solomon Islands)

    resolution:  high
    lon_range :  140E to 161E
    lat_range :   12S to   9N

3)  hawaii.map (Hawaiian Islands)

    resolution:  high
    lon_range :  161W to 154W
    lat_range :   18N to  23N

4)  asia.map (Asia and Pacific Ocean)

    resolution:  medium
    lon_range :   10E to 236E (124W)
    lat_range :   55S to  82N

5)  world.map

    resolution:  low
    lon_range :    0E to 360E
    lat_range :   90S to  90N
```

Figure 49: Presently, the vector program comes with five map files of varying resolution and coverage.

The `.map` file is an ASCII text file with two columns for each record specifying

the longitude and latitude in degrees. A flag record,

```
-9999.00  -9999.00,
```

signals boundaries between pieces of land and a special flag record,

```
-8888.00  -8888.00,
```

signals inland water basins.

If one uses only a small subset of a map file to generate several plots, one can optionally create their own subset map file; thus, program speed is increased. This is especially evident when using the rather large asia.map and world.map files. A CODAS utility, program `extract`, is available for creating a subset map file from a parent map file. To run the map extracting program, type:

```
extract <input map file> <output filename> <minimum longitude>
        <maximum longitude> <minimum latitude> <maximum latitude>
```

where the input map file contains the longitude-latitude coordinates, and the output filename will contain the subset specified by the minimum and maximum, longitude and latitude parameters.

10.1.2 Setting the Control File

Options governing the input, output, and plot parameters are set by the user within the `vector.cnt` control file (see Appendix A.30), which is provided with the CODAS package. The syntax within the control files was explained in Section 3.2. We recommend renaming this file to a name related to the type of job; for instance, `vechaw1.cnt` would be a suitable name for the first set of vector plots of a cruise leg near Hawaii. Thus, when modifications to the plots are desired, one has various examples to use as models.

The control files consists of many keywords for configuring the various options. Each line begins with a keyword recognized by the vector program, which is then usually followed by the user-supplied value for that particular option. All text enclosed with `/* */` are comments and therefore ignored.

The user must then edit the value to the right of the keyword. If a particular option is not desired or the default value will suffice, the user can simply delete that entire line. An explanation of control file parameters is given in Appendix A.30.

10.2 Contour Plots

Contour plots are very useful for analyzing the fine details of the current structure in the vertical along a given transect. An example can be seen in Figure 50. Typically, a cruise is broken into pieces for each cruise segment for which the ship steams in a roughly constant direction, such as an equatorial transect.

The contour software is written in standard FORTRAN IV. It is distributed separately from the CODAS package because it is not readily portable to all machines that have been made accessible to CODAS. However, with minor software modifications and recompilations, it should be portable to most computers. If you encounter any problems, please inform the authors.

Usage of the contour program parallels the use of most CODAS software. The user prepares an input file (`*.con`) of current velocity component data using program `adcpsect` described in Section 9.2. A plot is made for each velocity component by configuring options within the control parameter file, `contour.cpa`. Separate control files and program executions are used for each velocity component. The program is ran by entering

`contour` [*file.con*] [*contour.cpa*]

Where *file.con* is the output from `adcpsect` and *contour.cpa* is the contour control file.

10.2.1 Input File

For each leg of a cruise that is chosen for a contour plot, the user prepares a file of current velocity components. One normally begins by defining the gridding scheme in horizontal space using program `llgrid` (see Section 9.1.1). This program breaks the spatial domains into time ranges, which are concatenated to the control file of program `adcpsect`. The `adcpsect.cnt` control file is modified to configure the contour options as discussed in Section 9.2.2. An example is shown in Appendix A.2.

The output `*.con` file of `adcpsect` consists of either time, latitude, or longitude as the first column, depth as the second column, and u- and v-components as the last two columns.

10.2.2 Setting the Control File

One sets plot options within the file `contour.cpa`. We recommend naming this to a name related to the section, such as `eq1_u.cpa` for "first equatorial transect for u-component". Thus one easily creates a control file for the v-component by copying the former control file to a name such as `eq1_v.cpa` and making a few minor changes, such as the title. The control files are modified using a text editor. This file differs from the `*.cnt*` files of the other CODAS programs in that keywords are not used. Suggested values, defaults, and comments for each line are provided. A brief discussion of the

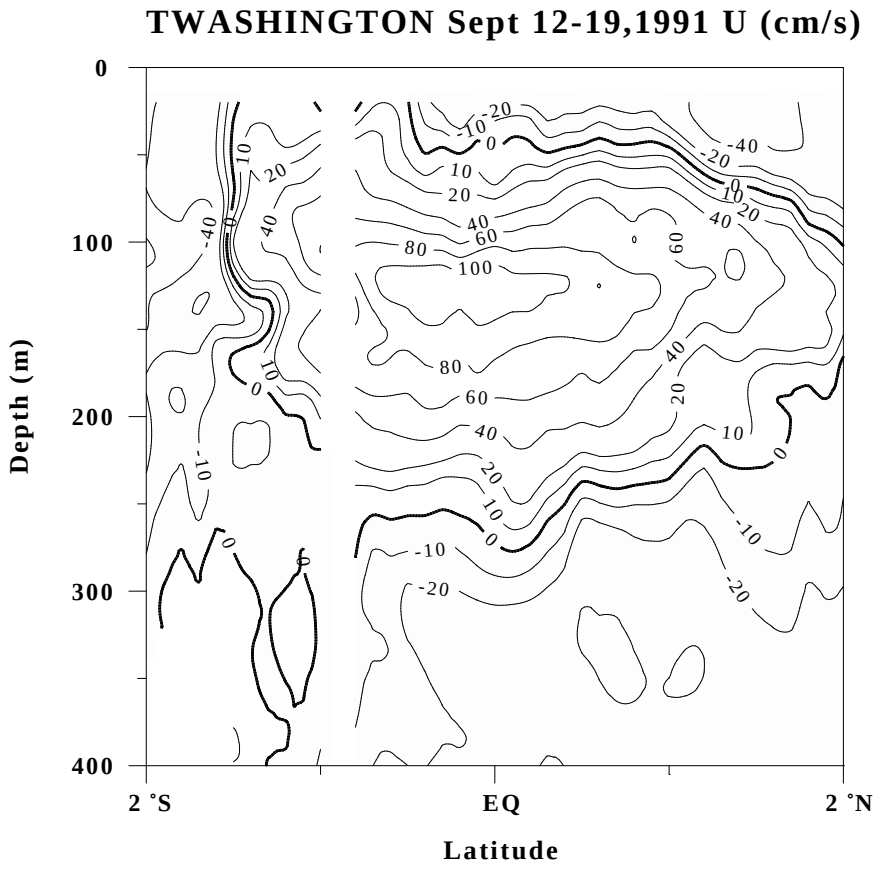


Figure 50: Example of a contour plot. The vertical white streak is due to a data gap and use of the option of not interpolating across the gap.

various options is given below. The comments in the control file are adequate for most options.

We recommend leaving the suggested values for the `I/O CONTROL DATA:` lines. The `TITLE INFORMATION:` allows up to 84 characters on a given record. Within the `GRID CONSTRUCTION & INTERPOLATION PARAMETERS:`, the `NX` and `NY` define the number of grid lines for the X-axis and Y-axis, respectively. These are usually taken as

$$NX = (abs(XL - X1)/Xstep) + 1$$

Where `X1` is the smallest x-axis value, `XL` is the largest x-axis value (remember the x-axis is latitude, longitude, or time), and `Xstep` is typically 0.1 (inch). The same logic follows for the y-axis in determining `NY`. The rest of the parameters are normally left at the default values as given in the sample control file.

Within the `PLOT PARAMETERS`, the `XLPL` and `YLPL` parameters denote the lower right corner for the X-coordinate and the upper left corner of the plot for the Y-coordinate, respectively. The values are determined as

$$XLPL = (Xscale * abs(Xrange) + X1PL)$$

Where `X_scale` is the typically 0.5 (1 inch per 2 degrees), `X_range` is just the highest minus the lowest x-axis boundary, and `X1PL` is the number of inches from the left of the page for the lower left corner of the X-coordinate. The same logic follows for the `YLPL`. The `XTTL` and `YTTL` are normally set to `X1PL` and `YLPL` plus 0.2, respectively. The remaining parameters are normally left at the default settings and are explained in the control file.

The `CONTOUR LEVELS:` configure the intervals at which the velocities are contoured, the type of line, labelling, and shading. The `AXES ANNOTATION PARAMETERS:` define the axes labelling and ticking. The `NTICX` and `NTICY` are normally reset for each contour section. The `X-axis label` (and thus `NXCHAR`) will change only when a different variable is used for the abscissa. The remaining options can be left at the default settings.

10.3 Stick Plots

A time series analysis tool is provided for analyzing the mean, trend, tides, inertial, and residual fields versus depth. Normally one chooses to analyze a period of at least a 3-day span while the ship is on-station in order to obtain statistically significant results, although some applications are relevant for sections while the ship is underway. The software requires Matlab, a commercial statistical and graphical tool. An address for Matlab is given in Section **??**.

The usage of the task consists of preparing the input files of current velocity data using `adcpsect`, modifying a Matlab script file (`runstick.m`), entering Matlab, and executing the job.

10.3.1 Input Files

The input to `runstick.m` is generated by program `adcpsect`. First, one prepares time ranges to be used by `adcpsect` using `llgrid` or `timegrid` (see Sections 9.1.1 and 9.1.2. The former is more applicable if the ship is steaming for the desired period to analyze, while the latter is used for on-station time periods. The latter breaks the on-station time period into equal time ranges, such as hourly, that in turn, are used for calculating means of the u- and v-components of velocity at pre-defined depths. One helpful method for determining if the ship is on-station is to use cruise track plots of time versus latitude and longitude. The time ranges as produced by the programs above are concatenated onto the `adcpsect.cnt` control file. Parameters within this control file are set similarly to the `contour` options. An example is given in Appendix A.4. The desired output files from `adcpsect` are Matlab files with filename extensions, `_xy.mat` and `_uv.mat`.

10.3.2 Setting Options within the Matlab Script

One sets the program options within a Matlab script, `runstick.m`, using a text editor. An example is shown in Appendix B.6. A variety of stick plots can be produced:

1. Velocity
2. Mean, trend, tides, and inertial
3. Residual
4. Inertial clockwise (cw) and counterclockwise (ccw)
5. Diurnal cw and ccw
6. Semidiurnal cw and ccw

The `runstick.m` can also perform an harmonic analysis of the currents and produce stick plots of the mean and trend and ellipse plots of the phase and direction of the diurnal, semidiurnal, and inertial signals.

The Matlab script control file is self-explanatory and does not need elaboration. The parameter that most often must be adjusted is the `xyrange =` which sets the ranges for the abscissa and ordinate of the plot. Once the parameters have been set, one enters Matlab and types:

```
runstick
```

Various plots are generated on the screen which can be saved as PostScript files. Examples are shown in Figures 51, 52, 53. Within Figure 51, the orientation of each stick (top panel) gives the direction of the current: up is northward, to the right

is eastward. The bottom panel shows the results of a least squares fit of the hourly averages to a mean, trend, semidiurnal and diurnal tides, and an inertial cycle. In the first column, the arrow shows the mean current, and the headless stick shows the sum of the mean plus the trend at the end of the station. For each harmonic, the current ellipse is shown in the first column. The orientation of the stick in the second column shows the direction of that harmonic component of the current at the beginning of the station, and the arrowhead at the end of the stick shows the direction of rotation of the current vector around the ellipse. Within Figure 52, the top panel displays the absolute velocities for the sum of the mean, trend, tides and inertial while the bottom panel shows the residuals, which is just the difference of the former with the original absolute velocities. Finally, Figure 53 shows the absolute velocities of the inertial cycle (top panel) and the diurnal tides (bottom panel) which were obtained with least squares fit of the hourly averages. The `cw` and `ccw` refer to clockwise and counterclockwise rotation, respectively. A similar plot for the semidiurnal tide is not shown.

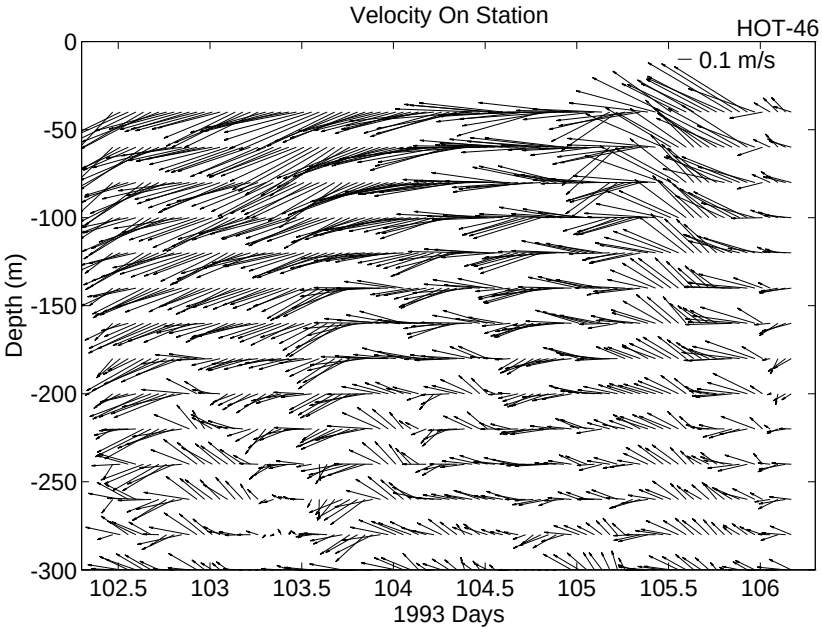
10.4 On-station versus Underway Plots

A helpful method of assessing the data quality is plotting depth averages of the velocity components and various ancillary parameters for on-station versus underway portions of the cruise. The procedure entails first running a program `arrdep` that strips the time ranges for the on-station and underway time periods of the cruise, secondly running CODAS program `profstat` to extract and statistically analyize the data, and third plotting the output of the on-station versus underway statistical summaries in Matlab.

10.4.1 Input Files

One prepares time ranges based on whether the ship was on-station or underway using the `arrdep` program (see Section 9.1.3). The software package provides control files for both the on-station and underway periods as `arrdepos.cnt` and `arrdepuw.cnt`. Examples can be seen in Appendix A.6. The time ranges from these programs are concatenated onto the control files, `profstos.cnt` and `profstuw.cnt`, for the on-station and underway periods, respectively. These control files are used for setting options for the `profstat` program (see Section ??), which accesses the CODAS database. As previously explained, `profstats` allows extraction of ancillary parameters as well as current velocities.

The output of `profstats` is available as an ASCII file (`.prs`) and as a Matlab file (`_os.mat` or `_uw.mat`). The Matlab files are used as input to the on-station versus underway plots.



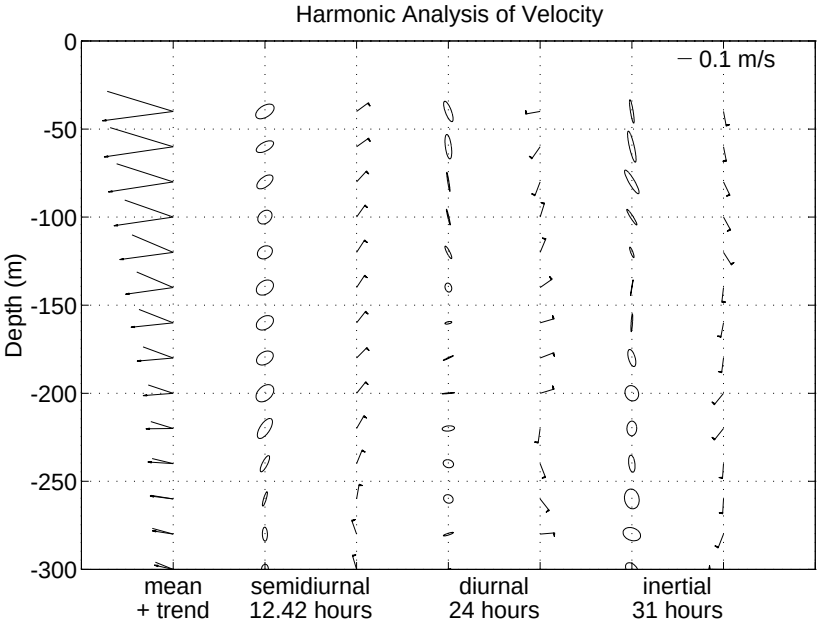


Figure 51: Example of a stick plot (above) for the absolute velocities of hourly averages and harmonic analysis (below). The vertical scale is heavily distorted in this plot relative to how it is produced within Matlab. Additional explanation is provided in the text.

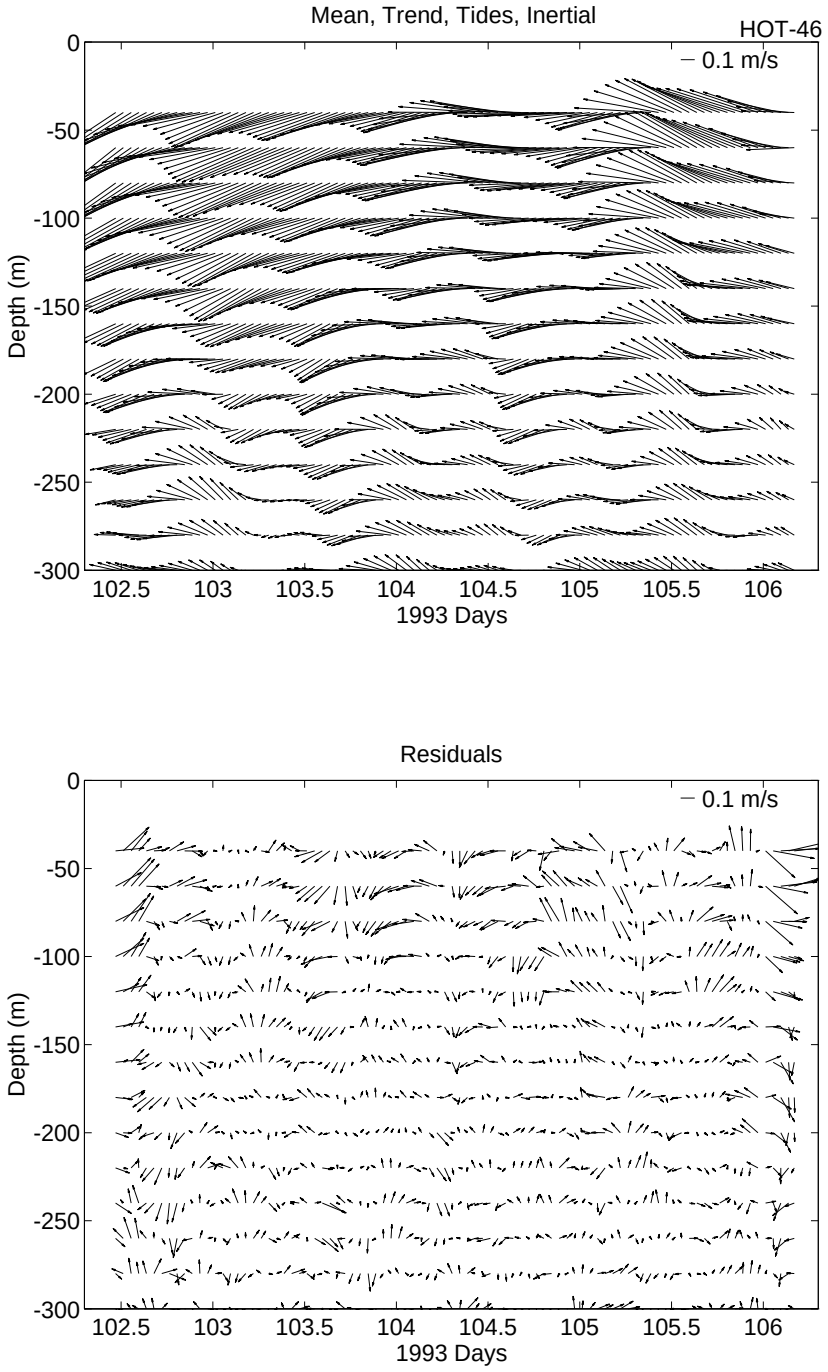


Figure 52: Stick plot (above) for the the sum of the mean, trend, tides and inertial cycle and (below) for the residuals which are the difference of the top panel with the observed absolute velocities. The vertical scale is heavily distorted in this plot relative to how it is produced within Matlab.

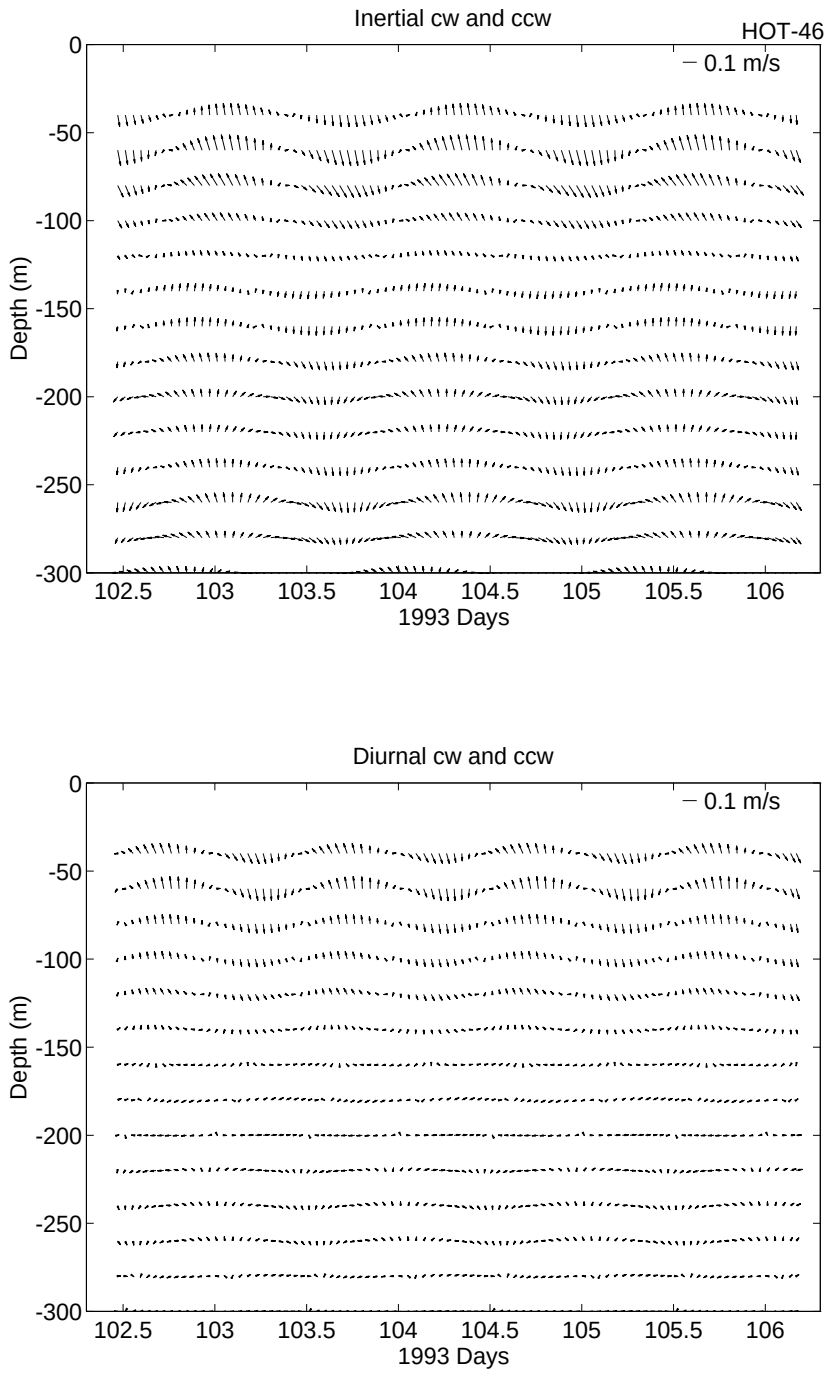


Figure 53: Stick plot (above) for the inertial cycle and (below) for the diurnal tide. The `cw` and `ccw` refer to clockwise and counterclockwise rotation, respectively. The vertical scale is heavily distorted in this plot relative to how it is produced within Matlab.

10.4.2 Setting the Options in the Matlab Script

A Matlab script `stn_udw.m` (see Appendix B.8) is provided with the CODAS package. This script is modified with a text editor. The input and output file names must be set by the user. The plot subtitle normally does not need to be altered. Once the script has been prepared, one enters Matlab and types the following:

