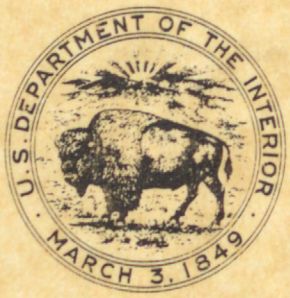
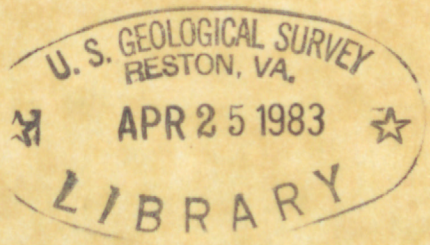
(200)
R290
no.83-276



Open-File Report
83-276

AGRAM: a Series of Computer Programs for Processing Digitized Strong-Motion Accelerograms

Version 1.0



United States Department of the Interior
Geological Survey

United States Department of the Interior
Geological Survey

USGS LIBRARY-RESTON
3 1818 00068968 5

AGRAM: a Series of Computer Programs for Processing Digitized Strong-Motion Accelerograms

Version 1.0

Compiled by
April Converse
07 February 1983

Open-File Report
83-276

✓ Two orad

This report is preliminary and has not been reviewed for conformity with U. S. Geological Survey editorial standards.

Any use of trade names is for descriptive purposes only and does not imply endorsement by the USGS.

This text is stored in the file named DR2:[110,110]AGRAM.DOC in the PDP 11/70 at the National Strong Motion Data Center. The task files for all the programs are also in DR2:[110,110].

342781

Open-file report
(Geological Survey
(U.S.)

Open-File Reports are Distributed by:

Open-File Services Section
Branch of Distribution
U. S. Geological Survey
Box 25425, Federal Center
Denver, Colorado 80225

(303) 234-5888

TABLE OF CONTENTS

PREFACE

This report presents a preliminary description of the computer programs that are used for processing digitized strong-motion accelerograms by the U.S. Geological Survey. Recent improvements to the programs provide an improved instrument correction, maximum use of densely digitized data, and freedom to choose interactively from a variety of long-period filtering schemes. The programs are still in the development process; many of the options they provide have yet to be evaluated and many other features intended for the programs have yet to be implemented. With continued support and further development, however, the programs will evolve into a comprehensive series of relatively transportable FORTRAN 77 computer programs available to any investigators who require strong-motion data processing software.

The programs and report are changing continuously as their development progresses. The current version of the report is intended for use as an interim user's guide for members of the USGS. As yet, it is incomplete and often merely provides names of those people within the USGS who can provide guidance for aspects of the programs that this author has not considered yet. The report is published as an open file report to inform organizations outside the USGS of the goals for the completed project and of the project's current status, but the report will become outdated as the capabilities of the programs continue to expand.

07 February 1983

April Converse
U. S. Geological Survey
Mail Stop 78
345 Middlefield Road
Menlo Park, CA, 94025

telephone: (415) 323-8111
extension 2881

or FTS 467-2881

ACKNOWLEDGMENTS

This report and the computer programs it describes are compilations of techniques and recommendations devised through cooperative efforts of many members of the USGS. A. Gerald Brady, William Joyner, Peter Mork, Virgilio Perez and Michael Raugh have all made substantial contributions.

Many of the techniques used in these programs, even the names of some of the programs, were adapted from a series of programs that were originally developed at the Earthquake Engineering Research Laboratory of the California Institute of Technology during the years 1968 to 1972.

This work was funded in part by National Science Foundation Grant CA-114, under interagency agreement with the USGS through 1982. Support for subsequent development has been assumed by the USGS.

CHAPTER 1

Introduction

1.1 Overview

The computer programs described in this report are those that are used to process analog strong-motion accelerograms that have been digitized by IOM/TOWILL Corporation in Santa Clara, California. The programs will also accept data that have been recorded in digital form by a Sprengnether DR100 recorder and they will eventually accept data that have been acquired by other types of digital recorders or by manual digitization.

The contents of a tape written at the digitizing facility are processed at the National Strong Motion Data Center (NSMDC) at the USGS offices in Menlo Park, California. First, the BUTTER program is used to rejoin the separately digitized frames of an accelerogram into one continuous record. The SCALE program then scales the data to represent time and acceleration rather than digitizer units. Next, HIFRIC interpolates the data, applies an instrument correction and filters high frequencies from the data. CORAVD integrates acceleration to obtain velocity and displacement and optionally performs a linear base line correction or filters out long-period content. The PHASE3, 4, 5 and 6 programs perform spectral analyses of the CORAVD results.

1.2 Data Sources

1.2.1 Analog Records

Most of the strong-motion recording instruments that provide the raw data to be processed by the AGRAM programs discussed in this report are analog accelerographs that record on 70mm film. Older accelerographs record on 12-inch paper and a few multi-channel accelerographs record on 7-inch film. Strong-motion accelerographs do not record continuously but begin recording only after earthquake motion has reached a significant level. Recorders are usually set to trigger with approximately .01g in the vertical direction and are designed to be up to speed in, at most, 0.1 second.

Most records contain three accelerometer traces, one or more reference (or "fixed") traces, and one or more time traces. The three accelerometer traces correspond to orthogonal components of motion. The first (topmost) accelerometer trace of the most prevalant records represents horizontal motion, the second represents vertical motion, and the third represents horizontal motion perpendicular to that of the first trace. The reference traces are produced by "fixed" mirrors rigidly attached to the accelerograph frame; they are used to correct for film distortion or transverse slippage of the film as it moved through the accelerograph. Time traces show a pulse approximately every half centimeter, the distance between pulses representing a half-second time interval. The time traces are generated by an internal timer that is more accurate than the recording speed is constant. Some accelerograms also show WWVB time code traces with which the date and time of motion can be determined. All these analog traces (except the WWVB trace) must be digitized before they can be processed by the computer programs.