```
stn_udw
```

Various plots are spooled to the screen and all plots are placed in a PostScript file for printing. Examples of some of these are shown in Figures 54, 55, 56, and 57.

11 Miscellaneous Utilities

A variety of useful utilities are available for pulling information from the CODAS database or for making minor changes to some of the structures or array data.

11.1 Generating Useful Listings

This section describes a set of programs that can generate lists of particular variables from a CODAS database.

11.1.1 Listing Ship Heading

To extract the mean and last ship heading variables from the `ANCILLARY` structures in the database, the program `lst_hdg` is used. The operator must set up the control file, `lst_hdg.cnt`, a copy of which is placed by the `adcptree` script in the `cal/rotate/` subdirectory of the processing tree, and is shown in Appendix A.16.

The parameter `dbname`: specifies the name (and optional path) of the database to be modified.

The parameter `output`: specifies the name of the output ASCII file.

The parameter `step_size=` allows one to skip profile times with default = 1 (grab information for every ensemble time).

The parameter `year_base=` establishes the year to be used for expressing time in decimal days.

Finally, the keyword `time_ranges`: heralds the list of time ranges of interest for which the heading data will be extracted.

11.1.2 Listing the Profiles in the Database

The program `lst_prof` is used to extract a list of profiles within a given set of time ranges. The listing consists of the profile times with latitude and longitude bracketed within comment delimiters.

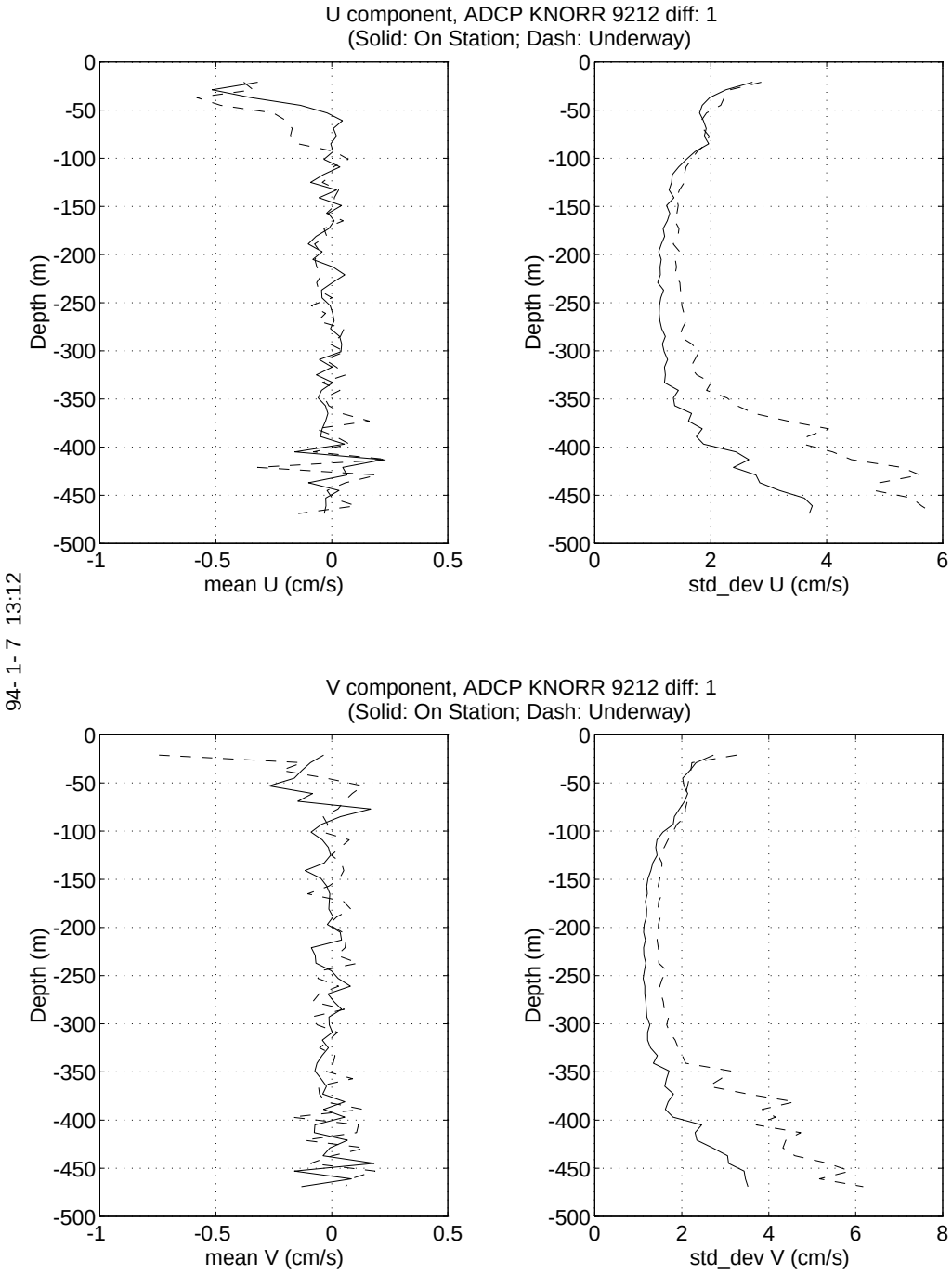


Figure 54: Example 1 of quality assessment plot.

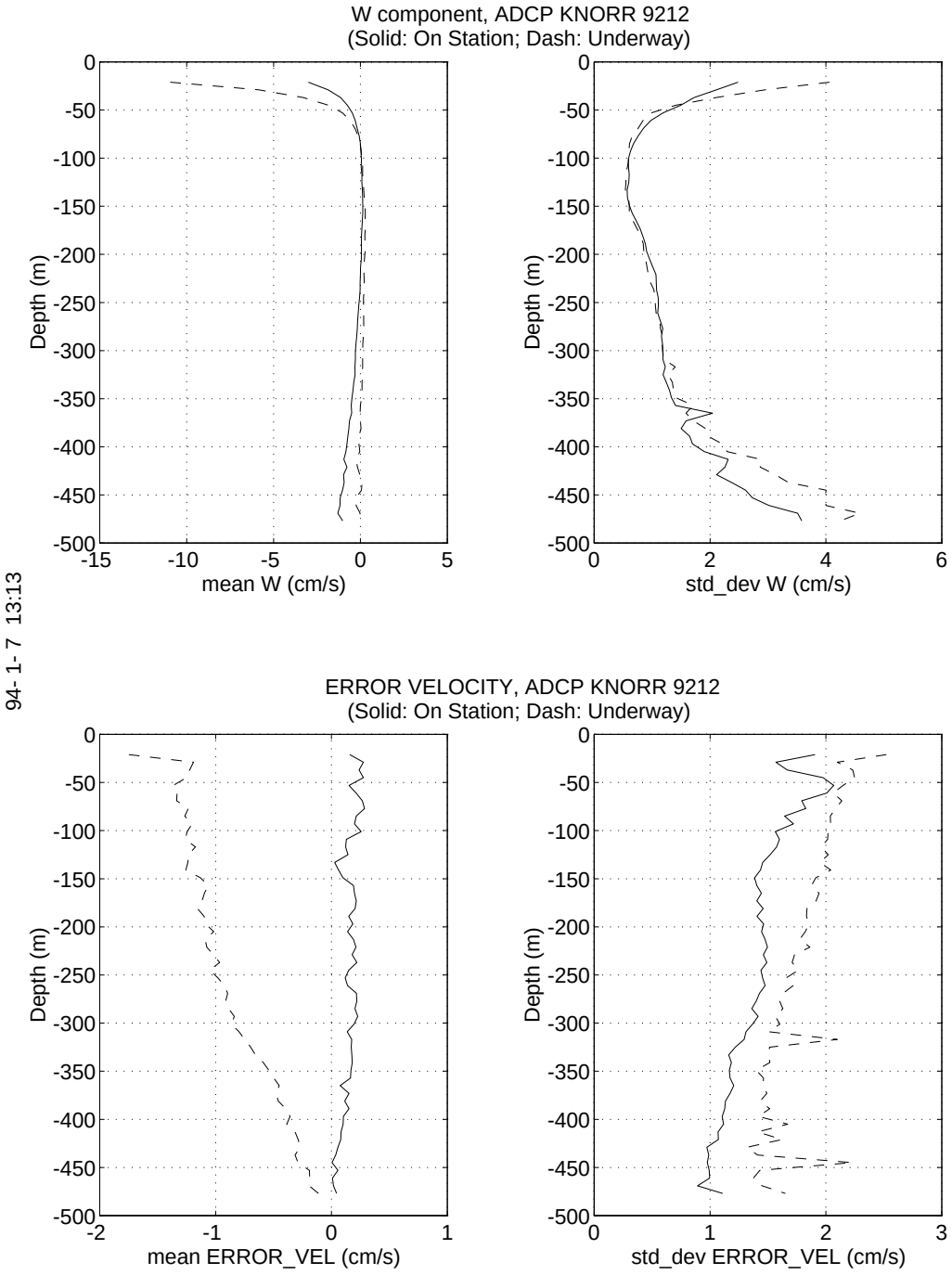


Figure 55: Example 2 of quality assessment plot.

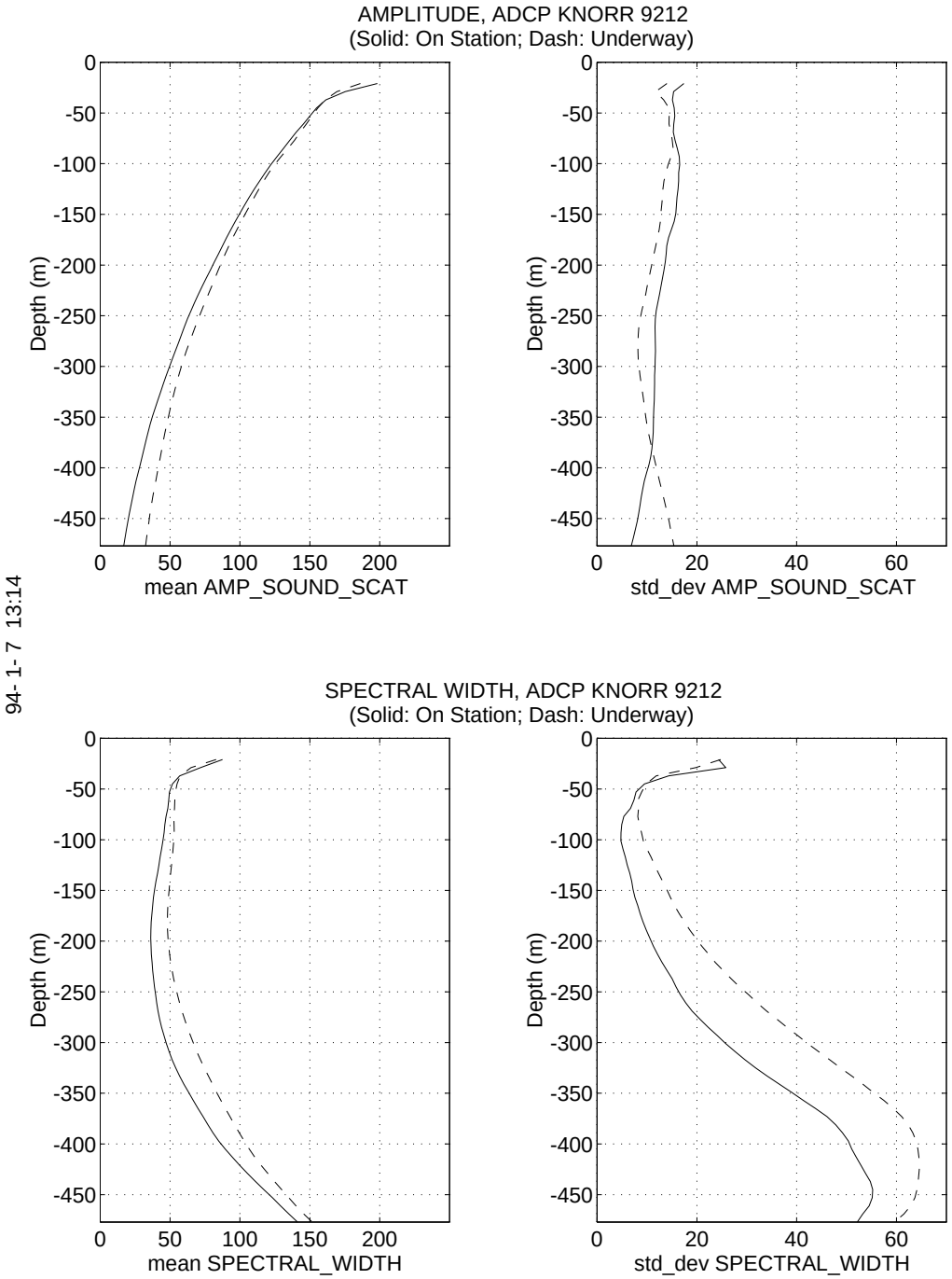


Figure 56: Example 3 of quality assessment plot.

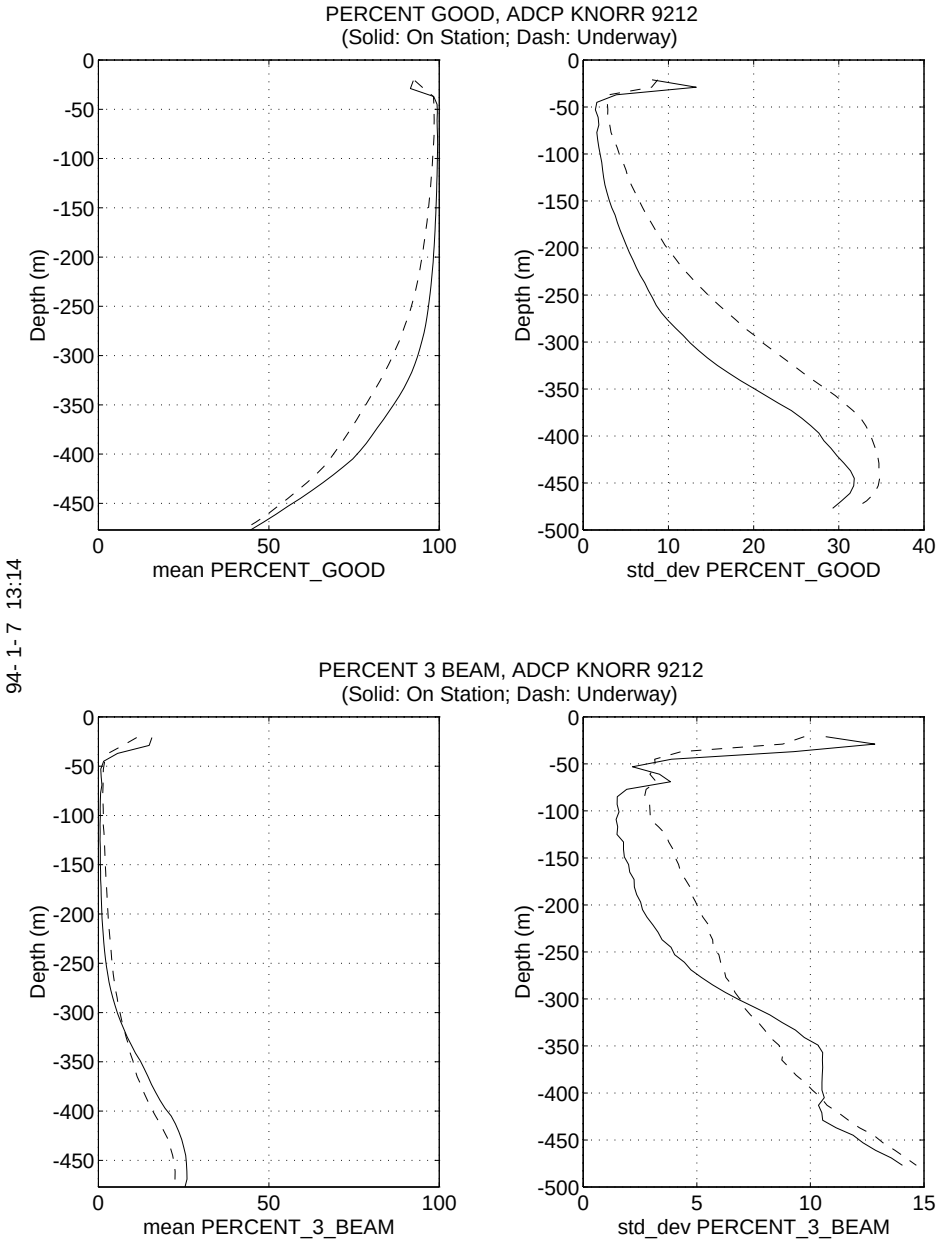


Figure 57: Example 4 of quality assessment plot.

To run the program, the operator must set up the control file, an example of which (`lst_prof.cnt`) is copied by the `adcptree` script to the `adcpdb` subdirectory, and is listed in Appendix A.16. The parameters are identical to those used in `lst_hdg` control file as described in the preceding section. After editing the control file, the operator simply types:

`lst_prof` [`lst_prof.cnt`]

11.1.3 Listing Database Blocks

There are instances when a listing of the database blocks may be useful, as this enables conversion from logical block numbers to time ranges, and provides a profile count within each database block. The program `lstblock` is used to generate this list. An example of the output file is provided in Figure 58.

The program is run by typing:

`lstblock` [*path*] *dbname outfile*

11.1.4 Listing the History of the Configuration Settings

One way to view the configuration as set by the operator during data acquisition is to use `showdb`, select 16 (GET DATA), and choose entry 35 (CONFIGURATION_1). This provides a snapshot of the configuration settings for the given block. A utility, `lst_conf`, allows one to generate a table of the configuration settings over a given time span. One must set options in a control file, `lst_conf.cnt`. These options are similar to the control file options seen in `lst_prof` described above (except there is not a `step_size` keyword). After editing the control file, the operator simply types:

`lst_conf` [`lst_conf.cnt`]

11.2 Updating the Database

This section describes programs that can be used to modify the database, but are not routinely employed in ADCP processing. They cover functions like modifying the profile times after the raw data have already been loaded into the database, deleting entire blocks or a range of profiles from the database, undoing some of the editing, etc.

11.2.1 Changing Profile Times after Loading

The best time to adjust the recorded profile times is during loading, using the program `loadping`. This is because `loadping` allows both a constant offset and drift correction. However, many installations fail to provide a means of checking the profile times for accuracy prior to loading, and the need for and parameters for time correction may not become evident until the calibration stage. Hence, it may become necessary to adjust

```
/* Database summary for: 00072 */
/* Number of blocks: 14 */
/*   0  n=   4  00072001.blk */    93/09/09  19:56:30  to   93/09/09  20:14:58
/*   1  n= 189  00072002.blk */    93/09/09  20:21:30  to   93/09/10  12:01:30
/*   2  n=  84  00072003.blk */    93/09/10  12:11:50  to   93/09/10  19:06:50
/*   3  n= 278  00072004.blk */    93/09/10  19:11:50  to   93/09/11  18:16:49
/*   4  n= 278  00072005.blk */    93/09/11  18:21:51  to   93/09/12  17:26:50
/*   5  n=  28  00072006.blk */    93/09/12  17:31:50  to   93/09/12  19:46:50
/*   6  n= 203  00072007.blk */    93/09/12  19:55:10  to   93/09/13  12:45:12
/*   7  n=  47  00072008.blk */    93/09/13  12:52:32  to   93/09/13  16:42:32
/*   8  n=  89  00072009.blk */    93/09/13  16:47:32  to   93/09/14  00:07:33
/*   9  n= 189  00072010.blk */    93/09/14  21:13:53  to   93/09/15  12:45:36
/*  10  n= 278  00072011.blk */    93/09/15  12:50:37  to   93/09/16  11:55:36
/*  11  n= 278  00072012.blk */    93/09/16  12:00:36  to   93/09/17  11:05:37
/*  12  n=  72  00072013.blk */    93/09/17  11:10:37  to   93/09/17  17:05:36
/*  13  n=  19  00072014.blk */    93/09/17  17:15:42  to   93/09/17  18:45:43
```

Figure 58: Output from program `lstblock`. First column is the logical blocknumber, n= number of profiles, followed by the block filename and time range.

the profile times after loading. This is accomplished with the program `chtime`. In its present version, `chtime` is limited to applying a constant offset to the times of profiles within a given range of blocks. A correction for drift can be crudely approximated by specifying increasing (decreasing) offsets for successive blocks.

To use this program, the operator must edit the control file `chtime.cnt`, a copy of which is available from the `codas3/cntfiles/` subdirectory, and is presented in Appendix A.7.

The parameter `dbname:` specifies the name (and optional path) of the database to be modified.

The parameter `seconds=` specifies the offset to be applied, in integral seconds.

The parameter `block1=` indicates the logical block number from which to start applying the correction. The logical block number is best established by using the `showdb` utility program.

The parameter `block2=` indicates the last logical block number for which the correction is to be applied.

If different offsets are to be applied to the various blocks, then the `chtime` program must be run as many times, with the control file parameters edited to reflect the different setting between each run.

11.2.2 Deleting Blocks and Profiles from the Database

During the database editing stage, a facility for editing out entire profiles was already discussed. This consisted of setting the `last_good_bin` field of the `ACCESS_VARIABLES` variable to either 0 (very shallow profile) or -1 (instrument glitch) using the `dbupdate` program. The utility described in this section is intended primarily to rectify problems other than the aforementioned, specifically to "unload" profiles that should not have been loaded into the database in the first place. This is imperative in the case where a set of profiles were inadvertently loaded more than once, as the presence of profiles with duplicate times can lead to confusion when attempting to search for profiles by time, as if often done. This utility is also applicable for deleting a range of profiles collected for test purposes only, that the operator does not want to be considered as part of the regular data. (Usually, this involves much fiddling with the instrument configuration, which can unnecessarily complicate data processing and extraction.) Another application would be to delete profiles collected while the ship is in port, probably because collection was started too early prior to departure, or not stopped soon enough after arrival.

The utility program is known as `delblk`, and is run interactively. Prior to starting the program, the operator must already have identified the profiles to be deleted, by block number (if entire blocks are to be deleted), by block and profile number (if only subsets of blocks are to be deleted), or by time range. Note that in the case of duplicate profiles, the last option is not recommended if some amount of processing has already been done, since it may not be clear which copy of the profiles was

modified and which may have been missed. Each time the database is updated, the blocks get resorted by time, and the duplicates can easily change order with respect to each other. A subsequent time-based search will always find the set that got sorted ahead of the duplicate. Careful review of the database using `showdb` is recommended to identify the **current** block and profile numbers of the duplicates that ought to be deleted.

To run the program, the operator types

`delblk` [*[path/] dbname*]

The operator is then presented with a menu of options:

```
% DATABASE: ademo successfully opened

 Enter one of the following numbers :

  1.  Delete block(s)
  2.  Delete block-profile range
  3.  Delete time range
  4.  Exit program

 ==>
```

The remainder of the program is self-explanatory. It should be noted that the operation of this program differs from that of `dbupdate`. The program does not alter any of the data variables in the database, but only modifies the profile or block directory, as needed. Basically, it just tags the directory entry for the blocks/profiles to be deleted. Hence, profile data are actually preserved. No storage is recovered even from the directory overhead. If option 1 (Delete block(s)) was used, then the operator can use the appropriate operating system command to remove the block file and recover some disk space. Again, care must be taken in identifying which file can be deleted, because there is no straightforward mapping between the sequence numbers used in the block file names (which actually correspond to the order in which the data were loaded), and the logical block numbers specified in `delblk` (which are recalculated each time the database is closed after being updated and blocks are resorted by time).

11.2.3 Modifying the Depth Array

This section describes a utility for altering the depth array stored in the database. During loading with `loadping`, the depth at the first bin is calculated as:

$transducer_depth + blanking_length + 0.5 * (bin_length + pulse_length)$

and depth at the i^{th} bin is calculated as:
$depthatfirstbin + (i-1) * bin_length$
So the main reason why the depth array may need to be recalculated is a misspecification of the transducer depth parameter in the ADCP DAS menu. The correction then consists of adding some offset to the depth array, as well as correcting the transducer depth field (`tr_depth`) in the database's `CONFIGURATION_1` array.

The program is known as `depthcng` and is run interactively. Since depth is a block variable in a CODAS shipboard ADCP database (i.e., it is stored only once per block), the operator must predetermine which logical blocks in the database need to have their depth array modified.

To run the program, the operator types:

`depthcng` [[*path*] *dbname*]

The program then prompts for the parameters it requires:

```
% DATABASE: ademo successfully opened

 ENTER BLOCK # RANGE (BLK#  BLK#) ==> 0 3

 ENTER depth to be added (m) ==> 5

 ENTER correct value of transducer depth (m) if it needs to be updated
     (or -1 otherwise) ==> 5

 BLOCK # :  0
 BLOCK # :  1
 BLOCK # :  2
 BLOCK # :  3
% DATABASE: Closed successfully
```

The results of the program can be verified by using the `showdb GET DATA` option, retrieving the `CONFIGURATION_1` and `DEPTH` arrays for the blocks specified.

11.2.4 Setting the First Good Bin

This section describes a utility that is used for setting the `first_good_bin` field in the `ACCESS_VARIABLES`. For some installations, for example, it may be necessary to discard the top bin or so for a large range of ensembles whenever the ship is underway. The program `set_top` was written for this purpose. Its effect on the database is to reset the `first_good_bin` field of the `ACCESS_VARIABLES` structure to the operator-specified value, instead of `1`. The operator must edit the control file `set_top.cnt`, a copy of which is stored in `codas3/cntfiles` and shown in Appendix A.25.

The parameter `dbname`: specifies the option path and name of the database to be modified.

The parameter `speed_threshold`: defines the minimum ship speed required over the ensemble for the ship to be considered underway. Ship speed is estimated by averaging the raw ADCP velocities over the reference layer. (If the user decides to reset the top good bin regardless of ship speed, this can be accomplished by selecting a zero speed_threshold.)

This reference layer is selected by specifying the `ref_bin_range`: parameter, the start and end values of which must be delimited by the keyword `to`.

The parameter `first_good_bin`: gives the value to which the `ACCESS_VARIABLES.first_good_bin` field is to be reset.

Finally, the keyword `time_ranges`: heralds the list of one or more time ranges in the database to be modified.

After editing the control file, the operator simply types

```
set_top [ {\it set_top.cnt} ]
```

The result is verified by using the `showdb GET DATA` option to examine the `first_good_bin` field of the `ACCESS_VARIABLES` structure.

11.2.5 Regenerating a Database from Existing CODAS Block Files

A CODAS block directory file (`*dir.blk`) can always be generated from any set of CODAS block files using the utility program `mkblkdir`. This effectively creates a database comprised of that set of block files. This is useful for reducing a database to include only a subset of the original block files, for reassembling a database from block files derived from different databases, or for simply reconstructing a corrupted or missing block directory file. In addition, the program `mkblkdir` can be used to convert the block files to a binary format compatible with the user-specified platform. That is, a database that has been created on a Sun workstation, for example, can be ported to a PC by running it through `mkblkdir`. One may also use this utility to rename the database[14] The database name is derived from the first five characters of the block file root name; for instance, 00231001.blk and 00231002.blk are two block files of the database 00231.

To run the program, the user must first edit the control file. An example is given in Appendix **??**.

The parameter `DB_NAME` is used to specify the name of the database to be created. For PC-DOS compatibility, it must be 5 characters or less in length. An optional directory specification may be included, as long as that directory already exists and the user has write-permission on it.

[14]This utility makes a separate copy of the block files and a new block directory file in the process of renaming.

To create the block directory file, specifications for the DATASET_ID and BLOCK_DIR_TYPE are needed. If these are not specified in the control file, and the option PRD_NAME is not used to specify a producer definition file that contains these specifications, then DATASET_ID defaults to ADCP-VM and BLOCK_DIR_TYPE defaults to 0 (the only type currently available).

These may be provided by either specifying those options in the control file, or by providing the name of a producer definition file containing those specifications, using the option PRD_NAME. Otherwise, DATASET_ID defaults to ADCP-VM and BLOCK_DIR_TYPE defaults to 0 (the only type currently available).

To convert the binary format of the block files to some different format, the user may specify the DESTINATION option. Currently, the values PC_COMPATIBLE_HOST, VAXD_COMPATIBLE_HOST, SUN3_COMPATIBLE_HOST are recognized. If this option is not used, it defaults to HOST_ENVIRONMENT, that is, the type of machine that is currently running mkblkdir. If no conversion is necessary or requested, then the block files are merely copied over to the new database, renaming them if the DB_NAME differs from their original names, and renumbering them in the order that they appear in the control file.[15]

Following the above options, the user can specify the options SD_NAME and BLOCK_FILE any number of times. The option BLOCK_FILE is used to specify the name of a block file to be included in the new database. It may include a directory (path) specification. The option SD_NAME is used to specify an ASCII file of structure definitions. This is necessary if some of the block file variables are of type STRUCT_VALUE_CODE and the structure definitions have not been properly loaded, if at all, at the time the block files were created. If this option is not used, then mkblkdir will assume that the correct structure definitions for any structure-type variables are in the block file itself. Otherwise, a CONVERT_ERROR or UNDEFINED_STRUCTURE error may occur during runtime. For details on how to create a structure definition file, see the section "The External Structure Definition File" in the online document file codas3/doc/codas3.doc.

After editing the control file, the user types:

```
mkblkdir [ mkblkdir.cnt ]
```

Some caution is in order when mkblkdir is run to assemble a database from block files originally belonging to different databases. The time range of the profiles in a given block file should not overlap those of another because of the manner in which database time searches are done: when a particular time is being sought, the database

[15] It should be noted that the block filename numbers are different from the "logical" block numbers used by the CODAS programs, such as that displayed by showdb: the former are assigned in the order in which the block files have been added to the database and do not change thereafter; the latter are a consequence of the CODAS database software sorting the block file starting times into sequential order, any time that a change in the database contents makes this necessary (e.g., when profiles or entire blocks are deleted).

routine starts by identifying which block file encompasses that time by checking out the block file time ranges in the block directory file. As soon as it identifies the first such block file, it then proceeds to search within that block file for the profile with time equal to or greater than the key being sought. If the profile happens to belong to a subsequent block file with an overlapping time range, it will never be found when searching by time. (Programming to work around this constraint is on our to-do list.)

11.3 Conversion of Common Oceanographic Quantities

```
USAGE: atg <salinity> <temperature> <pressure>
EXAMPLE: atg 40.0 40.0 10000.0

The adiabatic temperature gradient is 3.255976E-04 C/dbar
when S = 40 (PSS-78), T = 40 deg C, and P = 10000 dbar.



USAGE: depth <pressure> <latitude>
ASSUMES: standard ocean T = 0.0 deg C and S = 35 (PSS-78)
EXAMPLE: depth 10000.0 30.0

depth = 9712.653072 m at P = 10000 dbar, 30 degrees latitude



USAGE: grav <pressure> <latitude>
EXAMPLE: grav 10000.0 30.0

gravity = 9.804160 m/s^2 at P = 10000 dbar, 30 degrees latitude


SAL78=1.888091 :CND= 40.00,T=40 deg C,P=10000 dbars: M=SAL_TO_CR
compare conductivity ratio: 1.814775
SAL78=40.00000 :CND=1.888091,T=40 deg C,P=10000 dbars: M=CR_TO_SAL
compare salinity: 0.110231


USAGE: press <depth>
EXAMPLE: press 0.0

pressure = 0.000008 dbar at depth = 0 m
```

```
USAGE: svan <salinity> <temperature> <pressure>
EXAMPLE: svan 40.0 40.0 10000.0

specific volume anomaly = 981.301907 cl/t
        density anomaly = 59.820375 kg/m**3
for S = 40 (PSS-78), T = 40 deg C, P = 10000 dbar
```

```
USAGE: svel <salinity> <temperature> <pressure>
EXAMPLE: svel 40.0 40.0 10000.0

sound speed = 1731.995394 m/s
for S = 40 (PSS-78), T = 40 deg C, P = 10000 dbar
```

```
USAGE: theta <salinity> <temperature> <pressure> <reference-pressure>
EXAMPLE: theta 40.0 40.0 10000.0 0.0

potential temperature = 36.890726 deg C
for S = 40 (PSS-78), T = 40 deg C, P = 10000 dbar, ref P = 0 dbar
```

11.4 Time Conversion

Below we describe three programs that are distributed with CODAS for ease in converting between decimal days and other time formats. It should be reiterated that we define decimal days as the number of days that have elapsed since 00:00:00 of 01 January of the base year. For example, 0.5 in 1992 decimal days is 01 January 1992 at 1200 hours.

11.4.1 From decimal day to yy/mm/dd hh:mm:ss

The program `to_date` is used for converting time in decimal days to `yy/mm/dd hh:mm:ss`. Typing the command `to_date` without any arguments gives the following usage message:

```
command line: year_base day
year_base is year-1900, day is the decimal year day.
```

So, to convert day 0.5 of year 1992:

`to_date` *92 0.5*

which returns:

```
92/01/01  12:00:00
```

11.4.2 to_day

The program `to_day` performs the inverse of `to_date`, converting a year, month, day and, optionally, hour, minute, second to decimal day. Typing the command `to_day` without any arguments gives the following usage message:

```
command line: year_base yy mm dd hh mi se
base_year, yy are year-1900.  All others are optional.
```

Some examples follow:
`to_day` *92 92 1 1 12*

```
0.500000
```

`to_day` *91 92 1 1 12*

```
365.500000
```

`to_day` *92 92 2 29 5 45 30*

```
59.239931
```

`to_day` *92 92 12 31*

```
365.000000
```

11.4.3 to_week

The program `to_week` is used to convert time in decimal days to the week of the year (starting from 0), and the number of seconds into that week. Typing the command `to_week` gives the usage message:

```
Usage: to_week year_base decimal_day
For example : to_week 92 123.452
```

Here are a few examples:
`to_week` *93 9*

```
 The decimal week is : 2
 The second in week is : 0.000000
```

9.0 in 1993 decimal days is actually 10 January 1992, a Sunday that starts off the third week of that year, or decimal week 2 (decimal week 0 happens to consist of only 2 days since the year begins on a Friday). of the year. So the decimal week is 2, and it is 0 seconds into that week.

`to_week` *92 9* The decimal week is : 1 The second in week is : 432000.000000

9.0 in 1992 decimal days is 10 January 1993, a Friday in the second week of that year, or decimal week 1 (decimal week 0 consists of only 4 days since the year starts on a Wednesday). So the decimal week is 1, and five days' worth of seconds have elapsed, or 432,000 seconds of the week.

The main use of this program is for matching ADCP times expressed in decimal days to GPS navigation and heading data, where time is often expressed as seconds of the week.

References

[1] Caldwell, P., 1995. *NODC Archives Shipboard ADCP Data.* Earth System Monitor, Vol. 5, No. 3, March 1995.

[2] Chereskin, T.K., E. Firing and J.A. Gast, 1989. *On identifying and screening filter skew and noise bias in acoustic Doppler current profiler measurements.* J. Atomos. Oceanic Technol., 6, 1040-1054.

[3] Joyce, T.M., 1989. On in situ "calibration" of shipboard ADCPs. J. Atomos. Oceanic Technol., 6, 169-172.

[4] Pollard, R. and J. Read, 1989. A method for calibrating shipmounted Acoustic Doppler Current Profilers, and the limitations of gyro-compasses. J. Atmos. Oceanic Technol., 6, 859-865.

[5] RD Instruments, 1989. *Acoustic Doppler Current Profilers Principles of Operation: A Practical Primer.* Available from RD Instruments, 9855 Businesspark Av., San Diego, CA 92131.

A CODAS Control Files

CODAS control files are text files that must be edited by the user to specify the appropriate control parameters to a program. It facilitates documentation of the processing as well repeated execution of a program until satisfactory results are achieved. Below in alphabetical order are excerpts from the control files discussed in the manual.

A.1 adcpsect.cnt Control File Structure

A.1.1 Part 1 of 2

```
/**************************************************************************

  FILE:  adcpsect.cnt

--------------------------------------------------------------------------
  CONTROL FILE STRUCTURE:

  dbname:       < CODAS database name >
  output:       < output filename >
  step_size:    < number of profiles to advance >
  ndepth:       < number of depth bins to read and process >
  time_ranges:  < combined or separate >
  year_base=    < base year for decimal days >

  option_list:
    [ verbose ]                       { prints debugging and auxiliary output }
    [ underway?    < no | yes limit= <3> > ]                  { limit in m/s }
    [ pg_min=      <30> ]
    [ AGC_margin=  <10> ]

    [ skew_limit:
        agc_limit=  <100>
        max_velocity_difference= <0.8> ]                              { m/s }

    [ reference:       { choose only one of the following: }
        < reference_bins <5> to <20> |
          none |
          final_ship_ref |
          nav_position |
          nav_velocity |
          bottom_track |
          dir_position > ]

    [ navigation:      { output in .nav text file }
                       { choose one or more of the ff. then 'end' }
        [reference_bins <5> to <20>] {bin range; first bin is 1}
        [final_ship_ref]
        [nav_position]
        [nav_velocity]
        [bottom_track]
        [dir_position]
        end ]

    [ rotate:  < ship_coordinates | n > ]
        { where <n> is the angle in degrees, old x-axis to new x-axis }

    [ ctd:
        dbname:  < CODAS CTD database name > ]

    [ regrid:
        < integrate | interpolate | centered_average | average |
          per_grid_interval >
        < depth | svan | pot_svan | sigma_theta | sigma_t >
          { one of the last 4 only if the ctd: option has been selected }
        < grid      number= <50> origin= <16> increment= <8> |
          grid_list number= < 5> boundaries: <20> <40> <60> <80> <100> > ]
```

A.1.2 Part 2 of 2

```
/*****************************************************************************

  FILE:  adcpsect.cnt

-----------------------------------------------------------------------------
  CONTROL FILE STRUCTURE:

    [ contour:         { output in .con text file and _uv.mat Matlab Mat-file }
        < latitude | longitude | time >
        < middle | mean >
        minimum_npts= <10>
        units=        <0.01> ]

    [ vector:          { output in .vec text file and _uv.mat Matlab Mat-file }
        minimum_npts= <10> ]

    [ transport:       { output in .trn text file and _tr.mat Matlab Mat-file }
        minimum_npts= <10>
        units=        <1.e6>                                  { for Sverdrups }
        direction=    < all | left | right > ]

    [ sequential_cor:  { output in .cor text file }
        bin_range: <3> to <40> ]  { starting from 1, inclusive }

    [ statistics:      { output in .sta text file }
        [ mean ]
        [ variance ]
        [ extrema ]
        [ norm_extrema ]
        [ indices ]
        [ all ]
        units= <0.01>
        end ]

    [ flag_mask:
        < bit-mask > { bit-mask is any one or more of those defined in
        .              adcpmask.h; these cumulatively indicate which
        .              criteria to consider in flagging profile bins
        .              as bad prior to calculations.  Use this option
        end ]          if you want flagging other than the default ALL_BITS }

    end

  { list of YMDHMS time pairs: }
  < yy/mm/dd hh:mm:ss > to < yy/mm/dd hh:mm:ss >
  .
  .
  .
-----------------------------------------------------------------------------*/
```

A.2 adcpsect.cnt for Extracting Data for Contour Plots

```
/*************************************************************************
   This particular version is used to extract
   water track velocities for plotting contours.
   This version is similar to the version for
   extracting velocities for plotting stick diagrams.
 INPUT:  CODAS database
 OUTPUT: 1. text file (.con file) with following columns:
        a. x-axis data (latitude in decimal degrees for this example)
                  but can be longitude or time)
        b. y-axis data (depth in meters for this example)
        c. zonal velocity (in cm/s for this example)
        d. meridional velocity (in cm/s for this example)
      2. text file of statistics--mean (.sta file)
      3. Matlab files
        a. .muv contains array "uv" with two columns for each ensemble,
          first u and then v.  The row corresponds to the depth grid.
        b. .mxy contains array "xyt" with a column for each ensemble,
          and three rows: longitude, latitude, decimal day;
          and vectors "z" and "zc" with the depth grid and the grid
          interval centers, respectively.
-------------------------------------------------------------------------*/
 dbname:             ../adcpdb/a9212
 output:             s1.con    /* must be <= 5 chars. for DOS */
 step_size:          1
 ndepth:             58
 time_ranges:        separate
 year_base=          1992
 option_list:
  pg_min=           30
  reference:        final_ship_ref
  regrid:           centered_average
             depth
    grid number=   45
     origin=      20
     increment=   10
  contour:          latitude   mean
   minimum_npts=  3
   units=          0.01
  statistics:       all
   units=          0.01
   end
  flag_mask:        ALL_BITS
   end
 end
 /* Time range: 92/12/04  18:48:37 to 92/12/09  17:29:11 */
92/12/04  18:53:29  /* 5965.232  5965.232,    338.78714 (begin time range) */to
```

A.3 adcpsect.cnt for Extracting Navigation

```
/*************************************************************************

  FILE:  adcpsect.cnt

  This control file is used with the adcpsect program
  to extract ADCP data for specified sections along a cruise track
  from a CODAS database, perform the desired calculations,
  and write the result to text and Matlab files.
  The sections are specified as time ranges.