Digitization is currently performed at IOM-TOWILL corporation in Santa Clara, California with automatic trace-following, laser scanning equipment. The laser scanner provides much better resolution than can be attained with the manual digitization that was used in the past, but it can digitize at most a 10 square centimeter area at one time. Between 500 and 800 points per centimeter of trace length are digitized with a resolution of 1 micron (10**-6 meter) and an RMS error of the order of 10 microns. Records longer than 10 centimeters are divided into two or more "frames" that are digitized independently. Transverse "butt" lines are scribed onto such records as dividers between frames. All the continuous traces on each frame are digitized with half an inch of overlap beyond the butt lines and the butt lines are also digitized. The overlap and butt lines are used by the BUTTER program to rejoin the separately digitized frames.

The scanning laser automatically digitizes the main and reference traces once the operator starts it on each trace in turn. A manual intervention is required for handling decision making such as crossing intersecting traces, getting over a gap or faint area in the trace, or resolving the effect of dirt and scratches on the film. The time trace is also measured manually at the leading edge (or any repeating clearly-defined corner) of each time mark.

IOM-TOWILL produces 9-track tapes to transfer the data to NSMDC. The laser scanner measures on the order of 6,000 points (12,000 numbers) per trace per 10-centimeter frame. Each frame is written to tape as a separate file. Each trace within that frame is preceeded by a 60-word header. The numbers are written in binary form and must be converted from the binary form used by the IOM-TOWILL computer to the internal binary form used by whichever computer will read the data. (Why wasn't the data formatted to ASCII or EBCDIC characters? Too much data?)

The tape is read by the MTDUP tape utility program on the PDP 11/70 computer at NSMDC. Then the BUTTER program reformats each number to PDP internal format and realigns adjacent frames. Next, the REFORM program must reformat BUTTER output so it can be read by SCALE. (The messy MTDUP-BUTTER-REFORM sequence will be revised just as soon as April can get time to do it.) The whole sequence of programs for processing data that has been digitized at IOM/TOWILL is:

MTDUP
BUTTER
REFORM
SCALE
HIFRIC
CORAVD
PHASE3
PHASE4
and PHASE6.

The support programs TSPLOT, BBFILE, REFORM and BWRITE can be used at various stages in this process. Note that the PHASE5 program is not available at NSMDC yet.

1.2.2 Digital Records

Recently developed instruments record directly onto digital magnetic tape. These instruments acquire data continuously, always saving the last several seconds of data in a buffer so that motion leading to instrument triggering can be captured with the rest of the record.

The computer programs described in this report were established to process digitized analog records, but they can be used to process digitally recorded data as well, once the data has been read and converted to an appropriate format. The programs have been used to process data from Sprengnether DR100 instruments and will be modified as necessary in the future to accommodate data from other digital recording instruments. DR100 tape cassettes are read using the program DR11K, next, the several components of motion that were recorded together are separated ("demultiplexed") by the CRTAPE program, and the data are then reformatted by the program named DR100. These three programs are not described in this report. For more information about them, refer to the text files at DR0:[2,2]DR11K.DOC, DR0:[2,2]CRTAPE.TXT and DR0:[2,2]DR100.DOC or talk with Ed Cranswick, Chuck Mueller, or Joe Fletcher.

DR100 recorders are either connected to velocity transducers or to force-balance accelerometers. The instrument correcting algorithms in HIFRIC are inappropriate for the velocity data and they aren't entirely suitable for the force-balance accelerometers either. Bill Joyner and Chuck Mueller have considered algorithms that can be incorporated into HIFRIC

eventually if we ever need them, but since the instrument responses for both types of DR100 transducers are flat from 2 to 50 Hertz, no instrument correction is necessary as long as frequencies outside 2 to 50 Hertz are beyond the range of interest. Thus, the data files generated by the DR100 program should enter the AGRAM processing at CORAVD.

The sequence of programs for processing digital accelerometer data is:

DR11K
CRTAPE
DR100
CORAVD
PHASE3
PHASE4
and PHASE6.

1.3 Background

The AGRAM programs are revisions or rewrites of the BUTTER and PHASE1 through PHASE6 programs that were installed at the Lawrence Berkeley Laboratory (LBL) computing center long before the National Strong Motion Data Center (NSMDC) existed. The LBL versions of the programs were used from 1974 through 1981 in preparing the strong motion data reports published by the Seismic Engineering Branch of the USGS. The PHASE1 through PHASE4 programs were originally developed at the Earthquake Engineering Research Laboratory of California Institute of Technology although modifications were made by Seismic Engineering Branch to upgrade procedures and to adapt the programs for use with the computers at LBL. PHASE5 and 6 were developed by Virg Perez. BUTTER was developed by Gerry Brady and by W.R. Roseman of IOM-TOWILL.

All the programs were revised in order to transport them from the CDC 7600 computer at LBL to the PDP 11/70 at the NSMDC, and although they were changed a great deal in the process, they retained the same names as the programs had at the LBL computing center. Now the programs are being revised to use a new data file format, to separate the processing functions into smaller and more independent programs, and to use new processing algorithms that have been developed by Michael Raugh and others.

Those programs that have been revised to the point where they produce blocked binary data files rather than the card-image text files produced by the LBL programs have been renamed as a reminder that the programs are now really quite different from their PHASE counterparts at LBL or CalTech. SCALE has replaced PHASE1 and HIFRIC and CORAVD have replaced PHASE2. BUTTER, BWRITE, and the PHASEs have yet to be revised to the same extent as SCALE, HIFRIC and CORAVD have been. They are documented only

briefly here, but their documentation will improve when the programs themselves do.