  This particular version is used to extract
  the ship velocity relative to the reference layer.

  INPUT:  CODAS database
  OUTPUT: 1. text file (.nav file) with following columns:
             a. profile time (in decimal days)
             b. zonal velocity relative to reference layer (in m/s)
             c. meridional velocity relative to reference layer (in m/s)
          2. text file of statistics--mean (.sta file)
          3. Matlab files
             a. .muv contains array "uv" with two columns for each ensemble,
                first u and then v.  The row corresponds to the depth grid.
             b. .mxy contains array "xyt" with a column for each ensemble,
                and three rows: longitude, latitude, decimal day;
                and vectors "z" and "zc" with the depth grid and the grid
                interval centers, respectively.
-------------------------------------------------------------------------*/

  dbname:             ../adcpdb/ademo
  output:             ademo    /* must be <= 5 chars. for DOS */
  step_size:          1        /* must be 1 for navigation */
  ndepth:             20
  time_ranges:        separate
  year_base=          1993

  option_list:
    pg_min=           30
    navigation:
      reference_bins  5  to   20
      end
    statistics:       mean
      units=          0.01       /* cm/s instead of default m/s */
      end
    flag_mask:        ALL_BITS
      end
    end

  93/04/09 00:02:00 to 93/04/10 23:58:00

/*************************************************************************/
```

A.4 adcpsect.cnt for Extracting Data for Stick Plots

```
/*************************************************************************
 This particular version is used to extract
 water track velocities for stick plots.

 INPUT:  CODAS database
 OUTPUT: 1. text file (.con file) with following columns:
        a. x-axis data (latitude in decimal degrees for this example)
        b. y-axis data (depth in meters for this example)
        c. zonal velocity (in cm/s for this example)
        d. meridional velocity (in cm/s for this example)
      2. text file of statistics--mean (.sta file)
      3. Matlab files
        a. _uv.mat contains array "uv" with two columns for each ensemble,
          first u and then v.  The row corresponds to the depth grid.
        b. _xy.mat contains array "xyt" with a column for each ensemble,
          and three rows: longitude, latitude, decimal day;
          and vectors "z" and "zc" with the depth grid and the grid
          interval centers, respectively.
-------------------------------------------------------------------------
 dbname:              ../adcpdb/ademo
 output:              os     /* must be <= 5 chars. for DOS */
 step_size:           1
 ndepth:              50
 time_ranges:         separate
 year_base=           1993
 option_list:
  pg_min=            30
  reference:         final_ship_ref
  regrid:            average
              depth
   grid number=     16
    origin=        30
    increment=     20
  contour:           time  middle
   minimum_npts=    1
   units=           0.01
  statistics:        mean variance extrema
   units=           0.01
   end
  end

  time_range:
 93/04/09  00:02:00  to  93/04/10  07:36:00
 93/04/09  00:02:00  to  93/04/09  01:02:00
  .
```

A.5 adcpsect.cnt for Extracting Data for Vector Plots

```
/*************************************************************************
   This particular version is used to extract
   water track velocities for vector plots of a given depth grid.
   It can extract three depth layers at each run.
 INPUT:  CODAS database
 OUTPUT: 1. text file (.vec file) with following columns:
        a. longitude (in decimal degrees)
        b. latitude  (in decimal degrees)
        one or more pairs of: (one for each layer in the depth grid)
        c. absolute zonal velocity for a given layer (in m/s)
        d. absolute meridional velocity (in m/s)
        .
      2. text file of statistics--mean (.sta file)
      3. Matlab files
        a. .muv contains array "uv" with two columns for each ensemble,
          first u and then v.  The row corresponds to the depth grid.
        b. .mxy contains array "xyt" with a column for each ensemble,
          and three rows: longitude, latitude, decimal day;
          and vectors "z" and "zc" with the depth grid and the grid
          interval centers, respectively.
-------------------------------------------------------------------------*/
 dbname:                   ../adcpdb/a9212
 output:                   a9212  /* must be <= 5 chars. for DOS */
 step_size:                1
 ndepth:                   58
 time_ranges:              separate
 year_base=                1992
 option_list:
  pg_min=                 30
  reference:              final_ship_ref
  regrid:                 average
                depth
   grid_list number=    10
    boundaries:        /* 20 */ 21  25  75 125 /* The 1st boundary must */
                175 225 275   /* start at or below the */
                325 375 425   /* shallowest depth bin  */
  vector:
   minimum_npts=        2
  flag_mask:              ALL_BITS
   end
  end

/* Time range: 92/12/04  18:48:00 to 93/01/22  10:58:00 */
92/12/04  18:48:33  /* -17.537  -149.569,    338.78372 (begin time range) */to
```

A.6 arrdep.cnt

```
/*******************************************************************************
     This control file is used with the arrdep program
     to generate a text file of time ranges for profiles that
     occurred while the ship was on-station or underway,
     depending on the setting of the range: parameter below.

 INPUT:  text file of ship velocity (.nav output from adcpsect)
 OUTPUT: text file of time ranges
--------------------------------------------------------------------------------
 CONTROL FILE STRUCTURE:

 reference_file:   < input file of ship velocity relative to reference layer >
 output_file:      < output filename >

 year_base=        < base year for decimal days >

 n_refs=       <13> { number of ensembles to use in calculation }
 i_ref_l0=     <6>  { index of first ensemble used before change in velocity }
 i_ref_l1=     <6>  { index of last ensemble used before change in velocity }
 i_ref_r0=     <8>  { index of first ensemble used after change in velocity }
 i_ref_r1=     <8>  { index of last ensemble used after change in velocity }

 range:        < on_station | underway | all | whole_station | whole_underway >
 up_thresh=    <2.5> { m/s, for jump detection, about 1/2 underway speed }
 down_thresh=  <2.5> { m/s, for jump detection, about 1/2 underway speed }

 margin=       <10>  { seconds to subtract from first time, add to second,
               to allow for rounding errors and ensure that the
               time range brackets the ensemble times }
--------------------------------------------------------------------------------*/

   reference_file:   ../nav/ademo.nav
   output_file:      ademo_uw.arr

   year_base=   1993
   n_refs=       13        /* number of ensembles used in calculation
                    of time shift */
   i_ref_l0=     1         /* first and  */
   i_ref_l1=     5         /* last profile used before change of nav - vel*/
   i_ref_r0=     7         /* "l" is before jump, "r" is after */
   i_ref_r1=     11        /* minimum is 1, not zero */

   range:       underway
   up_thresh=    2.5         /* m/s, for jump detection */
   down_thresh=  2.5

   margin=    10          /* seconds to subtract from first time, add to
                  second, to allow for rounding errors and ensure
                  that the time range brackets the ensemble
                  times  */
```

A.7 chtime.cnt

```
/*****************************************************************************

 FILE:  chtime.cnt

     This control file is used with the chtime program
     to shift the profile times by some constant amount
     in one or more CODAS database blocks.

 INPUT:  CODAS database
 OUTPUT: CODAS database with updated profile times
-----------------------------------------------------------------------------
 CONTROL FILE STRUCTURE:

 dbname:    < CODAS database name >
 seconds=   < no. of seconds to add to profile time >
 block1=    < starting block number >
 block2=    < ending block number >
-----------------------------------------------------------------------------*/

 dbname:   \wepocs\adcp\w143
 seconds=  -360
 block1=   0
 block2=   0           /* update only block file no. 0 */

/*****************************************************************************/
```

A.8 contour.cpa

```
/* SAMPLE CONTROL PARAMETERS FILE FOR CONTOUR PROGRAM:

   Defaults values have been provided as starting points for typical plots.
   You may alter them as needed.  You still have to specify those parameters
   for which no defaults can be given, for obvious reasons.

   Warning:  The contours program does not mind numeric parameters being
   dispersed across new lines.  However, it crashes if string parameters
   (plot title, input data format, X- and Y-axis labels) do not appear
   on the immediately succeeding line.  Be careful about placement of
   comments (make sure you comment out any intervening new lines) and
   blank lines in these cases.

   Use the new sample.src file as a sample batch file for submitting jobs
   that use this commented version of the .CPA file.  Change all file names
   with prefix "sample" in your copy of the batch file to your own file names.
   Leave everything else (including "temp.cpa") as is.

  -------------------
   I/O CONTROL DATA:
  -------------------*/
2       /* KD1 = title pen (0, 1, 2) 2 = title drawn with heavy pen */
1       /* KD2 = grid constr/interp (0, 1) 1 = user-supplied */
2       /* KD3 = ungridded input data (0, 1, 2) 2 = suppress listing of data */
1       /* KD4 = plot parameters (0, 1) 1 = user-supplied */
2       /* KD5 = contour levels/shading (0 to 3) 2 = user-supplied w/ shading */
4       /* KD6 = blanking polygon data (0, ...) 4 = suppress plot & listing */
2       /* KD7 = tracing (2 to 6) 2 = minimal (flow of execution) */
0       /* KD8 = line & shade drawing (0, 1) 0 = no drawings requested

  --------------------------------
   TITLE INFORMATION: 21A4 format
  --------------------------------*/
TWASHINGTON Sept 12-19,1991 U (cm/s)

/*-----------------------------------------------
   GRID CONSTRUCTION & INTERPOLATION PARAMETERS:
  -----------------------------------------------*/
0       /* INOUT = I/O of gridded data (0, 1, -1) 0 = no grid I/O */
41     /* NX = no. of X-axis grid lines = abs(XL - X1) / Xstep + 1 */
41      /* NY = no. of Y-axis grid lines = abs(YL - Y1) / Ystep + 1 */
-2      /* X1 = smallest X value of grid */
-400    /* Y1 = smallest Y value of grid */
 2      /* XL = largest X value of grid */
 0      /* YL = largest Y value of grid */
```

```
0.2     /* DEL = anistropy weighting (0, <1, >1) 0.2 = weight to X gradients
                 (used in velocity sections) */
0.5     /* CAY = interpolation type (0.0 to inf) 0.5 = half and half */
0       /* NRNG = interpol. range for missing data (in units of grid cells) */
1       /* NSM = no. of Laplacian smoothings

  ------------------------------------------------------------------------
   INPUT DATA FORMAT: * = first three columns = X-data, Y-data, contour-data
(E9.4,E8.1,E11.3,11X)
(E9.4,E8.1,11X,E11.3)
  ------------------------------------------------------------------------*/
(E9.4,E8.1,E11.3,11X)

/*-----------------
   PLOT PARAMETERS
  -----------------*/
1.0     /* SCALE = scale factor */
1.0     /* X1PL = lower left corner X-coordinate (in inches from page left) */
2.5     /* Y1PL = lower left corner Y-coordinate (in inches from page bottom) */
  5     /* XLPL = lower right corner X-coord. (X-scale * abs(X-range) + X1PL) */
6.5     /* YLPL = upper left corner Y-coord. (Y-scale * abs(Y-range) + Y1PL) */
.067    /* HGTPT = height of labels marking data (in inches) */
.08     /* HGTC = height of labels on contours (in inches) */
-3      /* LABPT = data point labeling option (-3, -2, -1, >=0) -3 = omit */
1       /* NDIV = no. of times grid is subdivided (0 to 2) */
3       /* NARC = no. of segments each contour is subdivided into (1 to 10) */
1.0     /* XTTL = title location X-coordinate (usually equal to X1PL) */
6.7     /* YTTL = title location Y-coordinate (usually equal to YLPL + 0.2) */
-0.14   /* THT = height of title characters (in inches; put '-' for boldface) */
0.0     /* TANG = title rotation, 0 = horizontal title) */

/*------------------------------------------------------------------------
   CONTOUR LEVELS:
   column 1 = contouring intervals: check for sufficient range and step size
   column 2 = label option (-3 = omit label,
                            -1 = omit decimal point,
                           >=0 = number of places after decimal point)
   column 3 = contour line option (1 = ordinary, 2 = heavy, 3 = dotted)
   column 4 = contour level option (1 = auto-calculate, 0 = all user-supplied)
   NOTE: if shading is requested, boundary interval must be on first line
  ------------------------------------------------------------------------*/
   0, -1, 2, 0
-120, -1, 1, 0
-100, -1, 1, 0
 -80, -1, 1, 0
```

```
 -60, -1, 1, 0
 -40, -1, 1, 0
 -20, -1, 1, 0
 -10, -1, 1, 0
  10, -1, 1, 0
  20, -1, 1, 0
  40, -1, 1, 0
  60, -1, 1, 0
  80, -1, 1, 0
 100, -1, 1, 0
 120, -1, 1, 0
 1.E35/

/*-----------------------------
   AXES ANNOTATION PARAMETERS:
  -----------------------------*/
5       /* NTICX = no. of tick marks along X-axis : remember to add one */
9      /* NTICY = no. of tick marks along Y-axis : remember to add one */
2       /* NDENX = density of X-axis labels (>= 0) _\ 0 = no labels           */
2       /* NDENY = density of Y-axis labels (>= 0)  / n = every nth tick mark */
0       /* NSKPX = no. of X-axis tick marks to skip before starting to label  */
0       /* NSKPY = no. of Y-axis tick marks to skip before starting to label  */
-1      /* NDECX = no. of decimal places on X-axis nos., -1 = no dec. point */
-1      /* NDECY = no. of decimal places on Y-axis nos., -1 = no dec. point */
2       /* IOPTX = special labeling option,    >3 = julian date (X-axis only)
                                           __\  1 = normal, 0 = no minus sign */
0       /* IOPTY = special labeling option   /  2 = latitude, 3 = longitude   */
0.1     /* TICL = length of axes tick marks (in inches) */
-0.1    /* HNUM = height of axes numbers (in inches, put '-' for boldface */
-0.11   /* HLAB = height of axes labels (in inches, put '-' for boldface */
8       /* NXCHAR = no. of chars. in X-axis label (0 = no label, max. = 81) */
9       /* NYCHAR = no. of chars. in Y-axis label (0 = no label, max. = 81) */
Latitude /* X-axis label if NXCHAR not 0 (format 20A4) */
Depth (m) /* Y-axis label if NYCHAR not 0 (format 20A4) */

/*-----------------
   END-OF-JOB CARD
  -----------------*/
  8*0
```

A.9 edfix.cnt

```
/*****************************************************************************

  FILE:  edfix.cnt

         This control file is used with the edfix program
         to edit out bad fixes from a text file of GPS and/or
         Transit fixes according to certain quality parameters,
         and to optionally merge Transit and GPS fixes into one file.

  INPUT:  if editing GPS fixes, text file of GPS fixes
          if editing Transit fixes, text file of Transit fixes
          if merging while editing, both Transit and GPS fix files
  OUTPUT: text file of fixes with bad ones commented out (% in first column)
-----------------------------------------------------------------------------
  CONTROL FILE STRUCTURE:

  output:              < output filename >
  transit_file:        < input Transit fix filename | none >
      min_elevation=   < min. allowed satellite elevation in degrees >
      max_elevation=   < max. allowed satellite elevation in degrees >
      max_iterations=  < max. allowed no. of iterations in calculation >
      max_dr=          < max. distance to dead reckoned position >
      time_since_gps=  < max. proximity to last GPS fix in decimal days >
  gps_file:            < input GPS fix filename >
      max_hdop=        < max. allowed horizontal dilution of precision >
      time_since_3=    < max. proximity to last 3-satellite fix, decimal days >
-----------------------------------------------------------------------------*/

  output:              ademo.edf
  transit_file:        none
      min_elevation=   7
      max_elevation=   70
      max_iterations=  3
      max_dr=          4.0
      time_since_gps=  0.006
  gps_file:            ademo.gps
      max_hdop=        6
      time_since_3=    0.25

/*****************************************************************************/
```

A.10 fix_temp.cnt

```
/*****************************************************************************
     This control file is used with the fix_temp program
     to correct the transducer temperature for profiles within the
     given time range(s) in a CODAS database (ancillary1.tr_temp).
     The temperature correction may be specified either
     as a constant offset to be added to the original value, or
     the correct temperature itself, replacing the original.

     It also recalculates all variables that have been
     calculated from the transducer temperature:
     ancillary_1.snd_spd_used, U, V, W, and error velocities.

 INPUT:  CODAS database
 OUTPUT: CODAS database with corrected temperature, etc.

   true_temperature = original_temp + offset_temperature

  will be applied in velocity correction, where original_temp
  is the temperature original recorded in the ping data file.

-----------------------------------------------------------------------------
 CONTROL FILE STRUCTURE:

 dbname:                < CODAS database name >
 original_soundspeed=   < soundspeed that was input to the DAS if DAS was
                 not set to calculate soundspeed from temperature >
                { set to 0 otherwise }
 offset_temperature=    < no. of degrees to add to original temperature >
                { set to 0 if using true_temperature }
 true_temperature=      < value to store in place of original >
                { used only when offset_temperature is set to 0 }
 true_salinity=
 end                    { this 'end' must precede the time_ranges: list }
 time_ranges:           { list of YMDHMS time pairs }
  < yy/mm/dd hh:mm:ss > to < yy/mm/dd hh:mm:ss >
   .
   .
-----------------------------------------------------------------------------*/

 dbname:  ../adcpdb/ademo
 original_soundspeed=  1500.0
 offset_temperature=  -1.3512    /* using CTD to correct the ADCP record */
 true_salinity= corrsal.dat      /* interpolated from CTD stations */
 end
 time_ranges:
 93/04/09 00:02:00 to 93/04/10 23:58:00
/*****************************************************************************/
```

A.11 getnav.cnt

```
/**************************************************************************

  FILE:  getnav.cnt

         This control file is used with the getnav program
         to retrieve the navigation information for selected profiles
         within the given time range(s) from a CODAS ADCP database.

  INPUT:  CODAS database
  OUTPUT: text file of profile times and positions
--------------------------------------------------------------------------
  CONTROL FILE STRUCTURE:

  dbname:          < CODAS ADCP database name >
  output:          < output filename >
  step_size:       < profile sampling rate >
  year_base=       < base year for decimal day conversions >
  time_ranges:     { list of YMDHMS time pairs }
    < yy/mm/dd hh:mm:ss > to < yy/mm/dd hh:mm:ss >
    .
    .
--------------------------------------------------------------------------*/

  dbname:           ../adcpdb/ademo
  output:           ademo.fix
  step_size:        1
  year_base=        1993
  time_ranges:
    93/04/09 00:02:00 to 93/04/10 23:58:00

/**************************************************************************/
```

A.12 llgrid.cnt Control File Structure

A.12.1 Part 1 of 2

```
/*****************************************************************************

 FILE:  llgrid.cnt

     This control file is used with the llgrid program
     to generate a text file of time ranges that roughly correspond to
     longitude/latitude grid crossings.  The grid is specified as a
     starting position and an increment for each dimension.