1.4 Invoking the programs

The programs are not installed yet since they are still changing so frequently. Until they are installed, you must type the directory identification along with the program name to invoke any of them. The disk, directory and program names for all the programs discussed in this report are the following:

DR2:[110,110]TSPLOT, STSP, VTSP, CTSP, and ITSP
DR2:[110,110]BBFILE
DR2:[110,110]BWRITE
DR2:[110,110]REFORM
DR2:[110,110]BUTTER
DR2:[110,110]SCALE
DR2:[110,110]HIFRIC
DR2:[110,110]CORAVD
DR2:[110,110]SLOWAVD
DR2:[110,110]PHASE3
DR2:[110,110]PHASE4

Most of these programs (and all of them, eventually) receive their run parameters from the command line in which they are invoked although other means for providing run parameters to the programs may be added in the future. At present, if no run parameters are given on a command line that invokes an AGRAM program, the program will display instructions. In the instructions displayed by the programs and in the instructions given in this report, items that the user must provide are shown in brackets. (Note that you do not type the brackets when providing such an item.) Items in square brackets ([]) are required, items in angle brackets (<>) are optional. Items in quotes ("") are literals.

The command lines for these programs are arranged similarly to those for the "MCR" commands provided by the RSX operating system in use at NSMDC. Output files are listed on the left-hand side of an equal sign and input files are listed on the right-hand side of the equal sign. The file names may be abbreviated as in MCR commands: if the directory is not specified for the first file name given on the command line, the user's current working directory is assumed; if the directory or directory and file name prefix is not specified for any but the first name given, the directory and prefix used for the preceeding file name is assumed. If neither the first file name prefix nor the input file name prefix is specified, "TEMPORARY." will be used. If a file name suffix is not specified, a default will be selected that depends on which program and which file is involved.

Any other run parameters a program requires are listed on the command line after the file names. (Unlike the MCR command lines in which run parameters are appended to the file names as "switches".)

Commands that do not fit on one line can be continued by using a hyphen (-) as the last character on the line to be continued.

1.5 Warnings

The file names given in command lines must not include version numbers. If this restriction causes problems, it can be changed. The file names cannot include wildcards ("*") either.

The programs that don't interpret their command lines at all yet are: BUTTER, BWRITE, and the PHASEs. BUTTER and BWRITE get their run parameters by prompting the user. The PHASEs get their run parameters from the header lines in their data files.

Some of these programs have sections of code that haven't been tested yet. If you receive a message stating that you are using untested code, please show April how the program was run so she can use your files and command lines for a test case.

CHAPTER 2

Data Files

2.1 Overview

The data files processed by the AGRAM programs conform to a standard file organization intended for all time-series disk files at the NSMDC. These files are referred to as "blocked binary" data files. Their contents are in binary form and are arranged as a series of fixed-length, 512-byte blocks that may be accessed randomly or sequentially. The first few blocks of each file are reserved for auxiliary information that supplements or describes the time-series data in the remainder of the file. Refer to appendix B for description of the content of the header blocks.

Each blocked binary file contains time-series data for a single component of motion and a single type (acceleration, velocity or displacement) of motion. The NSMDC data file czar (Ed Cranswick) wants to change this, however, and eventually store data for three components in each file as was done earlier.

Input/output operations are much faster with the blocked binary files than with the card-image text files that had been processed by the PHASE programs since there is no translation back and forth between internal (binary) form and external (character code) form when reading or writing binary files. The blocked binary files take less disk space than did the card-image text files too. Blocked binary files are also arranged to take advantage of the direct, disk-to-memory transfers provided by some operating systems like the RSX operating system at NSMDC. This method avoids the transfer time and memory space required with the intermediate buffers used by standard FORTRAN input/output operations. The input/output can still be done with standard FORTRAN, though, as is done in the versions of the AGRAM programs on the VAX at NSMDC.

Since the data is in binary form, the blocked binary files must be formatted somehow before their contents can be displayed, processed with a text-editor, or transported to another computer. The BBFILE and REFORM programs do such formatting although they still aren't as versatile as they need to be. BBFILE prepares a text-file dump of all or part of a blocked binary file. REFORM reformats between various types of text data files and blocked

binary data files.

2.2 Raw Data

Raw data comes from the digitizing facilities or recording instruments in a variety of formats and on a variety of media. IOM-TOWILL digitized data is written on magnetic tape, DR100 data is recorded on tape cassettes, manually digitized data from Willy Lee's digitizer downstairs is written on a floppy disk, and we still occasionally receive data on punched cards. Each type of incoming data requires a special program or programs to read and translate the data. Talk with Gerry Brady to learn the format of the IOM-TOWILL tapes; they are read by the MTDUP program and are translated by the BUTTER program. Talk with Joe Fletcher or Larry Baker to learn the format of the DR100 cassettes; they are read by the DR11K program and are translated by the program named DR100. Talk with Jack Boatwright to learn the format of the floppy disks from the digitizer downstairs. Jack has a program that will read the disks, but it does not write blocked binary data files (yet).

2.3 DR100, HIFRIC and CORAVD files

Time-series data in files generated by the DR100 program are integers (256 2-byte values per block) and time series data generated by the AGRAM programs are reals (128 4-byte values per block). DR100, HIFRIC and CORAVD output files contain acceleration, velocity or displacement values only: time is at even intervals, the time interval being stored in a header block.

Some of the recording instrument characteristics that were once intended to be stored in header blocks are now stored in a separate, "DBS", file for DR100 data. Ed Cranswick and Chuck Mueller oversee the DR100 and DBS files. Instrument characteristic items are still stored in the header blocks of AGRAM files.

2.4 BUTTER and SCALE files

The time-series data in BUTTER and SCALE output files include the abscissa for each sample since the data are not at even intervals. Data items are real and occur as a series of (x,y) pairs. In BUTTER files, x and y are in digitization units (=microns if the digitization was done at IOM/TOWILL). In SCALE files, x is in seconds, and y is scaled, uncorrected instrument response in cm/sec/sec offset by a constant value (the constant

being in a header block).