 INPUT:  CODAS database
 OUTPUT: text file of time ranges which can be appended to an adcpsect
      control file to extract profile data for the desired sections
-----------------------------------------------------------------------------
 CONTROL FILE STRUCTURE:

 dbname:       < CODAS database name >
 output:       < output filename >
 step_size:    < profile sampling rate >                { 1 = every profile }
 latitude
  origin:     < decimal degrees >
  increment:  < decimal degrees >
 longitude
  origin:     < decimal degrees >
  increment:  < decimal degrees >
 time_ranges:  { list of YMDHMS time pairs }
  < yy/mm/dd hh:mm:ss > to < yy/mm/dd hh:mm:ss >
  .
  .
  .
-----------------------------------------------------------------------------*/
```

A.12.2 Part 2 of 2

```
/* Primary parameters for llgrid.cnt */
 dbname:        ../adcpdb/ademo
 output:        ademoall.llg
 year_base:     1993
 step_size:     1

/* Several examples of settings are shown for the lat/long options */

/* Options as used in preparation for vector plots.  In this case,
  a large geographical area will be covered so the increment is
  large; typically the increment is set to 0.10 for spatial
  domains of roughly 30 by 30 degrees. */
 lat_origin:    -0.25   /* center bins */
 lat_increment:  0.50   /* grid by 5/10 degree latitude */
 lon_origin:    -0.25
 lon_increment:  0.50   /* and 5/10 degree longitude */

/* Options as used in preparation for a meridional contour plot. */
 lat_origin:    -0.05   /* center bins */
 lat_increment:  0.10   /* grid by 1/10 degree latitude */
 lon_origin:    100000
 lon_increment:  0.10   /* and 1/10 degree longitude */


/* Options as used in preparation for a zonal contour plot. */
 lat_origin:  1000000   /* center bins */
 lat_increment:  0.10   /* grid by 1/10 degree latitude */
 lon_origin:    -0.05
 lon_increment:  0.10   /* and 1/10 degree longitude */

/* Note in the two examples above, the axis which is held
  constant is deactivated by setting the origin to a large
  number  */

  time_ranges:
  93/04/09 00:02:00 to 93/04/10 23:58:00
```

A.13 loadping.cnt

```
/*****************************************************************************

  FILE:  loadping.cnt

         This control file is used with the loadping program
         to load ADCP data from ping files into a CODAS database.

  INPUT:  1. ping file(s)
          2. producer definition file
  OUTPUT: 1. CODAS ADCP database (.blk files)
          2. log file, if requested
  NOTE:   1. It is alright to load the ping files incrementally into the same
             CODAS database, running loadping to load a few of them at a time.
          2. Make sure that the case does not arise where a CODAS block file
             has a time range that overlaps another block file (this occurs
             when the ping files are not listed/loaded in chronological order
             and both NEW_BLOCK_AT_FILE? and NEW_BLOCK_AT_HEADER? are set to
             'no').  In such cases, searching by time for profiles that fall
             within the overlap will check the earlier of the overlapping
             blocks ONLY.
-----------------------------------------------------------------------------
  CONTROL FILE STRUCTURE:

  DATABASE_NAME:            < CODAS database name >
  DEFINITION_FILE:          < input producer definition file >
  OUTPUT_FILE:              < output filename | none >
  MAX_BLOCK_PROFILES:       < maximum no. of profiles per block <= 400 >
  NEW_BLOCK_AT_FILE?        < yes | no > { start new block for each ping file }
  NEW_BLOCK_AT_HEADER?      < yes | no > { start new block for each header }
  NEW_BLOCK_TIME_GAP(min):  < n >  { start new block when gap > n minutes }
                                   { n must be <= 32767 }
  PINGDATA_FILES:           { list pingdata files from next line }
    < input ping filename > { followed by 0 or more instances of the ff.: }
        [ time_correction:
            start_header_number:  < header # to start time correction >
            correct_time:         < correct GMT time >
            PC_time:              < PC recorded time >
            clock_rate:           < PC-to-GMT clockspeed ratio > ]
        [ skip_header_range:      < header # > to < header # > ] { don't load }
        [ skip_profile_range:
            hdr=                  < header # >
            prof=                 < profile # > to < profile # > ]
            end
    .
    .
    .
-----------------------------------------------------------------------------*/

  DATABASE_NAME:            ../adcpdb/ademo
  DEFINITION_FILE:          ../adcpdb/ademo.def
  OUTPUT_FILE:              ademo.lod
  MAX_BLOCK_PROFILES:       400
  NEW_BLOCK_AT_FILE?        yes
  NEW_BLOCK_AT_HEADER?      no
  NEW_BLOCK_TIME_GAP(min):  30

  PINGDATA_FILES:
../ping/pingdata.000
  skip_header_range:        7 to 8
  skip_header_range:        75 to 76
  skip_header_range:        290 to 291
  end
../ping/pingdata.001
  end
```

A.14 lst_btrk.cnt

```
/*******************************************************************************

 FILE:  lst_btrk.cnt

      This control file is used with the lst_btrk program
      to extract the bottom track velocity data within the
      specified time range(s) from a CODAS database into a text file.

 INPUT:  CODAS database
 OUTPUT: text file with the following columns:
      1. profile time (in decimal days)
      2. zonal ship velocity over the ground (in m/s)
      3. meridional ship velocity over the ground (in m/s)
      4. bottom depth (in meters)
      An entry with just a '%' in the first column signifies a gap of
      one or more profiles for which no bottom track data were available.
-------------------------------------------------------------------------------
 CONTROL FILE STRUCTURE:

 dbname:       < CODAS database name >
 output:       < output filename >
 step_size=    < number of profiles to advance >
 year_base=    < base year for decimal days >
 time_ranges:  { list of time ranges to process }
  < yy/mm/dd hh:mm:ss > to < yy/mm/dd hh:mm:ss >
  .
  .
  .
-------------------------------------------------------------------------------*/

 dbname:     ../../adcpdb/ademo
 output:     ademo.btm
 step_size=  1
 year_base=  1993
 time_ranges:
  93/04/09 00:02:00 to 93/04/10 23:58:00

/*******************************************************************************/
```

A.15 lst_hdg.cnt

```
/*****************************************************************************

 FILE:  lst_hdg.cnt

      This control file is used with the lst_hdg program
      to extract the ship's mean heading and last heading
      from a CODAS database into a text file.

 INPUT:  CODAS database
 OUTPUT: text file with the following columns:
      1. profile time (in decimal days)
      2. ship's mean heading over the ensemble (in decimal degrees)
      3. ship's last heading for the ensemble (in decimal degrees)
-----------------------------------------------------------------------------
 CONTROL FILE STRUCTURE:

 dbname:       < CODAS database name >
 output:       < output filename >
 step_size=    < number of profiles to advance >
 year_base=    < base year for decimal day conversions >
 time_ranges:  { list of YMDHMS time pairs }
  < yy/mm/dd hh:mm:ss > to < yy/mm/dd hh:mm:ss >
  .
  .
  .
-----------------------------------------------------------------------------*/

 dbname:      ../../adcpdb/ademo
 output:      ademo.hdg
 step_size=   1
 year_base=   1993
 time_ranges:
  93/04/09 00:02:00 to 93/04/10 23:58:00

/*****************************************************************************/
```

A.16 lst_prof.cnt

```
/*****************************************************************************

 FILE:  lst_prof.cnt

      This control file is used with the lst_prof program
      to produce a list of times and positions of selected
      profiles within the given time range(s) from a CODAS database.

 INPUT:  CODAS database
 OUTPUT: text file of profile times and positions
-----------------------------------------------------------------------------
 CONTROL FILE STRUCTURE:

 dbname:       < CODAS database name >
 output:       < output filename >
 step_size:    < number of profiles to advance >
 time_ranges:  { list of time ranges to select from }
   < yy/mm/dd hh:mm:ss > to < yy/mm/dd hh:mm:ss >
   .
   .
   .
-----------------------------------------------------------------------------*/

 dbname:     ademo
 output:     ademoprf.lst
 step_size:  1
 time_ranges:
   93/04/09 00:02:00 to 93/04/10 23:58:00

/*****************************************************************************/
```

A.17 mkblkdir.cnt

```
/*****************************************************************************

 FILE:  mkblkdir.cnt

      This control file is used with the mkblkdir program
      to generate a new CODAS database from a set of
      CODAS block files.  The new database will consist of
      a new block directory file and a copy of each input
      block file, converted if necessary to the host machine
      format.  Note that the list of input block files should
      NOT include the directory block file.

 INPUT:  1. CODAS block files (*.blk)
       2. producer definition file
       3. structure definition file, if needed (that is, if the
         data structures in the block files are not defined therein)
 OUTPUT: CODAS database
-----------------------------------------------------------------------------
 CONTROL FILE STRUCTURE:

 DB_NAME       < output CODAS database name >
 PRD_NAME      < input producer definition filename >
 [ DESTINATION < PC_COMPATIBLE_HOST | VAXD_COMPATIBLE_HOST | SUN3_COMPABTIBLE_
HOST > ]
 end
 [ SD_NAME     < input structure definition filename > ]
 [ BLOCK_FILE  < input CODAS block filename > ]
 .
 .
 .
-----------------------------------------------------------------------------*/

 DB_NAME    new/a9103
 PRD_NAME   codas3/demo/adcpdb/codas3.def
 end
 BLOCK_FILE codas3/demo/adcpdb/a9103001.blk
 BLOCK_FILE codas3/demo/adcpdb/a9103002.blk
 BLOCK_FILE codas3/demo/adcpdb/a9103003.blk

/*****************************************************************************/
```

A.18 nmea_gps.cnt

```
/*******************************************************************************

  FILE:  nmea_gps.cnt

         This control file is used with the nmea_gps program
         to convert NMEA-formatted GPS fix files to a columnar
         text format suitable for use with the codas3/adcp/nav
         programs, and/or to Mat-file format for Matlab.  In the
         latter case, the data are edited according to the
         given criteria prior to output.

  INPUT:  NMEA-formatted GPS fix files
  OUTPUT: 1. if requested, text file (.gps) with the following columns:
             a. fix time, in decimal days
             b. longitude, in decimal degrees
             c. latitude, in decimal degrees
             d. no. of satellites used
             e. quality index
             f. horizontal dilution of precision (HDOP)
             g. 0, since no dopN
             h. 0, since no dopE
             i. 0, since no dopV
             j. altitude
          2. if requested, one or more Mat-files (.mat or .m?? if more than one)
             with the following arrays:
             a. fix time, in decimal days
             b. longitude, in decimal degrees
             c. latitude, in decimal degrees
             d. no. of satellites used, if requested
             e. horizontal dilution of precision, if requested
-------------------------------------------------------------------------------
  CONTROL FILE STRUCTURE:

  [ YMD_BASE:            < yy/mm/dd for decimal day conversions > ] { 00/01/01 }
  [ TIME_RANGE:          < ALL | yy/mm/dd hh:mm:ss to yy/mm/dd hh:mm:ss > ]
  [ OUTPUT_ASCII_FILE:   < none | root for output ASCII filename > ]
  [ OUTPUT_MAT_FILE:     < none | root for output Mat-filename >
    [ max_HDOP=            < max. allowed HDOP for Mat-file output > ]
    [ subsamples=          < fix sampling rate for Mat-file output > ]
    [ max_mat_array_size=  < maximum no. of fixes per Mat-file > ]
                         { only needed for PC-Matlab (no virtual memory) }
    [ SAVE_NUM_OF_SAT      { output no. of satellites info. to Mat-file } ]
    [ SAVE_HDOP            { output HDOP info. to Mat-file } ]
    [ GAP_INCLUDED         { output bad fixes to Mat-file as NaNs } ]]

  end                    { must always precede INPUT_GPS_FILES: }

  INPUT_GPS_FILES:       { list in chronological order }
    < NMEA-formatted GPS fix filename >
    .
    .
    .
-------------------------------------------------------------------------------*/

  YMD_BASE:             91/01/30
  TIME_RANGE:           91/01/30 22:54:45 to 91/01/30 22:55:00
  OUTPUT_ASCII_FILE:    nmeagps1
  OUTPUT_MAT_FILE:      nmeagps1
    subsamples=         50
    max_HDOP=           6
    SAVE_HDOP
    SAVE_NUM_OF_SAT
    GAP_INCLUDED
  end

  INPUT_GPS_FILES:
    tw17an00.c0

/*******************************************************************************/
```

A.19 profstat.cnt Control File Structure

A.19.1 Part 1 of 5

```
/*****************************************************************************

  FILE:  profstat.cnt

       This control file is used with the profstat program
       to generate statistics on selected profiles in a CODAS database.

 INPUT:  CODAS database
 OUTPUT: 1. text file of profile statistics (.prs)
      2. Matlab file of profile statistics (.mat)
-----------------------------------------------------------------------------
 CONTROL FILE STRUCTURE:

 dbname:       < CODAS database name >
 output:       < root for output filename >
 step_size:    < number of profiles to advance >
 ndepth:       < number of depth bins to read and process>
 time_ranges:  < combined | separate >

 { one or more variable names, followed by various arguments as shown }
 [ < variable name >
   < "label" >                    { in quotes, for labelling the output }

   [ reference:   < start bin > to  < end bin > ]
   [ difference:  < 0 | 1 | 2 > ]     { for no, first, second difference }
   [ statistics:  scale= < n > ]  { 0.01 for output in cm, 1.0 = m, etc. }
   [ histogram:
     nbins=     < n >
     origin=    < n >
     increment= < n >
     style=     < h | c | n | cn > ]      { n = normalized display
                                c = cumulative display
                               cn = cumulative & normalized
                                h = neither }
   [ flag_mask:
     < bit-mask >  { bit-mask is any one or more of those defined in
     .               profmask.h; these cumulatively indicate which criteria
     .               to consider in flagging profile bins as bad prior to
     .               calculations.  Default is ALL_BITS. }
     end ]
 end

 { list of YMDHMS time pairs: }
 < yy/mm/dd hh:mm:ss > to < yy/mm/dd hh:mm:ss >
 .
 .
 .
-----------------------------------------------------------------------------*/
```

A.19.2 Part 2 of 5, Example 1 of Use

```
/***            This particular version is used to obtain a general idea
       of the degree of variability in the vertical velocity (W)
       in order to derive a threshold for flagging errant profiles.
       It is called "profst00.cnt" and is used editing ***/

 dbname:       ../adcpdb/ademo
 output:       ademodf0             /* no extension! */
 step_size:    2
 ndepth:       56
 time_ranges:  combined

 /* variables list: */
 W
  "W component"
  reference:    5 to 20
  difference:   0
  statistics:   scale= 0.01
  end
 end


 /* time range list */
 93/04/09 00:02:00 to 93/04/10 23:58:00
```

A.19.3 Part 3 of 5, Example 2 of Use

```
/***          This particular version is used to obtain second difference
      statistics on U, V, and W velocity components in order to
      derive initial thresholds for flagging errant profiles.
      This version is called "profst02.cnt" and is used in editing. ***/

 dbname:       ../adcpdb/ademo
 output:       ademodf2                /* no extension! */
 step_size:    2
 ndepth:       56
 time_ranges:  combined

 /* variables list: */
 U
  "U component"
  reference:   5 to 20
  difference:  2
  statistics:  scale= 0.01
  end
 V
  "V component"
  reference:   5 to 20
  difference:  2
  statistics:  scale= 0.01
  end
 W
  "W component"
  reference:   5 to 20
  difference:  2
  statistics:  scale= 0.01
  /* histogram:   nbins= 10 origin= -0.1 increment= 0.02 style= n */
  end
 end

 /* time range list */
 93/04/09 00:02:00 to 93/04/10 23:58:00
```

A.19.4 Part 4 of 5, Example 3 of Use

```
/***       This particular version is used to obtain statistics for
       profiles collected when the ship was on-station.  It is
       called "profstos.cnt".  The time ranges are obtained
       by running program "arrdep" which utilizes control file
       "arrdepos.cnt".  A similar profstat.cnt file would exist
       for calculating statistics for when the ship was steaming
       (not shown in this appendix, but it is similar to below).  ***/
 dbname:        ../adcpdb/ademo
 output:        ademo_os
 step_size:     1
 ndepth:        58
 time_ranges:   combined

 /* variables list: */
 W
  "W component, ADCP DEMO"
  reference:  5 to 20
  difference: 0
  statistics: scale= 0.1
  end
 AMP_SOUND_SCAT
  "AMPLITUDE, ADCP DEMO"
  difference: 0
  statistics: scale= 1.0
  histogram: nbins= 12 origin= 20  increment= 20  style= nc
  end
 PERCENT_GOOD
  "PERCENT GOOD, ADCP DEMO"
  difference: 0
  statistics: scale= 1.0
  end
 ERROR_VEL
  "ERROR VELOCITY, ADCP DEMO"
  difference: 0
  statistics: scale= 0.1
  flag_mask: ALL_BITS end
  end
 PERCENT_3_BEAM
  "PERCENT 3 BEAM, ADCP DEMO"
  difference: 0
  statistics: scale= 1.0
  end
 end
```

A.19.5 Part 5 of 5, Example 3 of Use

```
/* continued from previous example */
/* append time ranges from arrdep arrdepos.cnt output: */
/*
 reference_file: ../nav/ademo.nav
 output_file: ademo_os.arr
 year_base= 1993
 n_refs= 13
 i_ref_l0= 1
 i_ref_l1= 5
 i_ref_r0= 7
 i_ref_r1= 11
 range: on_station
 up_thresh= 2.500000
 down_thresh= 2.500000
 margin= 10.000000*/
93/04/09  10:52:22  to  93/04/09  11:12:42
 /*    98.45315 to    98.46704;  1200 s, depart */
   .        .
   .        .
93/04/10  07:40:50  to  93/04/10  08:01:11
 /*    99.32014 to    99.33404;  1201 s, depart */

/*****************************************************************************/
```

A.20 putnav.cnt

```
/*****************************************************************************

  FILE:  putnav.cnt

         This control file is used with the putnav program
         to store the calculated profile positions and
         ship velocity into a CODAS database.

  INPUT:  CODAS database
          text file of estimated positions and ship velocity for each profile
  OUTPUT: CODAS database with profile positions and ship velocity
-----------------------------------------------------------------------------
  CONTROL FILE STRUCTURE:

  dbname:         < CODAS database name >
  position_file:  < input text file of positions, etc. > { output of smoothr }
  year_base=      < base year for decimal days >
  tolerance=      < max. allowed discrepancy between profile & fix time, sec >
  navigation_sources:  { one or more of the following }
    [ gps ]
    [ transit ]
    [ loran ]
    [ omega ]
    [ radar ]
    end
-----------------------------------------------------------------------------*/

  dbname:         ../adcpdb/ademo
  position_file:  ademo.sm
  year_base=      1993
  tolerance=      5
  navigation_sources:
    gps
    end

/*****************************************************************************/
```

A.21 refabs.cnt

```
/*****************************************************************************

  FILE:  refabs.cnt

         This control file is used with the refabs program
         to calculate the raw absolute reference layer velocities.
         Averaged between fixes, this is the difference between the
         velocity of the ship over the ground, determined from the fixes,
         and the velocity of ship relative to the reference layer velocity,
         from the ADCP profiles.

  INPUT:  text file of ship velocity relative to reference layer (adcpsect .nav)
          text file of position fixes with bad fixes commented out
  OUTPUT: text file with the following columns:
          1. fix time (in decimal days)
          2. longitude (in decimal degrees)
          3. latitude (in decimal degrees)
          4. zonal component of absolute reference layer velocity (in m/s)
          5. meridional component of absolute reference layer velocity (in m/s)
          6. fix interval (in minutes)
          7. zonal velocity cumulated over gap (in m/s)
          8. meridional velocity cumulated over gap (in m/s)
          9. length of gap (in minutes)
-----------------------------------------------------------------------------
  CONTROL FILE STRUCTURE:

  fix_file_type:    < simple | HIG >              { format of input fix file }
  reference_file:   < input file of relative ship velocity >  { adcpsect .nav }
  fix_file:         < input file of position fixes >         { time, lon, lat }
  output:           < output filename >
  year_base=        < base year for decimal days >

  ensemble_length=  < no. of seconds in ensemble >
  gap_tolerance=    < maximum seconds discrepancy between profile & fix time >
                                              { allow a little, perhaps 10 }
-----------------------------------------------------------------------------*/

  fix_file_type:    simple
  reference_file:   ademo.nav
  fix_file:         ademo.ags
  output:           ademo.ref
  year_base=        1993

  ensemble_length=  300                     /* 5-minute ensemble */
  gap_tolerance=    10

/*****************************************************************************/
```

A.22 refabsbt.cnt

```
/*****************************************************************************
     This control file is used with the refabsbt program
     to calculate ship displacement from both fix data and
     bottom track velocity data in a CODAS ADCP database,
     as averaged between consecutive fixes.

 INPUT:  1. text file of navigation fixes, already edited
        (as extracted from CODAS database by ubprint then edited)
      2. text file of bottom track velocities
        (as extracted from CODAS database by lst_btrk)
 OUTPUT: text file with the following columns:
      1. time of the second fix (in decimal days)
      2. zonal displacement between consecutive fixes (in meters)
      3. meridional displacement between consecutive fixes (in meters)
      4. zonal displacement calculated from bottom track (in meters)
      5. meridional displacement calculated from bottom track (in meters)
      6. zonal velocity cumulated over the gap
      7. meridional velocity cumulated over the gap
      8. length of gap (in minutes)
-----------------------------------------------------------------------------
 CONTROL FILE STRUCTURE:

 fix_file_type:    < simple | HIG >               { format of input fix file }
 reference_file:   < input bottom track velocity file from lst_btrk >
 fix_file:         < input file of position fixes >         { time, lon, lat }
 output:           < output filename >
 year_base=        < base year for decimal day conversions >

 ensemble_length=  < no. of seconds in ensemble >              { usually 300 }
 gap_tolerance=    < maximum seconds discrepancy between profile & fix time >
                            { allow a little, perhaps 60 }
-----------------------------------------------------------------------------*/

 fix_file_type:    simple
 reference_file:   ademorot.btm
 fix_file:         ../../nav/ademo.ags
 output:           ademo.ref
 year_base=        1993

 ensemble_length=  300
 gap_tolerance=    60

/*****************************************************************************/
```

A.23 rotate.cnt

```
This control file is used with the rotate program to rotate and simultaneously
apply an amplitude correction factor to the water and bottom track velocities
in a CODAS database.  It allows different rotation parameters to be used for
different profile ranges, and for the water and bottom track velocities.
It also allows the rotation to be performed using either a constant angle or
one relative to the ship's heading.  The angle is specified counterclockwise
from the gyro compass forward axis (which should be aligned with the ship's
keel) to the transducer forward axis.
 INPUT:  CODAS database
 OUTPUT: 1. updated CODAS database
      2. log file record of rotation run
--------------------------------------------------------------------------------
 CONTROL FILE STRUCTURE:
 DB_NAME:      < CODAS database name >
 LOG_FILE:     < output log filename -- appended to if already existing >
 { followed by one or more of the following _RANGE & OPTION_LIST specs: }
 < BLOCK_RANGE: | TIME_RANGE: | DAY_RANGE: > < all | <start> to <end> >
  OPTION_LIST:
   < water_track: | bottom_track: | water_and_bottom_track: >
         [ time_0=         < mean time in decimal days > ]
         [ time_angle_file: < file of ADCP profile times and angles >
         [ year_base=      < base year for time_0 > ]
    [ angle_0=        < constant angle in degrees > ]
    [ angle_1=        < coefficient of linear time term > ]
    [ angle_2=        < coefficient of squared time term > ]
    [ angle_3=        < coefficient of cubic time term > ]
    [ angle_sin_H=    < coefficient for sin of ship heading > ]
    [ angle_cos_H=    < coefficient for cos of ship heading > ]
    [ amplitude=      < amplitude of rotation > ]
    [ rotate_only! ]  { otherwise, above values will be stored in database }
    [ store_only! ]   { otherwise, velocities will be rotated }
    [ unrotate! ]     { totally unrotate velocities }
    end               { of rotation parameter option list }
   .
   end                 { of track list }
 .                       { more BLOCK_ or TIME_ or DAY_RANGE: to process }
--------------------------------------------------------------------------------*/
 DB_NAME:                          ../../adcpdb/ademo
 LOG_FILE:                         ademo.rot
 TIME_RANGE:                       all
  OPTION_LIST:
   water_and_bottom_track:
    /* time_angle_file:            gpscal.ang */ /* step 1:  gyro correction */
    amplitude=                  1.00             /* step 2:  transducer offset */
    angle_0=                    1.8              /* in degrees */
    end
   end
```

A.24 scanping.cnt

```
/*****************************************************************************

  FILE:  scanping.cnt

         This control file is used with the scanping program
         to scan ADCP ping files and generate a list of profile times
         (and fix times if available), variables recorded (if requested),
         and bad headers and profiles encountered while scanning.

  INPUT:  1. ping file(s)
          2. user buffer definition file, if requesting user buffer output and
             type is not ascii, 1020, 1021, 1280, 1281, 1320, 720 or 2240,
             or program is not being run on a PC
  OUTPUT: 1. text file listing headers, profiles, variable names
          2. if user buffer is type 1020, 1280, 1281 or 1320, and
             user buffer output is requested, a text file summary of
             Transit fixes (.trs):
             a. fix time (in decimal days)
             b. longitude
             c. latitude
             d. satellite elevation
             e. number of iterations
             f. distance to dead-reckoned position
             g. flag indicating if fix was accepted by navigator (1 = true)
             h. difference between PC and satellite time (in seconds)
          3. if user buffer is type ASCII or other and user buffer output
             is requested, a text file of the user buffer contents
-----------------------------------------------------------------------------
  CONTROL FILE STRUCTURE:

  OUTPUT_FILE:       < log output filename >
  SHORT_FORM:        < yes | no >  { yes = list vars. recorded for each header }
  UB_OUTPUT_FILE:    < user buffer output filename | none >
  USER_BUFFER_TYPE:  < none | 1280 | 1020 | 1320 | 1281 | 1021 | 720 | 2240 | ascii | other >
  UB_DEFINITION:     < user buffer definition file name | none >
  PINGDATA_FILES:
    < input ping filename >
    .
    .
    .
-----------------------------------------------------------------------------*/

  OUTPUT_FILE:         ademo.scn
  SHORT_FORM:          yes
  UB_OUTPUT_FILE:      none
  USER_BUFFER_TYPE:    720
  UB_DEFINITION:       ub_720.def
  PINGDATA_FILES:
../ping/pingdata.000
../ping/pingdata.001
/*****************************************************************************/
```

A.25 set_top.cnt

```
/**************************************************************************

 FILE:  set_top.cnt

     This control file is used with the set_top program
     to indicate the topmost bin for which data are considered good
     for profiles when the ship was underway,
     within the specified time_range(s) in a CODAS ADCP database.

 INPUT:  CODAS ADCP database
 OUTPUT: updated CODAS ADCP database
      (access_variables.first_good_bin reset to new value)
--------------------------------------------------------------------------
 CONTROL FILE STRUCTURE:

 dbname:           < input CODAS database name >
 speed_threshold:  < underway limit in m/s >
 ref_bin_range:    < start bin > to < end bin > { ref. layer for underway test }
 first_good_bin:   < first good bin number >
 time_ranges:      { list of time ranges to process }
  < yy/mm/dd hh:mm:ss > to < yy/mm/dd hh:mm:ss >
  .
  .
--------------------------------------------------------------------------*/

 dbname:            prc4
 speed_threshold:   2.0
 ref_bin_range:     2  to  10
 first_good_bin:    2
 time_ranges:
  88/04/23 01:32:45   to  88/04/23 04:17:45

/**************************************************************************/
```

A.26 smoothr.cnt

```
/*****************************************************************************

  FILE:  smoothr.cnt

         This control file is used with the smoothr program
         to interpolate and smooth the absolute reference layer velocities
         (derived from the refabs program) to the profile times.
         It uses a Blackman window function of width T:
         w(t) = 0.42 - 0.5 * cos(2 * pi *t / T) + 0.08 * cos(4 * pi * t / T).
         It also calculates profile positions from the smoothed velocities.

  INPUT:  1. text file of raw absolute reference layer velocities
          2. text file of ship velocity relative to reference layer
  OUTPUT: 1. text file of absolute ship velocities and positions
             interpolated and smoothed to the profile times
          2. binary file (.bin) with same information as text file for use
             with Matlab plotting routine 'callrefp.m'
          3. log file (.log)
-----------------------------------------------------------------------------
  CONTROL FILE STRUCTURE:

  reference_file:    < input file of relative ref. layer velocities (.nav) >
  refabs_output:     < input file of raw absolute ref. layer vel. from refabs >
  output:            < text output filename >
  filter_hwidth=     < half-width of filter in decimal days >
  min_filter_fraction=  < how much of filter mass must have data: 0 to 1 >
  max_gap_ratio=     < parameter limiting gap acceptance; suggest 0.05 >
  max_gap_distance=  < distance uncertainty threshold for position calc. (m) >
                                                          { suggest 2000 }
  max_gap_time=      < gap time threshold for starting a segment (seconds) >
                                                 { suggest 3600, one hour }
  ensemble_time=     < no. of seconds in ensemble >
  max_speed=         < max. speed of ship in question (m/s) >
  min_speed=         < expected short-term speed variability on or off station >
  iterations=        < no. of times to optimize the positions:  2 should do >
  fix_to_dr_limit=   < threshold for printing fix times and deviations (deg) >
-----------------------------------------------------------------------------*/

  reference_file:       ademo.nav
  refabs_output:        ademo.ref
  output:               ademo.sm
  filter_hwidth=        0.0208333     /* half an hour */
  min_filter_fraction=  0.05
  max_gap_ratio=        0.05
  max_gap_distance=     500
  max_gap_time=         3600          /* one hour */
  ensemble_time=        300           /* 5-minute ensemble */
  max_speed=            5.0
  min_speed=            1.0
  iterations=           3
  fix_to_dr_limit=      0.00050

/*****************************************************************************/
```

A.27 timslip.cnt Control File Structure

A.27.1 Part 1 of 2

```
/*****************************************************************************

 FILE:  timslip.cnt

     This control file is used with the timslip program
     to detect ship accelerations and turns from ship velocity,
     as recorded by the ADCP, and from position fixes.  It can
     be used to derive an estimate of the time difference between
     the ADCP ensembles and the satellite time.  It also provides
     estimates of the amplitude and phase of the transducer misalignment.

 INPUT:  1. text file of ship velocity (.nav output from adcpsect)
      2. text file of position fixes
 OUTPUT: text file (.cal file) with the following columns:
      1. jump direction (+/-1 = accel./decel.; +/-2 = turn right/left)
      2. time of middle fix (in decimal days)
      3. number of ensembles used
      4. number of fixes used
      5. change in zonal ship velocity
      6. change in meridional ship velocity
      7. variance of the reference layer velocity
      8. minimized variance of the reference layer velocity
      9. estimated time difference between fix and PC time (in seconds)
     10. estimated amplitude
     11. estimated phase (in degrees countercc from gyro to transducer)
-----------------------------------------------------------------------------
 CONTROL FILE STRUCTURE:

 fix_file_type:  < simple | HIG >
 fix_file:       < input file of position fixes >
 reference_file: < input file for ship velocity >
 output_file:    < output filename >

 year_base=      < year base for decimal day calculations >
 min_n_fixes=    < minimum number of fixes for calculation
            of time shift to proceed > { normally the same as n_refs= }
 n_refs=         < number of ensembles used in calculation of time shift >

           { The jump in velocity will be between indices (n_refs-1)/2
             and (n_refs-1)/2 +1.  For example, it will be between 6
             and 7 if n_refs is 13.  Note that all these indices start
             from 0, C-style, not from 1.  However, the lowest index
             specified must be >= 1, since the time of the start of
             ensemble 1 is (approximately) the time recorded with
             ensemble 0. }
```

A.27.2 Part 2 of 2

```
 i_ref_l0=       < index of first ensemble to use before jump > { min. is 1 }
 i_ref_l1=       < index of last ensemble before jump >
 i_ref_r0=       < index of first ensemble after jump >
 i_ref_r1=       < index of last ensemble after jump >

 up_thresh=      < about 1/2 underway speed > { for jump detection (m/s) }
 down_thresh=    < about 1/2 underway speed > { for jump detection (m/s) }
 turn_speed=     <2>  { min. speed (m/s) for turn detection }
 turn_thresh=    <60> { magnitude of course difference (deg) for turn detect. }

 dtmax=          < ensemble length + some >  { max. seconds between ensembles,
                                  to screen gaps }
 tolerance=      <5.e-5> { days (about 5 secs) tolerance for iterative timslip calc. }
 grid:           < fix | ensemble >     { calculate reference layer velocities
                           averaged between fix times,
                           or over ensemble intervals }
 use_shifted_times?   < yes | no | fixed time shift in seconds >
                 { use the best-fit time shift in the calibration }
-------------------------------------------------------------------------------*/

 fix_file_type:      simple
 fix_file:           ../../nav/ademo.ags

 reference_file:     ademorot.nav /* ademoraw.nav */
 output_file:        ademo_9r.cal

 year_base=          1993
 min_n_fixes=        9 /* 5 7 9 */


 n_refs=             9 /* 5 7 9 */

 i_ref_l0=           1
 i_ref_l1=           3 /* 1 2 3 */
 i_ref_r0=           6 /* 4 5 6 */
 i_ref_r1=           8 /* 4 6 8 */

 up_thresh=          3.0          /* m/s */
 down_thresh=        3.0          /* m/s */
 turn_speed=         2.0          /* m/s */
 turn_thresh=        60           /* degrees */

 dtmax=              360          /* seconds, for 300-second ensembles */
 tolerance=          5.e-5        /* days, about 5 seconds */
 grid:               ensemble

 use_shifted_times?  no
```

A.28 timegrid.cnt

```
/*****************************************************************************

 FILE:  timegrid.cnt

      This control file is used with the timegrid program
      to break up a given time range into smaller time ranges
      of a specified interval.

 INPUT:  none
 OUTPUT: text file of time ranges that can be appended to an adcpsect
       control file to extract profile data for the desired sections
-----------------------------------------------------------------------------
 CONTROL FILE STRUCTURE:

 output:         < output filename >
 time_interval:  < in minutes >
 time_range:
  < yy/mm/dd hh:mm:ss > to < yy/mm/dd hh:mm:ss >
-----------------------------------------------------------------------------*/

 output: ademo_os.tmg
 time_interval: 60
 time_range:
  93/04/09  00:02:00 to 93/04/10  07:36:00      /* on-station */
  93/04/10  07:36:01 to 93/04/10  23:58:00       /* southeastbound */

/*****************************************************************************/
```

A.29 ubprint.cnt

```
/*****************************************************************************

   FILE:  ubprint.cnt

          This control file is used with the ubprint program
          to extract the fix information stored with the ADCP profiles
          by the user-exit program in the user buffer and navigation
          structures.  It recognizes several versions of the user buffer:
          1020 & 1021, 1280 & 1281, 1320, 2240 & 720.

  INPUT:  CODAS database
  OUTPUT: one or more text files, one for each variable requested
-----------------------------------------------------------------------------
  CONTROL FILE STRUCTURE:

  dbname:                < CODAS database name >
  output:                < root for output filename >
  step_size=             < number of profiles to advance >
  year_base=             < base year for decimal days >
  variables:             { one or more of the following, depending on type }
    [ complete ]         { .ub = all user buffer information }
    [ TRANSIT_fix ]      { .trf = complete Transit fix information }
    [ TRANSIT_summary ]  { .trs = columnar output of Transit fixes }
                         { following option available only for type >= 1280 & 720 }
    [ GPS_summary ]      { .gps = GPS fixes at end of each ensemble }
                         { following option available only for type >= 1320 & 720 }
    [ avg_GPS_summary ]  { .ags = average GPS fix at end & start of next ens. }
                         { following options available only for type = 1320 }
    [ GPS_cal ]          { .gcl = .ags + ave. ship-ref. velocity for calib. }
    [ L_fix ]            { .lfx = GPS fixes at ensemble start, middle & end }
                         { following option available only for type = 720 & 2240 }
    [ GPS_fix ]          { .gfx = all GPS fixes (usu. 1 at start & 1 at end) }
                         { following option available only for type = 2240 }
    [ raw2_message ]     { .mg2 = raw message 2 }
    end
  time_ranges:
    < yy/mm/dd hh:mm:ss > to < yy/mm/dd hh:mm:ss >
    .
    .
    .
-----------------------------------------------------------------------------*/

  dbname:        ../adcpdb/ademo
  output:        ademo
  step_size=     1
  year_base=     1993
  variables:
    avg_GPS_summary
    end
  time_ranges:
    93/04/09 00:02:00 to 93/04/10 23:58:00

/*****************************************************************************/
```

A.30 vector.cnt

```
/*****************************************************************************
```

```
  FILE:  vector.cnt

         This control file is used with the vector program
         to generate a plot of velocity vectors, cruise track,
         and/or coastlines.

  INPUT:  if plotting velocity vectors or cruise track,
          one or more text files with the following columns:
             1. longitude (in decimal degrees)
             2. latitude (in decimal degrees)
             and velocities in the case of velocity vectors:
             3. u-component for a layer in the depth grid (in m/s)
             4. v-component for a layer in the depth grid (in m/s)
             .
             .        { more u-v pairs for other layers in the depth grid }
             .
          if plotting coastlines, a text file of map coordinates:
             1. longitude (in decimal degrees)
             2. latitude (in decimal degrees)
          The entry "-9999.0 -9999.0" separates land areas;
          the entry "-8888.0 -8888.0" separates inland water areas;
          see ./codas3/vecplot/mapfiles/*.map for examples.
  OUTPUT: PostScript file
-------------------------------------------------------------------------------
  CONTROL FILE STRUCTURE:

  { ---PLOT PLACEMENT--- }
  [ new_page ]                  { starts a new page for plotting }
  [ subplot:          < mni > ] { 3-digit number divides page into m by n
  plotting areas, and specifies i as the
  current plotting area; plots are numbered
  sequentially across then down the page;
                 default is 111 }
  [ orientation:      < portrait | landscape > ]       { default is landscape }
  [ top_margin:       < page margin in inches >    { default is 1.0 }
  [ bottom_margin:    < " >                        { default is 1.0 }
  [ left_margin:      < " >                        { default is 1.0 }
  [ right_margin:     < " >                        { default is 0.5 }

  { ---VECTOR PLOTTING OPTIONS--- }
  [ vecfile:          < input filename for velocity data > ] { lon lat u v .. }
  [ vecfiles:         < vecfile #1> [ < vecfile #2 > ... ] end ]
  [ u_v_pair:         < n > ] { index of (u,v) pair to plot from velocity file}
                 { default is 0 = no velocity vectors }
```

```
  [ cruise_track ]            { plot cruise track, default is no cruise track }

  [ arrow:
    [ scale:          < vector scale in cm/s per inch > ] { default: 100 cm/s }
    [ headlength:     < arrowhead length in inches > ]     { default: .1 inch }
    [ min_headlength: < minimum arrowhead length in inches > ] { default is half
of headlength; velocities that scale to less than this
value get drawn as circles instead of vectors }
    [ linewidth:      < pen weight in points > ]         { default: 0.6 point }
    end ]

  { ---MAP PLOTTING OPTIONS--- }
  [ map:
    [ file:           < input filename for map coords. > ]   { lon lat in deg }
    [ linewidth:      < pen weight in points > ]         { default: 0.4 point }
    [ shading:        < gray level for land areas > ]        { 0.0 = black to }
                                                      { 1.0 = white, default }
    end ]

  { ---AXES CONTROL OPTIONS--- }
  [ lon_range:           < start > to < end > by < tick-interval > ] { in deg }
  [ lat_range:           < start > to < end > by < tick-interval > ]
  [ axis:
    [ font:              < PostScript font name > ]     { default: Times-Bold }
    [ font_size:         < n points > ]                         { default: 10 }
    [ linewidth:         < pen weight in points > ]       {default: 0.5 point }
    [ decimal_place:     < m >,< n > ] { no. of decimal places on axes labels }
    [ label_interval:    < m >,< n > ] { label every mth tick mark on lon. axis
                     and every nth tick mark on lat. axis;
 default is 1 }
    [ scale:             < lon deg per inch > by < lat deg per inch > ]
         { default is maximum possible rectangular projection }
    [ origin:            < x >,< y > ] { plot orig. in inches from paper edge }
                               { default is centered within plotting area }
    end ]

  { ---PLOT TITLES AND ANNOTATIONS--- }
  [ main_title:       "< first  main title line >" ]       { the double quotes }
  [ sub_title:        "< second main title line >" ]       { are needed!       }
  [ plot_title:       "< plot or third title line >" ]

  [ title_font:       < PostScript font name > ] { affects only succeeding }
  [ title_font_size:  < no. of points > ]        { *_title: declarations }

  [ label:
```

```
    [ type:           < common | local > ]       { default is common }
    [ text:           "< text to be printed >" ] { quotes are needed }
    [ symbol:         < circle | square | delta | diamond | del | x | plus |
      filled_" |   "    |   "   |   "     |  " > ]
    [ position:       < x >,< y > ]          { in lon,lat degrees }
    [ justify:        < left | right | center > ] { default is left }
    [ font:           < PostScript font name > ] {default is Helvetica }
    [ font_size:      < in points > ]          { default is 12 }
    [ symbol_size:    < in points > ]          { default is 2 }
    end ]
  [ clear_labels ]     { clear list of labels from preceding plot }