A BUTTER output file contains data for all the traces on a digitized record, but a SCALE output file contains the data for just a single accelerometer trace.

2.5 File Names

DR100 File names have the form JJJHHMMSC.STA, where JJJ is the Julian day, HH is the hour, MM is the minute, S is a letter A through T, C is the component number, and STA is the 3 letter station name. The second, S, is represented as the letters A-T, each letter representing a 3 second interval. The component number, C, is 1,2, or 3 for acceleration; 4,5, or 6 for velocity; and 7,8, or 9 for displacement.

AGRAM output files may be renamed according to the DR100 convention once processing is complete, but during processing, another convention is used which allows us to distinguish from which stage (or PHASE) in the processing the file comes.

There is no convention for AGRAM file name prefixes, although "TEMPORARY." is often used since that is the prefix the programs assign by default. In the suffix, however, the first character indicates what sort of data is in the file and the second and third characters are numeric digits that indicate which trace from among the several traces that were recorded together is contained in the file. SMA records usually consist of traces from three orthogonal transducers with the first and third traces for horizontally oriented transducers and the second trace for a vertically oriented transducer. But things are not always so simple! There are remote recorders in the field that can record up to 16 traces on a single record. Transducers are not always arranged in orthogonal triplets, either.

Here follows a table relating the AGRAM file name suffixes and the programs that produce and read those files. The TINKER program shown in the table is not discussed elsewhere in this report; it is primarily used to process DR100 data.

file suffix	producing program	file content	reading program
.BUT	BUTTER	Butted IOM digitized frames.	REFORM, then SCALE
.R01	SCALE	Uninterpolated instrument response.	HIFRIC
.I01	HIFRIC	Interpolated instrument response.	TINKER
.C01	HIFRIC	Interpolated, instrument corrected case acceleration.	CORAVD, or TINKER
.A01	CORAVD or SLOWAVD	Baseline corrected acceleration.	phase 3,4,6, or TINKER
.V01	"	Velocity.	"
.D01	"	Displacement.	"

CHAPTER 3

BUTTER program

BUTTER reformats data read from an IOM-TOWILL tape and butts separately digitized frames together. It hasn't been changed to generate blocked binary output files yet. Ask Pete Mork or Chuck Mueller how to use it.

BUTTER needs work!
==========

- The IOM-TOWILL equipment can digitize an area 10 by 10 centimeters. For those records that are longer than 10 cm., BUTTER reconnects several separately digitzed frames. The program does not yet have the capability to reassemble records that are wider than 10 cm., but it will need to do so to process records from the newer remote recording instruments (up to 16 accelerometer traces on a 7-inch wide record).
- Output from BUTTER is given in text files rather than the blocked binary data files processed by the other programs in the series. These text files must be sent through the REFORM program to reformat them to the blocked binary files SCALE reads. (Internally, BUTTER processes the data in blocked binary files, but it formats the data just for output!)
- This version of BUTTER, unlike the one at LBL, discards all but one reference trace.
- The current version cannot generate a dummy reference trace properly when processing multi-frame records.
- The prompts BUTTER gives the user are not very clear. Commonly used run parameters ought to come from the command line and prompts for nonstandard situations ought to be easier to understand.
- BUTTER hasn't been revised much at all since Adolf Oliver and Larry Baker first adapted it for the PDP 11/70 computer. It uses outdated i/o subroutines, prompting subroutines and task building techniques that can be cleaned up to make the program smaller and more consistent with the other programs in the series.

CHAPTER 4

SCALE program

4.1 Overview

SCALE scales digitized accelerometer traces to seconds and cm/sec/sec. It transfers data from reformatted BUTTER files to files suitable for input to the HIFRIC program, separating the data into a separate file for each accelerometer trace given in the BUTTER file.

In addition to the accelerometer traces, most records contain timing marks and several fixed reference traces that are also digitized. SCALE makes longitudinal time corrections based on the timing marks (assuming that the clock runs more accurately than the transport mechanism) and corrects for possible transverse slippage by subtracting the reference trace from all accelerometer traces. For digitizations of records that have been photographically reduced, optical distortions are minimized by subtracting the reference trace.

The results from SCALE are often referred to as "uncorrected" data in the sense that no modifications have been made that involve any hypotheses as to the character of the ground motions or instruments involved. The data have been corrected merely for uneven film transport and for transverse slippage of the film as it moved through the recorder.

4.2 SCALE process for each accelerometer trace:

- Identify the reference trace nearest the accelerometer trace being processed by comparing the differences between the ordinates of the first point in the accelerometer trace and the first point in all the reference traces given.
- If the first point in the accelerometer trace occurs before the first point in the reference trace, extend the reference trace along the line defined by the first and second points in the reference trace. Extend the reference trace at the end of the record too, if necessary.
- Smooth the values of the time tick abscissas with a 1/4, 1/2, 1/4 running mean (also called a Hanning filter.)

- If the first point in the accelerometer trace occurs before the first time tick, use the interval between the first and second time ticks to extend hypothetical time ticks back beyond the beginning of the accelerometer trace. Use the interval between the last two actual time ticks to extend the time ticks beyond the end of the record too, if necessary.
- Perform the following steps for each point in the accelerometer trace.
  . Subtract the reference trace as follows: Identify the two reference trace points that bracket the data point. Use linear interpolation between the two reference trace points to calculate a reference trace ordinate value at the same abscissa as the data point. Subtract calculated reference trace ordinate from the data ordinate.
  . Subtract a constant also, to avoid numeric overflow problems while calculating the mean value of the trace. Use the difference between the first ordinate of the acceleration trace and the first ordinate of the reference trace as the constant.
  . Convert ordinate value from digitzer units to cm/sec/sec. Use the sensitivity of the recording transducer (cm/g) and the digitization units per centimeter in this conversion. (The default value for digitization units per centimeter is 10,000 microns/cm which is appropriate for IOM/TOWILL digitizations)
  . Convert abscissa from digitizer units to seconds using the two time ticks that bracket the data point. Calculate the time at the two ticks using the first point in the data trace as time zero, and using the time between consecutive ticks (usually 0.5 second) given as an input parameter. Interpolate linearly to calculate the time at the data point.
- Calculate the mean value of the entire accelerometer trace. The mean is saved in a header block in the output file and is subtracted from the ordinate of every point in the trace by the next program (HIFRIC, TSPLOT, or REFORM) to process the data.

4.3 To invoke SCALE, type --

```
[110,110]SCALE <output file(s)>,<user msg>=          -
               <butter file>,[time between ticks],   -
               [sens1],[sens2],[sens3]
```

Where :

<butter file> is a data file produced by BUTTER then reformatted by REFORM into a blocked binary file.

[time between ticks] is the time, in seconds, between the tick marks on the record. Usually 0.5.

[sens...] are the recording transducer's sensitivities in cm/g, and are given in the same order as the transducer's traces occur on the digitized record. First transducer corresponds to the

top-most accelerometer trace, last transducer to the bottom-most trace.

Example:

```
[110,110]SCALE temp.=butter.tmp,0.5,1.92,1.85,1.77
```

4.4 SCALE Still needs work:

- If there is no reference line, use the time ticks as a reference line. If the time ticks are missing too, use the linear least squares fit to the data as a reference line and assume the recorder progressed at a constant one centimeter per second.
- If the data trace extends beyond the first or last point in the reference trace, SCALE extends the reference line using the first or last pair of points in the reference trace. Program aborts (zero divide) if the pair are not really distinct points but just multiple digitizations of the same point. Similar problems occur if the first or last time tick is digitized more than once too. Pete Mork gets around this bug by editing BUTTER's text output file, but once BUTTER is changed to produce blocked binary data files, this problem had better get fixed!
- instead of extending the reference trace with the slope of the first or last pair of reference points, use the slope of the first or last 1 centimeter worth of reference line. But if using time ticks as reference, continue to use the slope of the first or last pair.
- Extend time ticks with average of the nearest few intervals rather than just the one nearest interval?
- Add an option to allow smoothing the reference trace for use when dealing with hand digitized data? Also, may want to subtract out the llsqf of the reference line from everything in a hand-digitized data file as is done in BUTTER for the IOM digitized data. These functions probably should be performed in a separate program, a BUTTER counterpart for hand digitzed data.
- WW, the tick varying diagnostic, is calculated from the smoothed tick values. It should really be done on the unsmoothed values. And once its fixed, increase the warning level from 2.5% back to 5%.
- Get time between ticks, sensitivity values, and digitization units/cm from the input file's header blocks. Must decide where and how to put them into the headers first, though.

4.5 Differences between SCALE and PHASE1

- SCALE input and output files are blocked binary files: PHASE1 input and output files were card-image text files. Input files for BOP1, an intermediate version of the program, were text files, output files were blocked binary files, but with non-standard headers.
- PHASE1 resets backsteppping points, SCALE does not.
- SCALE should operate on data that BUTTER has not decimated so that HIFRIC can make best use of the densely digitized data. PHASE1 usually processed data on which the decimation option in BUTTER had been applied.
- PHASE1 smoothed the reference trace, SCALE does not.
- When a data trace extends beyond the first or last time tick, PHASE1 extended the ticks by using the average of all the tick intervals, SCALE uses the nearest (first or last) interval.

CHAPTER 5

HIFRIC program

5.1 Overview

HIFRIC applies interpolation, instrument correction and a high-cut filter to SCALE data. Options provide alternative instrument-correcting and filtering algorithms. With the standard option, HIFRIC applies Mike Raugh's time-domain algorithm; with another option it applies Bill Joyner's frequency domain algorithm; and with a third option the instrument correction and filter are suppressed altogether, providing interpolated, uncorrected data.

Both correcting algorithms apply a second-order differential equation representing motion of a viscously-damped, one-degree-of-freedom oscillator (see page 46 of reference [1].) The two algorithms produce almost identical results when applied to densely digitized SMA data with standard filter parameters, but the standard time-domain method runs faster than the frequency domain method. More investigation needs to be done with both methods before we understand what their limitations are when used with other types of data, with non-standard filter parameters, and with which situations, if any, one method is more appropriate than the other.

With the time domain algorithm, the data are interpolated to an even sampling of 600 samples per second, then the frequencies from zero to Nyquist are divided into six equal-width bands. The first band is that in which instrument correction is performed. The second is a transition band providing a cosine taper from full to zero frequency response. The remaining bands are anti-aliasing (i.e., near-zero frequency response). After filtering, the data are decimated by removing 2 out of every 3 points, reducing the sampling density from 600 samples per second to 200 samples per second. The dense sample rate of 600 and decimation factor of 3 are used in routine processing, but different values for these two parameters may be provided for non-standard processing.

The time-domain method imposes more restrictions on its input parameters than does the frequency-domain method. These restrictions aren't inherent in the method (according to Bill

Joyner), but stem from the way it was coded. The restrictions are:

- The width of the filter's transition band must be the same as the width of the pass band. If the program is to allow a transition band smaller than the pass band, more weights may be needed for the convolution operators than the program now provides. With more weights, the program will run more slowly.
- The band width (cycles per second) must divide evenly into the sampling rate (samples per second).
- The number of bands (6 in standard processing) must be an even multiple of two if decimation is performed, and the ratio of dense to decimated data must equal bands/2.

With the frequency domain method, equal-lengthed segments of densely interpolated time series data are transformed to the frequency domain with a Fast Fourier Transform. Transformed values are instrument corrected, filtered, then transformed back to the time domain. The separately filtered segments of data are fitted back together using an "overlap-add" method similar to that described in reference [9].