  [ legend:
    [ type:           <common or local> ]                  { default: common }
    [ position:       < x >, < y > ]    { in inches from plot origin, if local,
           else from paper origin, if common }
                               default: lower right corner of plot (local) /
page (common) }
    [ label:          "< scale label>" ]            { default: "Speed (cm/s)" }
    [ length:         < length in cm/s > ]     { default: same as arrow scale }
    [ font:           < PostScript font name > ]     { default: Times-Bold }
    [ font_size:      < n points > ]                        { default: 10 }
    [ linewidth:      < pen weight in points > ]       {default: 0.5 point }
    end ]

  [ time_stamp: ]      "< text to use for time stamp >" ] { default is
   system date/time; set to "" to suppress time stamp }
  [ annotation:        "< text to print below time stamp >" ]
      { e.g., "University of Hawaii"; default is no annotation }

  { ---OUTPUT OPTIONS---}
  [ psfile:  < PostScript output filename > ]           { default is stdout }
  [ make_plot | add_plot ] { start plot generation from foregoing options;
     make_plot overwrites the output file, if it exists;
     add_plot appends to it }

  { ---MISCELLANEOUS OPTIONS--- }
  [ skip_to:   < label > ] { skips succeeding control file entries up to label }

-------------------------------------------------------------------------------*/

  main_title:      "ADCP DEMO"
  sub_title:       "April 9 to 10, 1993"
  annotation:      "University of Hawaii"
  label:
```

```
  text:          "Central America"
  font_size:     10
  position:      -90.0, 14.25
  justify:       center
  end

lon_range:        -92.0 to  -89.0  by  0.5       /* tic every half-degree */
lat_range:         11.0 to   14.5  by  0.5
axis:
  label_interval:  2,2                        /* label every other tic */
  end

vecfile:         ademo.vec
psfile:          ademo.ps
map:
  file:         /home/noio/efiring/codas3/vecplot/mapfiles/world.map
  shading:      0.95
  end

orientation:     portrait

subplot:         221
plot_title:      "Layer:  21m to 25m"
u_v_pair:        1
make_plot

subplot:         222
plot_title:      "Layer:  25m to 75m"
u_v_pair:        2
add_plot

subplot:         223
plot_title:      "Layer:  75m to 125m"
u_v_pair:        3
add_plot

subplot:         224
plot_title:      "Layer:  125m to 175m"
u_v_pair:        4
add_plot

new_page

subplot:         221
plot_title:      "Layer:  175m to 225m"
```

```
  u_v_pair:        5
  add_plot

  subplot:         222
  plot_title:      "Layer:  225m to 275m"
  u_v_pair:        6
  add_plot

  subplot:         223
  plot_title:      "Layer:  275m to 325m"
  u_v_pair:        7
  add_plot

  subplot:         224
  plot_title:      "Layer:  325m to 375m"
  u_v_pair:        8
  add_plot

/*****************************************************************************/
```

B CODAS MATLAB Scripts and Functions

Several Matlab scripts and functions have been written to perform some of the ADCP processing steps and to generate plots. A given routine may in turn invoke a number of subroutines, which are all stored in the directory `./codas3/matlab/matlab{3.5,4}`. Only the main routines directly invoked by the user are copied as part of the ADCP processing tree. Hence, the user must set his or her `MATLABPATH` so Matlab can automatically locate auxiliary routines when needed. Many of the script copied to the ADCP processing tree will need to be edited by the user to specify items like input filenames and other control parameters. These are listed below in alphabetical order.

B.1 adcpcal.m

```
function adcpcal(cal, label, trange, dvrange)
% function adcpcal(cal, label, trange, dvrange)
%     cal is the output matrix from TIMSLIP.C
%     label is text for plot titles.
%  This function calculates statistics for the
%  amplitude and phase calibration factors, and
%  the seconds to be added to the recorded time to
%  maximize agreement with the navigation, and
%  plots them as time series and as histograms.
%  Statistics are appended to the file adcpcal.out.
%  The third and fourth arguments are optional:
%     trange is a vector [t0 t1] with the time range of the
%        calibration points to be used.
%     dvrange is a vector [umin umax vmin vmax] with the
%        range of acceptance of velocity component differences,
%        post-acceleration minus pre-acceleration.
%       Eric Firing
%       Fri  02-17-1989; major mod Fri  12-15-1989

% >>>>>>>>>>>>>>>>>>>> user must edit the following lines >>>>>>>>>>>>>>>>>>>>
                     % phase seems more often to contain a trend than amplitude,
                     % so it may be justified to clip phase less stringently
clip_ph  = 3;        % degrees on either side of median
clip_amp = 0.04;     % eliminate anything over x percent from median
clip_var = 0.05;     % maximum variance of ref layer velocity
clip_dt  = 60;       % maximum time slip beyond median
% >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> end editing >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
.
.
.
```

B.2 callrefp.m

```
function callrefp(t0)
%
% Usage:  callrefp(t0)
% where:  t0 = time origin (for x-axis) in decimal days
%
% callrefp generates the input files and sets up the call for plotting
% the absolute reference layer velocities (smoothed and/or unsmoothed).
%
% It calls the ref2mat and sm2mat programs for converting
% the user-specified input files into Mat-files, and then
% calls the refplot.m function to perform the plotting.
% These programs/function must therefore be in the user's search path.
%
% >>>>>>>>>>>>>>>>>>>> user must edit the following lines >>>>>>>>>>>>>>>>>>>>

% input files:             % set either one to [] if not to be plotted
smoothrfile = 'ademo.bin'; % smoothed velocity input file
                           % (binary output of smoothr program)
refabsfile  = 'ademo.ref'; % unsmoothed velocity input file if use_ref = 1
                           % (text file output of refabs program)

% if plotting smoothrfile, set following to match smoothr.cnt values:
max_gap_ratio = 0.05;
min_speed     = 1.0;
max_speed     = 5.0;

% output file (t0 will be appended to this root + extension .ps):
outfile = 'ademo';        % root for output filename (max. 5 chars.)

% axis settings:
dt      = 2.0;            % no. of days to plot (x-axis length in decimal days)
y0      = -1;             % velocity origin (y-axis min. value, in m/s)
dy      = 2;              % velocity range (y-axis range, in m/s)

% plot labels:
cruise  = 'ADCP DEMO';    % title for plot
year    = 1993;           % base year for decimal days (x-axis label)

% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<< end editing <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
.
.
.
```

B.3 clkrate.m

```
% clkrate estimates the PC clockspeed error rate by a least-squares fit
% of the difference between the PC time and satellite time.  It
% automatically weeds out outliers prior to estimation and detects
% breaks in the trend (most probably due to the PC clock being reset
% at mid-cruise), fitting each detected trend separately.  The intercept
% in each case can be interpreted as the PC clock reset time.
%
% Input:  text file with following columns:
%         1) fix time (in decimal days)
%         2) PC time - fix time (in seconds)
%         This can be the .trs output of scanping, if Transit fixes were
%         recorded with the ADCP ping files; or the .scn output of
%         scanping, if GPS fixes were recorded with the ADCP ping files,
%         edited to extract only the decimal day and (pc-fix) time columns
%         (the Cshell script cleanscn can be used to do this).
% Output: .clk file with estimate of PC clockspeed vs. satellite clock
%         and the PC reset time
% Usage:  1) Edit the file clkrate.m to specify the input parameters:
%            a) Examine the (PCtime - Fixtime) column of the input file for outliers.
%            b) Set the abs_max_bad parameter below to exclude those outliers.
%            c) Examine the column for breaks in the general trend.
%            d) Set the abs_max_gap parameter below to catch those breaks, if any.
%         2) In Matlab, type 'clkrate'.
%            The script will first attempt to fit all the included points.
%            If breaks are detected, it will then fit each section separately and
%            display an estimate of the PC clock rate for each section.
%            In that case, either average the different estimates or pick the
%            best one (the one corresponding to the longest time series).
%         3) Use this with the PC reset time(s) for the time_correction
%            entries in the loadping control file.
% Note:   It may be necessary to manually edit the fix file
%         and delete specific lines that defy underlying trends
%         but cannot be caught by suitable thresholds.
%
% 91/05/21 Willa Zhu
%
% >>>>>>>>>>>>>>>>>>>> user must edit the following lines >>>>>>>>>>>>>>>>>>>>

infile = 'ademo.cln';     % input file
dtcolumn = 2          ;   % column number of (pc - fix) time in input file
outfile = 'ademo.clk';    % output file (diary)
yearbase = 93;            % base year for decimal days
abs_max_gap = 7;          % break-in-trend threshold (absolute value, seconds)
abs_max_bad = 1000;       % outlier threshold (absolute value, seconds)

% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< end editing <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
.
.
.
```

B.4 cruistrk.m

```
% cruistrk is used to generate plots of the cruise track (lon vs. lat,
% lon vs. time, and lat vs. time).
%
% Usage:  1) Edit this file to specify the input parameters
%         2) In Matlab, type 'cruistrk'
% Input:  ASCII fix file with columns decimal day, longitude and latitude
%         (from ubprint or some other source).
% Output: PostScript file cruistrk.ps
%
% >>>>>>>>>>>>>>>>>>>> user must edit the following lines >>>>>>>>>>>>>>>>>>>>

fix_file    = 'ademo.ags'
cruise_id   = 'ADCP DEMO'
year_base   = 1993;
plot_symbol = '.';          % choose from '.', '+', '--', etc.

% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<< end editing <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
.
.
.
```

B.5 runbtcal.m

```
% runbtcal calls btcal.m to calculate the bottom track calibration.
% It is meant to be called repeatedly, each time calling worst.m to delete the
% currently worst fix before recalculating, in order to see the effect of
% eliminating one fix after another.
%
% Be sure to edit the 3 lines indicated below before starting.  Also,
% be sure to type:
%
%     >> first = 1
%
% upon starting up Matlab, before calling runbtcal for the first time.

 % >>>>>>>>>>>>>>>>>>>>>>>> edit these lines >>>>>>>>>>>>>>>>>>>>>>>>

 fname = 'ademo.ref';  % input filename (refabsbt's output)
 froot = 'ademo';      % fname without the extension
 label = 'ADCP DEMO';  % title for plot

 % <<<<<<<<<<<<<<<<<<<<<<<< end editing <<<<<<<<<<<<<<<<<<<<<<<<
.
.
.
```

B.6 runstick.m

```
% Usage:  1) edit the file runstick.m to specify the input parameters
%         2) in Matlab, type 'runstick'
%
% Input:  *_uv.mat and *_xy.mat output from adcpsect
% Output: PostScript file (.ps)
%
% runstick generates one or more of the following types of plots:
%
%   1) Velocity;
%   2) Mean, trend, tides, inertial;
%   3) Residual;
%   4) Inertial clockwise (cw) and counterclockwise (ccw);
%   5) Diurnal cw and ccw;
%   6) Semidiurnal cw and ccw;
%   7) Harmonic analysis of velocity
%      a) mean and trend, and
%      b) ellipse, phase and direction of diurnal, semidiurnal and inertial.
%
% where the y-axis is always depth (in meters) and the x-axis is:
%
%   1) time, or longitude, or latitude for plot types 1 thru 6
%   2) labeled 'mean+trend', 'semidirunal', 'diurnal', 'inertial' for type 7
%      (Note:  The user must edit the Matlab version 3.5 PostScript output file
%              before printing in order to delete the numerical labels on the
%              x-axis; Matlab v. 4 has a facility for suppressing these.)
%
% The period for semidiurnal = 12.42 hours
%                diurnal     = 24 hours
%                inertial    = 12/sin(lat) hours
%                              e.g., if lat = 22.75 deg, inertial = 31 hours
%
% For each type of plot to be generated, the user must specify one or
% more of the following parameters:
%
%   1) xyrange     = [ xmin xmax ymin ymax ]
%                     x- and y-axis range of the plot.
%   2) vscale      = velocity units per inch
%                    e.g. (m/s) per paper inch
%   3) key         = [ x y v ]
%                    where x and y are location of key in axis units
%                    and v is length of key in velocity units.
%   4) arrowlength = length of arrowhead in velocity units
%                    Note: set arrowlength = 0 to omit arrowheads;
```

```
%                          this greatly reduces memory requirements.
%

% 91/05/30 Willa Zhu

% >>>>>>>>>>>>>>>>>>>> user must edit the following lines >>>>>>>>>>>>>>>>>>>>

                    %-------------------------------------------------------
infile = 'os'      ;% root of input Mat-files (_uv.mat & _xy.mat
                    % output of adcpsect)
cruise = 'ADCP DEMO'% cruise id (appears in upper righthand corner of plot)
year   = 1993;      % base year for decimal days
outfile= 'ademo_os';% root of plot output file; set to
                    % [] to disable PostScript output
append = 0;         % 0 = overwrite outfile if it exists; 1 = append
pause_flag = 1;     % 1 = pause between screenfuls of plots; 0 = don't

                    %-------------------------------------------------------
x_index = 3;        % 1 = longitude on x-axis
                    % 2 = latitude on x-axis
                    % 3 = time on x-axis

if x_index == 3     %-------------------------------------------------------
  harmonic  = 4;    % 0 = don't plot harmonic analysis of velocity
                    % 1 = plot mean and trend at end time
                    % 2 = 1 + ellipse, phase & direction for semidiurnal
                    % 3 = 2 + ellipse, phase & direction for diurnal
                    % 4 = 3 + ellipse, phase & direction for inertial
  plot_title = '';  % optional string to be concatentated to standard plot titles

elseif x_index == 1 | x_index == 2      % specify plot title
  plot_title = '158W Southbound';
end

                    %-------------------------------------------------------
                    % select types of plots to generate:  1 = plot
                    %                                     0 = don't plot
velocity    = 1;    % velocity stick plot
mean_trend  = 1;    % mean, trend, tides, inertial stick plot
residual    = 1;    % residual stick plot
semidiurnal = 1;    % semidiurnal stick plot
diurnal     = 1;    % diurnal stick plot
inertial    = 1;    % inertial stick plot

xyrange = [ 98.0   99.5  -300 0];  % [ x_min x_max y_min y_max ]
```

```
vscale  = 1;                        % velocity units per paper inch
arrowlength = 0.025;                % arrowhead length in velocity units
key = [ 99.2 -10 0.1];              % key position/scale = [x y v]

if inertial ~= 0                    %----------------------------------------
  inertial_period = 2.22;           % 0.5/sin(pi/180*lat = 13 for demo) days
end

% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< end editing <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

runstic2                            % do it
```

B.7 setup.m

```
% SETUP.M
% USAGE: SETUP
%
%  SETUP initializes the global variables that will be used in the following
% .M file or function calls: GET.M, INIT.M, OFFSET.M, DRAW.M, BAD.M, LIST.M,
% etc. (see DBEDIT.DOC).  By convention, all global variables are in caps.
%
global DBNAME DEFAULT_NPRFS DEFAULT_NBINS DEFAULT_SEQ_FLAG INITed
global REFLAYER_STARTBIN REFLAYER_ENDBIN

LISTED = 1;
INITed = 0;

% Following are default settings for the demo database.          .
% Edit the following lines to set up your own defaults.
%
%                          *********************************
%                          ** getmat.exe default settings **
%                          *********************************

% >>>>>>>>>>>>>>>>>>>>>>>>>>>>>> Retrieval Parameters >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
DBNAME = '../adcpdb/ademo';      % retrieve data from CODAS database DBNAME
DEFAULT_SEQ_FLAG = 1; % 0 means retrieve nprfs profiles before & nprfs after getp arg
              % 1 means retrieve nprfs profiles after getp arg
              % -1 means retrieve nprfs profiles before getp arg
DEFAULT_NPRFS = 25;   % retrieve 50 profiles <before & after> / <after> getp arg
              % set to lower number if you run into memory problems
DEFAULT_NBINS = 128;  % retrieve maximum of 128 bins per profile
REFLAYER_STARTBIN = 5;% subtract this reference layer mean from U, V and W
REFLAYER_ENDBIN = 20; % set REFLAYER_STARTBIN to 0 to disable;

% >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> Editing Parameters >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
.
.
.
```

B.8 stn_udw.m

```
% stn_udw is used to plot on-station and/or underway section means and
% standard deviations of U, V, W, Error Velocity, Amplitude, Spectral Width,
% Percent Good and Percent 3-Beam Solutions.  It is useful for assessing
% ADCP performance on-station and underway.
%
% Usage:  1) Edit this file to specify the input parameters.
%         2) In Matlab, type 'stn_udw'
% Input:  .mat file with on-station and/or underway data (from profstat program)
% Output: PostScript file (.ps)
%

% >>>>>>>>>>>>>>>>>>>> user must edit the following lines >>>>>>>>>>>>>>>>>>>>

% >>> input files >>>        % set to [] if not to be plotted
stn_file  = 'ademo_os.mat';  % input Matfile name for on-station data
udw_file  = 'ademo_uw.mat';  % input Matfile name for underway data

% >>> output >>>
outfile   = 'ademo';         % root for output PostScript file
sub_title = '(Solid: On Station; Dash: Underway)';  % plot subtitle

% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< end editing <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

stn_udw2                     % do it
```

C Other CODAS Files

Included here are miscellaneous files referred to in the ADCP Processing Manual. They include a sample producer definition file and a file that explains the data processing mask settings.

C.1 adcp720.def

```
/*************************************************************************

  FILE:  adcp720.def

         This producer definition file is used with the loadping program
         to load ADCP ping data into a CODAS database.

         This particular version is for the Moana Wave's vessel-mounted
         ADCP using the version of the user-exit program that generates
         a 2240-type user buffer.  See the files adcp1281.def and
         adcp1320.def for the older 1320-, 1280-, and 1020-type user buffers.

-------------------------------------------------------------------------*/

DATASET_ID         ADCP-VM    /* vessel-mounted ADCP */
PRODUCER_ID        32R2MW0001 /* 32 = USA, R2 = UH, MW = Moana Wave */
BLOCK_DIR_TYPE     0
PROFILE_DIR_TYPE   3          /* time, position, and depth range keys */
/*
        DATA DEFINITION FOR ADCP DATA

frequency     id value_type  data_name         offset  scale  units
*/
BLOCK_VAR      0  SHORT     DEPTH               0       1       m
UNUSED         1  USHORT    TEMPERATURE         -10     1.E-3   C
UNUSED         2  USHORT    SALINITY            0       1.E-3   ppt
UNUSED         3  USHORT    OXYGEN              0       1.E-3   ppt
UNUSED         6  STRUCT    OPTICS              0       1       none
PROFILE_VAR    7  UBYTE     AMP_SOUND_SCAT      0       1       none
PROFILE_VAR    8  SHORT     U                   0       1.E-3   m/s
PROFILE_VAR    9  SHORT     V                   0       1.E-3   m/s
UNUSED        10  SHORT     P                   0       1       dbar
.
.
.
/*
     STRUCTURE DEFINITIONS
*/
DEFINE_STRUCT  CONFIGURATION_1  23
  ELEM         1  FLOAT     avg_interval                       s
  ELEM         1  SHORT     compensation                       none
  ELEM         1  SHORT     num_bins                           none
  ELEM         1  FLOAT     tr_depth                           m
.
.
.

DEFINE_STRUCT  USER_BUFFER  5
  ELEM         1  SHORT    version                             none
  ELEM         1  SHORT    n_samples                           none
  ELEM         1  SHORT    s_added                             none
  ELEM         1  SHORT    spare                               none
  ELEM         2  STRUCT   fix                                 none

DEFINE_STRUCT  fix      9
  ELEM         1  LONG     pc_seconds                          s
  ELEM         1  LONG     gps_seconds                         s
  ELEM         1  DOUBLE   latitude                            deg
  ELEM         1  DOUBLE   longitude                           deg
  ELEM         1  FLOAT    height                              m
  ELEM         1  BYTE     dop                                 none
  ELEM         1  BYTE     nsat                                none
  ELEM         1  BYTE     msg_type                            none
  ELEM         1  BYTE     spare                               none

.
.
.
```

C.2 dpmask.h

```
/*************************************************************

   DPMASK.H

   Assigns meaning to the bit positions in the data
   processing mask, for ADCP data.

   Provides a name list for these bit positions.