As is the case with the time-domain method, the range of parameters for which the frequency-domain method is appropriate has not been investigated carefully yet. The lengths of the separate segments of data and their overlapping area are set by the program to be 1024 points and 256 points respectively. These lengths may not be suitable for atypical data or atypical filter bands. Until the code is changed to allow the user to specify the segment and overlap lengths, users should beware of results unless run parameters are such that:

- The cutoff frequency at which the filter's transition taper begins should be less than or equal to half the frequency at which the taper ends.
- Frequency at the end of the taper should be less than half the sampling rate.
- Sampling rate between 50 and 1000 samples per second.
- Instrument period and damping are values typical of SMA accelerographs (period between 0.04 and 0.1 second; damping about 0.6 of critical damping).

HIFRIC will process DR100 data but the frequency response of the DR100 force-balance accelerometers are not accurately represented by the damped harmonic oscillator equation. And the equation is not at all appropriate for the DR100 velocity transducers. Bill Joyner and Chuck Mueller have devised a method for correcting the velocity data, but it has not been coded (yet?) since they feel that the correction would make very little difference in the data.

5.2 Differences between HIFRIC and the instrument correction in PHASE2

The differentiations required by the damped harmonic oscillator equation were calculated using a centered-difference method in the PHASE2 program. More accurate approximations of the derivatives are used in HIFRIC. The standard method incorporates Fourier differentiation (i.e., d(e**iwt)/dt = iw e**iwt) in convolution operators, which are applied in the time-domain, to approximate the first and second derivatives required. The alternative method applies Fourier differentiation in the frequency domain.

An anti-alias filter was applied in PHASE2 with an Ormsby convolution. In HIFRIC, a filter is incorporated in the same convolution that calculates the derivatives in the time domain. With the alternative method, the high frequency content is simply set to zero in the frequency domain.

5.3 Standard HIFRIC process:

1) Calculate weights for three convolution operators. Each operator has 61 weights which will span 0.1 second of 600 sample-per-second data.
2) Adjust input data by subtracting out the SCALE offset or by applying the DR100 coil constant and gain. The data now represents measured instrument response that has been scaled to approximate the acceleration of the instrument case. For a linear damped harmonic oscillator whose relative displacement is x(t), and whose natural frequency, w, is sufficiently high, the approximate case acceleration, y(t), is -w**2 * x(t). The instrument response is not entirely proportional to the acceleration of the instrument case, however, especially at high frequencies, so instrument correction is applied in step 4.
3) Interpolate linearly to 600 samples per second.
4) Apply convolution operators at every third sample of the input data (reducing the density to 200 samples per second), to compute z, z′, and z" at that point, where z is a high-cut filtered version of the approximate case acceleration, y. Combine z and its two derivatives according to the damped harmonic oscillator equation, where corrected acceleration =

$$z + 2*z'*n/w + z''/w**2$$

n is the fraction of critical damping of the instrument, and
w is the natural frequency of the instrument in radians per second.

This 4-th step simultaneously provides:

- instrument correction between 0 and 50 Hertz;
- anti-alias filter with cosine taper between 50 and 100 Hertz, and stop band from 100 to 300 Hertz;
- decimation to 200 samples per second.

The high-cut filter removes any noise in the digital data between 100 Hertz and 300 Hertz, including that which could have been aliased into this range by the earlier interpolation to 600 sps. It also guarantees that there is negligible energy transferred to the 0 to 100 Hertz range by the subsequent decimation.

5.4 To use HIFRIC, type --

```
[110,110]HIFRIC <output file>,<usermsg>=          -
                [SCALE file],[period],[damping], -
                <sps>,<ndense>,<fc>,<fz>, -
                <non-standard flag>
```

[period] is the period of the recording transducer, in seconds.

[damping] is the damping of the recording transducer, as a fraction of critical damping.

[period] and [damping] are required on the command line only if the period and damping values in the header block in the input file are incorrect or missing.

<sps> is the intended sample rate for the output data in samples per second. Default =200.

<ndense> is the ratio of the dense sample rate to the final sample rate, <sps>. <ndense> must be an integer. Default=3. (The instrument correction and filter are applied to the more densely sampled data.)

<fc> to <fz> is the transition band, in Hertz., for the filter's taper. Default = 50. to 100.

The optional non-standard flags are "INTERP" and "FDIC". With INTERP in the command line, HIFRIC will interpolate, but not instrument correct, in which case the [period] and [damping] values need not be given. With FDIC, HIFRIC will use the alternative frequency-domain instrument correcting method rather than the standard time-domain method.

The meaning of the numeric parameters ([period],[damping], <sps>,<ndense>,<fc> and <fz>) depend on the order in which they are given, so give a null value (two consecutive commas) if you wish to use a default value among other numeric parameters you wish to set.

Examples --

```
[110,110]HIFRIC fromf2a.c01=fromf1.r01,.040,.570
[110,110]HIFRIC fromf2a.c01=fromf1.r01,interp
[110,110]HIFRIC fromf2a.c01=fromf1.r01,.040,.570, -
                ,,40,90,fdic
```

5.5 HIFRIC still needs work:

- Some records require that an operator manually digitize a sharp peak where the trace is too faint for the laser scanner to follow. The operator cannot digitize as densely as the automatic scanner can, so the peak will appear sharper and will contain higher frequency Fourier components than it would have if its two neighboring points were more closely spaced. The high frequency components, even those falling in the lower portion of the transition band, are amplified by the instrument correction. Amplification of almost five times will occur in frequencies between 60 and 70 Hertz! Perhaps the interpolation in HIFRIC should not be linear about a manually digitized peak.
- The subroutines (SETWTS and FDIC) that apply the two alternative correction algorithms can be modified to alter characteristics like the number of convolution weights, the overlap and segment size, and so forth. Some of the choices for tuning these algorithms should be run options rather than predetermined code in order to handle atypical data.
- Both algorithms assume that the time series is zero before the first sample of actual data. Perhaps leading points should be set to a cosine taper to zero. Or perhaps the first few points should be dropped (M. Raugh did delete NWTS points from the end of the series.)
- The convolutions done in subroutine OPRTR should accumulate positive and negative sums separately, then add the two sums together at the end (according to M. Raugh.)

CHAPTER 6

CORAVD program

6.1 Overview

CORAVD calculates velocity and displacement from acceleration and optionally performs baseline correction of two possible types. One option makes a linear correction to the velocity and another option filters long periods from velocity and acceleration.

Analog records do not begin instantaneously with the beginning of the earthquake motion, but only after the motion has become strong enough to trigger the recorder. The linear baseline correction option establishes a reasonable value for the acceleration and velocity at the beginning of such a record provided that a least squares straight line fitted to some portion (or all) of the velocity time series is a reasonable representation of the trend of the velocity. The fitted line is subtracted from the velocity and a constant, equal to the slope of the line, is subtracted from the acceleration. The portion of the velocity to be fitted is specified as a run parameter. No attempt is made to estimate an initial value for the displacement. The displacement is calculated from the velocity after the velocity has been corrected.

Digital recorders with pre-event memory show, in their records, several seconds of motion that occurred before triggering. Initial values of zero for acceleration, velocity and displacement are appropriate for these records (provided the recording system operated as designed) and the linear correction should not be made for them.

The linear correction gives good results for some accurately digitized records, but many records will require that long periods be filtered from the data before reasonable displacements can be calculated or before reasonable spectra can be calculated in the PHASE3, 4 and 5 programs. Several different filtering schemes are provided by the program, but for routine processing a bidirectional Butterworth filter is used. The other filter algorithms that are available are a unidirectional Butterworth filter, an FFT filter, and an Ormsby filter. (The Ormsby filter is incredibly slow the way it is coded now, though. Please don't

submit any jobs that use Ormsby during regular work hours.) Although high frequencies have usually been filtered from the data in HIFRIC, CORAVD provides an option for high-cut filtering for those situations in which the HIFRIC processing has been bypassed. Beware against refiltering high frequencies if they've already been filtered in HIFRIC though; it would distort the data unnecessarily.

SLOWAVD is a slower, virtual-array version of CORAVD that can filter much longer records than can be filtered by CORAVD. Although a future version of CORAVD may provide for filtering long records, at least with the Butterworth filters, the current versions of the filtering routines will truncate a time series if all the data and necessary work space will not fit in main memory at one time.

6.2 Guidelines for selecting CORAVD options

By default, CORAVD does no filtering and applies the linear baseline correction, fitting the entire length of the velocity time series. Users must reconsider for each particular earthquake, or perhaps for each particular record, whether or not such processing is appropriate. The need for different handling will often show in a plot of the results from CORAVD where long period waves that were not apparent in the uncorrected acceleration can be seen in the integrated velocity or displacement or when the displacement at the end of the plot is much different than could really have occurred. More subtle long period noise will show in plots of PHASE3 results although an inexperienced user would have difficulty recognizing whether or not the long period response spectra from unfiltered data were misleading.

Once a need for filtering is recognized, the user must select a filter that cuts out unwanted low frequencies but that also cuts out (or distorts) as little of the remaining higher frequencies as possible. This requires some compromise and experimentation since a filter with a steep cutoff introduces inaccuracies in the frequencies that pass through the filter. Although accuracy in the passband is better with a filter that gradually tapers the cutoff, a wide transition (or "roll-off") band may retain too much of the unwanted frequencies or too little of the higher frequencies. Guidelines for selecting a cut-off frequency and transition bandwidth are given in reference [10]. It is necessary to rely on experience, to experiment with the filter parameters (filter, check the plotted results and refilter the original input if necessary), and to rely on advice from seismologists and geophysicists to obtain realistic displacements. See references [4] and [5] for a detailed description of the decisions made while processing a recent set of records.

Special studies may require the use of a different filtering

algorithm than the bidirectional Butterworth filter used in routine processing. The several other filter algorithms are provided for this purpose, but none of them have been tested yet.

A time series that clearly requires long-period removal is not suitable for straight line fitting if periods longer than about a quarter the duration of the line are present. The orientation of such a line would depend substantially on the position of the line within the long period cycles and would not represent a baseline for the time series. Consequently, routine use of the linear baseline correction is not recommended when the low-cut filter option is used, though there may be cases where that is appropriate.

6.3 Standard CORAVD process:

1) Integrate acceleration to compute velocity. Use trapezoidal integration and use zero as the initial velocity.
2) Subtract the linear least squares fit of the velocity from the velocity to establish an initial value for the velocity. Fit the entire velocity time-series unless user has specified that just a portion of the velocity, maybe just the quiet period at the end of the time-series, be used.
3) Subtract a constant from the acceleration, the constant being the slope of the line fitted to velocity.
4) Filter low frequencies from velocity and acceleration, if necessary. Use the same filter parameters for both.
5) Integrate velocity to compute displacement, using zero as the initial displacement.

6.4 Differences Between CORAVD and the baseline correction in PHASE2

PHASE2 calculated velocity from a filtered version of the acceleration, filtered the velocity, calculated displacement from the filtered velocity and filtered the displacement. This refiltering of a time series determined from an already filtered time series magnifies any distortion inherent in the filter, so CORAVD filters each time series just once, or preferably not at all. When filtering is used in CORAVD, velocity is calculated from acceleration before acceleration is filtered; displacement is calculated from the filtered velocity and is not refiltered itself. PHASE2 used the Ormsby filtering algorithm but in routine processing CORAVD uses the significantly faster bidirectional Butterworth filter algorithm. A brief discussion of the advantages of the Butterworth filter in addition to its increased speed is given in reference [11].