   Eric Firing
   Wed  09-07-1988

*************************************************************/
.
.
.
#define TIME_CORRECTED           0  /* loadping, chtime */
#define TRANSDUCER_ORIENTATION   1  /* rotate */
#define COMPASS_ERROR            2
#define TRANSDUCER_BEAM_ANGLES   3
#define SOUND_SPEED_VEL_COR      4  /* fix_temp */
#define SOUND_SPEED_DEPTH_COR    5
#define GLITCHES_REMOVED         6  /* badbin */
#define GLITCHES_CORRECTED       7
#define BOTTOM_SOUGHT            8  /* botmpas3 */
#define BOTTOM_MGB               9  /* last_85 */
#define MGB_SET                 10
#define DEPTH_RANGE_COR         11
#define RELATIVE_PROFILES       16
#define REFERENCE_VELOCITIES    17
#define VEL_TO_ABSOLUTE         18
#define VEL_TO_REF_LAYER        19
#define POSITIONS_OPTIMIZED     20  /* putnav */
#define NAV_SOURCE_GPS          21  /* putnav */
#define NAV_SOURCE_TRANSIT      22  /* putnav */
#define NAV_SOURCE_LORAN        23  /* putnav */
#define NAV_SOURCE_OMEGA        24  /* putnav */
#define NAV_SOURCE_RADAR        25  /* putnav */

.
.
.
```

D CODAS ASCII Dump Utility

A utility is available for obtaining an ASCII dump of CODAS block files. To perform this task, one first modifies the control file 'asc_dump.cnt' by adding a list of the block files for dumping. For example,

```
BLOCK_FILES:
  /home/kapaub/caldwell/adcp/moanawve.usa/00004/a9218001.blk
  /home/kapaub/caldwell/adcp/moanawve.usa/00004/a9218002.blk
  /home/kapaub/caldwell/adcp/moanawve.usa/00004/a9218003.blk
end
```

Note, if the block files are in the same directory from which execution occurs, then the paths are not necessary. One line is needed for each block file desired for conversion.

Secondly, one runs the program by entering

```
asc_dump asc_dump.cnt
```

The output will consist of one ASCII file for each CODAS block file with a similar file name root yet with the file extension, '.asc'. For instance, if the input is file 00006001.blk, then the ASCII output file is 00006001.asc. There is roughly a factor of 5 in expansion, so a typical block file size of 600K equates to an ASCII file of roughly 3 Mbytes (and a typical cruise-month of block files of roughly 10 Mbytes equates to about 50 Mbytes of ASCII files!).

The ASCII dump contains all the information stored in the block file. Key words are used to denote sections, each of which begins with a START_section and ends with an END_section. The basic layout with comments for each section are shown below.

```
START_DOCUMENTATION:
 "cruise summary block footer in entirety"
END_DOCUMENTATION:
START_DATA_LIST:
  "list of stored variables with indication of either
   block variable (stored once per block) or profile
```

```
   variable (repeated for each profile) and for either
   binned_array or structure as the type as noted below"
END_DATA_LIST:
START_BLOCK_VARIABLES:
       "signifies start of variables stored only once per block"
    START_ARRAY:               name_of_variable_from_data_list
      "block-type values, e.g. depth"
    END_ARRAY:
    START_STRUCT:              name_of_variable_from_data_list
      "list of the structure elements, e.g. configuration"
    END_STRUCT:
      "each block variable is shown as either an array or structure"
END_BLOCK_VARIABLES:
START_PROFILE_VARIABLES:     NUMBER PROFILES= 10
      "the number of profiles is generated by the ascii_dump program"
START_PROFILE_1:
Time= 1993/01/22 23:11:39
Longitude=  144 35' 18.50" Latitude=   13 22' 46.02"
      "the beginning of each profile always shows time and position
       then each profile variable in the data list is shown as either
       a binned_array or structure"
    START_ARRAY:               name_of_variable_from_data_list
      "values"
    END_ARRAY:
    START_STRUCT:              name_of_variable_from_data_list
      "list of the structure elements"
    END_STRUCT:
END_PROFILE_1:
.
.
START_PROFILE_X:
   .
   .
END_PROFILE_X:
END_PROFILE_VARIABLES:
```

E Vector Plotting Program

The vector plotting program generates a longitude vs. latitude plot with one or more of the following elements:

1) coastlines
2) vectors
3) cruise track

It requires two or three input files, depending on which of the above elements will be plotted:

1) the control file, which specifies the program plotting options
2) a map file, if coastlines are desired
3) the vector data file, if vectors and/or the cruise track are to be plotted

To run the program, the user usually edits a copy of the sample control file and then simply types:

```
vector [ control filename ]
```

assuming the vector executable is in the user's current directory or search path. The program then generates PostScript commands for displaying the plot, which can be spooled to a PostScript printer using the appropriate system command (e.g., 'lpr outfile.ps' on Unix systems), or displayed on a screen using a PostScript viewing package like ghostscript.

The Vector Control File

This text file contains all the user's directions to the vector plotting program. As a starting point, it is easiest to just copy the sample vector.cnt file in the ./codas3/cntfiles directory and then edit it as needed. The contents of this file can initially be classified into two types:

1) comments, always bracketed by /* */
2) options

Comments are ignored by the program, and should NOT be nested.
Options can follow three patterns:

1) a keyword alone
2) a keyword followed by a user-supplied string, quotation or value(s).

3) a keyword followed by a list of suboptions terminated with the word 'end'.

Keywords have special meaning to the program and must be spelled as shown (case-sensitive!). The colon, when present, is part of the keyword; there should be no space between the keyword and the colon, but there should be whitespace (blank or newline) following the colon. The colon is needed in case 2) and 3) above, that is, when a keyword must be followed by more information from the user or more sub-options terminated by 'end'.

When a particular keyword is omitted from the control file, the program will assume some default value or action, which may or may not be what the user wants.

Below is an explanation of all the keywords and their default settings. Angular brackets <> are used to signify where the user must supply a value for the keyword. The brackets must not be typed. Where additional punctuation or words appear outside the brackets, the user MUST include those as they are significant delimiters. For example, text for the title options must be enclosed in double quotation marks. Position coordinates (x,y) must have the two values separated by the comma.

A single control file can be used to produce multiple plots per page and multiple pages of plots. Most options are "sticky," that is, once set, they carry over to subsequent options, plots, and pages until they are reset to some other value. This is consistent with the expectation that a single control file is generally used to generate similar vector plots, for example, to show velocities at different depth layers for a given region. After outlining the options for the first plot, only those particular options that need to be different for the subsequent plot need to be respecified.

Plot Placement Options
---

top_margin: < n inches >
left_margin: < n inches >
right_margin: < n inches >
bottom_margin: < n inches >

The *_margin: options are used to set the page margins. The space within these margins is considered the plotting area. The plotting area includes room for the plot itself, the page and plot titles, and the numeric labels on the axis tick marks. The time stamp and annotation, if any, will always appear just inside the upper right corner of the page, regardless of the margin setting. (These can be suppressed but not relocated elsewhere; see time_stamp: and annotation: options below.) The scaling legend, by default, will

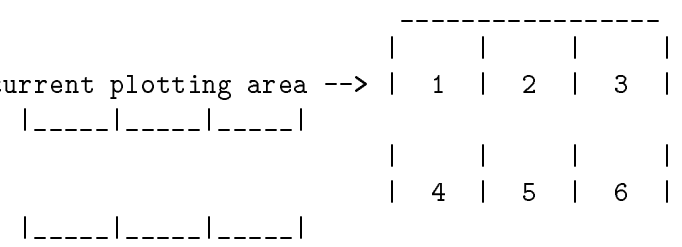
appear below the lowest plot axis, and may frequently end up in the
bottom margin. See the legend: option below to see how to relocate
the legend.

By default, the top, bottom, and left margins are all set to 1 inch,
and the right margin is set to 0.5 inch.

subplot: < mni >

The 3-digit number, mni, instructs the program to divide the page into m by n
equal-sized plotting areas, and to put the current plot in position i,
where the plotting areas are numbered sequentially across from top to bottom.
If this option is not used, then the program assumes that the entire page
is one plotting area.

The diagram below shows how vector interprets "subplot: 231", for example.



new_page
orientation: < landscape | portrait >

The new_page option is used to place subsequent plots on a new page.
It is only needed when more than one plot is being placed per page
(that is, the subplot: mni argument has m > 1 or n > 1).
When the subplot: option is not being used, or is set to 111,
the program automatically places each plot on its own page.

The orientation: specifies whether the x-axis of the plot will be parallel to
the longer edge of the page (landscape) or to the shorter edge of the page
(portrait). The default is landscape orientation.

Axis Control Options
----

lon_range: < left longitude > to < right longitude > by < tick interval >
lat_range: < min. latitude > to < max. latitude > by < tick interval >

These options specify the x- and y-axis range, respectively. The lon_range:

```
can be specified as being between -180 to 180 or between 0 to 360 degrees.
The lat_range: is some subset of -90 to 90 degrees.  The tick interval,
is specified in degrees and indicates where the axis ticks will occur;
a tick interval of 1, for example, means that a tick will show at every
integral degree.

If left unspecified, the lon_range: and lat_range: default to 0.0,
and the vector program will exit with the message:  "Nothing to plot!"

axis:
  font:            < PostScript font >
  font_size:       < n points >
  linewidth:       < n points >
  label_interval:  < n ticks >
  decimal_place:   < n decimal places >
  scale:           < lon degrees per inch > by < lat degrees per inch >
  origin:          < x inches > , < y inches >
  end

The axis: option is used to further specify how the axes are to be drawn
and labeled.  The font: and font_size: option may be used to change the
font used in printing the numerical labels on the axis tick marks from
the default setting of 10-point Times-Bold.

The linewidth: option can be used to attain heavier or lighter axis lines
and ticks.  The default setting is 0.5 point.

The label_interval: option is used to control the frequency of numeric
labels on the axes.  By default, every tick mark will be labeled (i.e.,
label_interval: is 1).  The user may set this option to 5, for example,
so only every 5th tick mark will be labeled.

The decimal_place: option is used to control the precision used in
displaying the numeric labels on the tick marks.  By default, the
program displays as many decimal places as required to reflect the
precision of the label_interval: value multiplied by the lon_range:
and lat_range: tick interval.  For example, if the lon_range:
tick interval specifies that ticks be spaced every quarter-degree while the
label_interval: specifies that every other tick be labeled,
then the program will be labeling every half degree and will
display one decimal place by default.

The scale: option is used to specify the x- and y-axis scale in degrees
per inch.  By default, the program selects scaling factors that achieve
the maximum possible rectangular projection that can be accommodated
```

within the plotting area. This projection causes the longitude axis
to be scaled at cosine of the midpoint of the latitude range.

When specifying the scale: option, the user must take care to pick
values that can accommodate the longitude and latitude range to be plotted
in what the program perceives to be the plotting area; otherwise, the
program exits with the error message: "Unable to fit plot on page."
The plotting area is determined by the page margin settings, the subplot:
option above, the presence of and font sizes used for plot and page titles,
the font sizes used for axis labels, and so on. If the error message
indicates that the plot barely fits, then fiddling with the title and
font settings may solve the problem. Otherwise, the user must either
narrow down the plotting range(s) or increase the scale: specification.

The origin: option is used to position the current plot's origin at a
specific distance from the page's origin, which coincides with the lower left
corner of the paper oriented according to the orientation: specification above.
This distance is specified in inches.

If the plot's origin is not specified using this option, the program
calculates it such that the plot will be centered within the plotting area.

The axis: list of sub-options must always be terminated by the word 'end'.

Vector Plotting Options

```
vecfile:      < [ path ] input file name >
vecfiles:     < [ path ] input file name >
      < [ path ] input file name >
  .
  .
  .
              end
```

The vecfile: option is used to specify the name of one input file that
contains data to be plotted. The vecfiles: option is used to specify
a list of one or more such input files. Each input file must be
an ASCII-formatted file with data in columns delimited by whitespace.
The first two columns must give longitude and latitude in decimal degrees.
When vectors are to be plotted, then one or more additional pairs of columns
must also be present to specify the U and V velocity components in m/s.

When the vecfiles: option is used, the list of input files must be
terminated by the word 'end'.

```
u_v_pair:      < n pair >
cruise_track
```

When vectors are to be plotted, the u_v_pair: option must be used to specify which pair of U/V columns from the input vector file should be plotted.

The cruise_track option is used to request that the longitude/latitude data from the input vector file be joined by lines to show the cruise track.

```
arrow:
  headlength:      < n inches >
  min_headlength:  < n inches >
  linewidth:       < n points >
  scale:           < n cm/s per inch >
  end
```

The arrow: option can be used to control how the vectors are drawn. The headlength: option is used to specify the maximum length of the arrowheads on the vectors, in inches. By default, the headlength: is set to 0.10 inch.

The min_headlength: is used to specify the minimum length of the arrowheads on the vectors for the cases where velocities scale below the headlength: specification. When velocities scale below this min_headlength: specification, a dot is drawn instead of an arrow. By default, the min_headlength: is set to half the headlength: specification.

The linewidth: option is used to set the pen weight for drawing the vectors. The default value is 0.6 point.

The scale: specifies the vector scaling in cm/s per inch. The default setting is 100 cm/s to an inch.

Map Plotting Options

```
map:
  file:       < [ path ] filename >
  shading:    < gray level, from 0.0 (black) to 1.0 (white) >
  linewidth:  < points >
  end
```

The map: option is used to request that coastlines be drawn. The file: option is used to specify the name of the file containing longitude and latitude coordinates for the coastlines. Several such files are distributed along with the program, in directory ./codas3/vecplot/mapfiles along with a utility program called 'extract', which is useful for reducing a map file to some smaller subset of interest that can be read in more quickly when multiple plots with the same coastlines are to be generated. (See the Map Files and Extract Program sections below for details.) The user can also supply such a file. It must be a 2-column ASCII file of longitude and latitude, where each set of continuous coastlines is terminated by '-9999.00 -9999.00' and coordinates for inland water boundaries are terminated by '-8888.00 -8888.00'.

The shading: option specifies the gray level to be used in shading in the land areas. A value of 0 blackens these areas, and a value of 1 results in no shading. Any value in between is admissible. The default value is 1 (no shading).

The linewidth: option is used to control the pen weight used in drawing the coastlines. The default is 0.4 point.

Note: As of 95/05, there are still a few bugs in the map-drawing routines that may cause wierd coastlines or even a core dump when particular longitude ranges are requested. Often, a minor adjustment in the longitude range (extending or reducing it a few degrees or so) may be all that is necessary to work around the problem.

Plot Titles and Annotations

```
main_title:  "< text for page title >"
sub_title:   "< text for subtitle >"
plot_title:  "< text for plot title >"
```

The main_title: and sub_title: options are used to specify page titles, which are centered over the plot(s). The text for these titles must be enclosed in double quotation marks. By default, they are printed in Times-Bold at 20 points and 10 points, respectively. See the title_font: and title_font_size: options below for how to change these settings.

The plot_title: is used to specify a title for a plot. This will be centered over that particular plot only. By default, it is printed in Times-Bold at 10 points. See the title_font: and title_font_size: options below for how to change these settings.

```
Maximum length for all titles is 80 characters.
To disable a title from a previous plot/page, simply set the appropriate
title option to an empty string, for example, plot_title:  "".

title_font:        < PostScript font >
title_font_size:   < no. of points >

These two options are used to change the font used for titles.  The
title_font: is used to specify a PostScript font, while the title_font_size:
is used to specify the font size in points.  These options must precede
the *_title: option for which they are intended to take effect, for example:

main_title:        "KN9302"
title_font_size:   15
sub_title:         "6 to 10 April 1993"
title_font:        Helvetica
plot_title:        "Layer: 21 to 25m"

In the above (rather strange) example, the main title will be printed
in the default font, Times-Bold at 20 points.  The subtitle will be
printed in Times-Bold, at 15 points.  And the plot title will be printed
in 15-point Helvetica.

label:
  type:         < common | local >
  position:     < x > , < y >
  justify:      < left | right | center >
  text:        "< text for label >"
  symbol:       < circle | square | delta | diamond | del | x | plus |
          filled_circle | filled_square | filled_delta |
          filled_diamond | filled_del >
  font:         < PostScript font >
  font_size:    < points >
  symbol_size:  < points >
  end

The label: option can be used to overlay text and symbols on the plot.
There are two types of labels:  common and local.
A common label is overlaid on every subsequent plot generated.
A local label appears only on the current plot.  By default, a label is common.

The position: option specifies the coordinates, in degrees,
at which the label is to be drawn.  By default, the label will be
left-justified at the location specified by the position: option.
The justify: option can be used to control the alignment.
```

```
The text: option specifies the text to be overlaid.  The text must
be enclosed in double quotation marks.  Maximum length is 80 characters.
The symbol: option is used to select which symbol to print.
If both text: and symbol: are specified, then the text appears
to the right of, left of, or centered under the symbol,
depending on whether the justify: option is set to left,
right or center, respectively.

By default, the text is printed using typeface Helvetica, 12 points.
The font: and font_size: options can be used to select some other font/size.

By default, 4-point symbols are printed.  The symbol_size: option is
used to specify some other size.

Important:  when several labels are being specified, all the sub_options
from the preceding label specification are carried over (sticky),
except for the position:, text:, and symbol: options.  Hence,
there is no need to respecify the other sub-options unless they
specifically need to be different from the immediately preceding label.

clear_labels

This option is used to turn off all common labels on subsequent plots.

legend:
  type:         < common | local >
  length:       < cm/s >
  label:       "< text for legend >"
  font:         < PostScript font >
  font_size:    < no. of points >
  linewidth:    < no. of points >
  end

When vectors are plotted (that is, when the u_v_pair option is used),
then the program normally prints a scaling legend right below the
lowest plot axis.  The legend: option can be used to control how
and where this legend is printed.

We distinguish between common and local legends:
A common legend is printed once per page.  A local legend is
printed once per plot.  The default type is common.

By default, the common legend is printed half an inch below the
lowest plot axis and nearly right-aligned with the right axis of
```

the rightmost plot. By default, a local legend is printed
just inside the plot axes, at a distance of 1/2", 3/4" from
the lower right corner of the plot.

The position: option can be used to change these default positions.
The < x > and < y > values should be specified in inches from the page origin,
in the case of a common legend, and in inches from the plot origin,
in the case of a local legend. The upper left corner of the scaling legend
will then be positioned at those coordinates.

By default, the program will use the label "Speed (cm/s)" on the legend.
The label: option can be used to specify some other label. It
can also be used to suppress printing of the legend altogether,
by setting it to an empty string ("").

By default, the scaling legend will display the same length as the
arrow: scale: specification. For example, if the arrow: scale:
specification is left at the default of 100 cm/s per inch, then
the scaling legend will show 100 cm/s and will be one inch long.
The length: option can be used to display some other value and
hence control the size of the printed legend.

By default, the label and numbers on the legend will be printed in
10-point Helvetica. The font: and font_size: options can be used
to select some other PostScript font/size.

By default, the scaling legend will be plotted using a 0.5-point
line. The linewidth: option can be used to specify a different line width.

The list of legend: sub-options must be terminated by the word 'end'.

time_stamp: "< text to use in place of system time stamp >"
annotation: "< text to print below time stamp >"

By default, the vector plotting program always prints the time from the
system clock in the upper right corner of the page. The time_stamp:
option would normally be used to suppress this default behavior by specifying
an empty string, as in time_stamp: "". It can also be used to specify
some other string in place of the system time.

The annotation: option can be used to specify text to be printed right below
where the time stamp appears (in the upper right corner of the page).
By default, no annotation is printed.

Output Options

---

ps_file: < PostScript output file name >

This option specifies the name to use for the program's output file,
consisting of PostScript language instructions to the printer.
If it is omitted, then the program prints the output to stdout
(the screen, unless redirected elsewhere). If the file already exists,
then the program either overwrites or appends to it, depending on
which command is used to start plotting:

make_plot
add_plot

These two commands instruct the program to start generating the plot
based on the options already read in. If make_plot is used, then
the output file is created or overwritten, if it already exists.
If add_plot is used, then the output file is appended to, if it
already exists, or created otherwise. Normally, the first plot
instruction in a given control file will be 'make_plot' while
subsequent plots will use 'add_plot' if the same ps_file:
specification is in effect.

Miscellaneous Options
---

skip_to: < label >

This option directs the vector program to ignore all subsequent control file
entries until it encounters a match for the indicated < label >.
The < label > can be any sequence of characters, for example, 'YY'.
This is a useful alternative to using /* */ to temporarily "comment out"
parts of the control file, especially when the section to be skipped over
already contains some comment fields (since nested comments are not allowed).

Map Files

Presently, the vector program comes with five map files of varying
resolution and coverage.

    1) codas3/vecplot/mapfiles/westpac.map (West Pacific)

        resolution: medium
        lon_range : 120E to 160E

```
        lat_range :   10S to  15N

    2)  codas3/vecplot/mapfiles/solomon.map (Solomon Islands)

        resolution:  high
        lon_range :  140E to 161E
        lat_range :   12S to   9N

    3)  codas3/vecplot/mapfiles/hawaii.map (Hawaiian Islands)

        resolution:  high
        lon_range :  161W to 154W
        lat_range :   18N to  23N

    4)  codas3/vecplot/mapfiles/asia.map (Asia and Pacific Ocean)

        resolution:  medium
        lon_range :   10E to 236E (124W)
        lat_range :   55S to  82N

    5)  codas3/vecplot/mapfiles/world.map

        resolution:  low
        lon_range :    0E to 360E
        lat_range :   90S to  90N
```

If you are using only a small subset of the map file to generate several plots, it is a good idea to create your own subset map file once and use this for all your plots, thereby saving time needed to read the original map file (this is especially important when using the rather large asia.map and world.map files). Please note that the above map files were digitized with a different purpose in mind, and some of them (in particular, the asia.map file) will have problems when the vector program shading: option is turned on (the coastline coordinates may not define a closed polygon). It may be necessary to forego the shading in such cases.

The Extract Program

To run the map extracting program, type:

```
    extract <input map file> <output filename> <minimum longitude>
       <maximum longitude> <minimum latitude> <maximum latitude>
```

where the input map file contains the longitude-latitude coordinates, and the

```
output filename will contain the subset specified by the minimum and maximum,
longitude and latitude parameters.
```