The table on the following page compares the processing steps once used in PHASE2 to the steps now used in CORAVD. In

the table, "llsqf" refers to a linear least-square fit, "acc(t)" refers to acceleration as a function of time, "vel" refers to velocity, "disp" refers to displacement, and y(t) refers to a straight line function. "Option A" is the linear correction option and "option B" is the low-cut filter option.

Step	PHASE2	CORAVD
1	- Interpolate to 100 equally spaced samples per second, filter out high frequency noise, and perform instrument correction.	- No. The counterparts of these calculations are done in HIFRIC now. The interpolation is now at 200 samples per second.
2	- Remove linear trend from acceleration: llsqf [acc(t)] = y(t) = mt + b acc(t) = acc(t) - y(t)	- No. This has already been done in BUTTER.
3	- Calculate velocity from acceleration: $vel(t) = \int acc(t)\, dt$	- Yes.
4	- Fit a straight line to the velocity: llsqf [vel(t)] = y(t) = mt + b	- Yes, with option A.
5	- Subtract the slope of the line fitted to velocity from the acceleration: acc(t) = acc(t) - m	- Yes, with option A.
6	- Subtract a baseline from acceleration: acc(t) = acc(t) - baseline(t) where the baseline is acc(t) processed as follows: . pad both ends of the curve with a mirror image of itself, . smooth with a 0.4-second, equal-weight running average, . decimate, retaining every 10th point, . apply low-pass Ormsby filter.	- Option B filters long periods from acceleration, but uses: . Padding with zeros, . No smoothing, . No decimation, . Butterworth high-pass filter applied directly to acceleration rather than Ormsby low-pass filter applied to a baseline.
7	- Recompute velocity from the corrected acceleration: $vel(t) = \int acc(t)\, dt$	- No.
8	- Fit another straight line to the recomputed velocity: llsqf [vel(t)] = y(t) = mt + b	- No. Already have a fit from step 4 since CORAVD velocity is not recomputed in step 7.
9	- Subtract the linear trend from velocity: vel(t) = vel(t) - y(t)	- Yes, with option A.
10	- Subtract the slope of the line fitted to velocity from the acceleration (a second time!): acc(t) = acc(t) - m	- No.
11	- Subtract a baseline from velocity: vel(t) = vel(t) - baseline(t) where the baseline is determined from velocity as it was for acceleration.	- Option B applies the same filter to velocity as it does to acceleration.
12	- Calculate displacement from velocity: $disp(t) = \int vel(t)\, dt$	- Yes.
13	- Subtract a baseline from displacement: disp(t) = disp(t) - baseline(t) where the baseline is determined from displacement as it was for velocity and acceleration.	- No.

figure 6a: Compare processing steps in PHASE2 and CORAVD.

6.5 To use CORAVD, type --

```
     [110,110]CORAVD <output files>,<usermsg>=[input file],
                     <begin fit>,<end fit>,<taper fit>,
For SLOWAVD, type --
     [110,110]SLOWAVD <output files>,<usermsg>=[input file],
                      <begin fit>,<end fit>,<taper fit>,
                      <filter type>,<filter parameters>,
                      <filter type>,<filter parameters>

Examples:
     [110,110]CORAVD temp.=fromf2a.c03, 6.0,40.0

     [110,110]SLOWAVD temp.=fromf2a.c03, 0,0, BI,0.17,2
```

<output files> =
names for the 3 output files. If these aren't given on the command line, or if just the prefix or just the suffix is given, missing parts of the file names will be selected by the program. If missing, the output files' prefix will be the same as the input file's prefix. If the output files' suffixes are missing, the first character in the suffix will be 'A', 'V', and 'D'. The second and third characters in the suffixes will be the same as in the input file's suffix and are assumed to be a number that indicates which component the data is for.

<usermsg> =
Name of an optional disk file to contain run messages and diagnostics that would normally go to an interactive user's terminal or to a batch job's log file.

[input file] =
Name of the input file. This file should be an output file from the HIFRIC or DR100 programs.

<begin fit> and <end fit> specify the endpoints, in seconds, of the least squares fit line to be used for the linear base line correction. If these are not given, the entire velocity trace will be fitted. If they are both given as zero, then no linear correction will be performed.

<taper fit> specifies the fraction of the least-squares fit range in which to apply a cosine tapered weighting factor. The taper is applied to both ends of the fit range. <taper fit> must be between 0.0 and 0.5. Default value =0.0.

<filter type> =
"BI", "UNI", "FFT", or "ORM" for bidirectional Butterworth filter, unidirectional Butterworth filter, FFT filter, or Ormbsy filter.

If two sets of filter types and filter parameters are given, the first is for a low cut (or high pass) filter and the second is

for a high cut (or low pass) filter.

<filter parameters>
Filter parameters are somewhat different for each type of filter. They specify the transition band between the pass band and stop band of the filter. (Although other filter tuning parameters may be added later.)

For the FFT and Ormsby filters, the transition is specified by giving a corner frequency, <corner>, at which the filter should start to roll off and a transition bandwidth, <tband>, after which the response is zero. The transition is a cosine taper in the FFT filter. In the ORM filter, the transition is a linear ramp with a short parabolic section on both ends where the ramp meets the pass band on one end and the stop band on the other end.

For the Butterworth filters, the transition is given as a corner frequency, <corner>, and a rolloff order, <n>. Unlike the corner frequency given for FFT and ORM filters, <corner> for the BI filter is the frequency at which the filter gain is down by 6 decibels from the pass band and <corner> for the UNI filter is where the gain is down by 3 dB. The rolloff order represents the order of a Butterworth polynomial and the corresponding rollof rate is <n>*24 decibels/octave for the BI filter and <n>*12 dB/octave for the UNI filter. A report relating a Butterworth rolloff order to the boundaries within which a desired filter should fit is given in reference [12].

For a BI directional Butterworth filter, give
    BI,<corner>,<n>
where
    <corner> = corner frequency in Hertz. Default=0.1.
    <n> = rolloff order, an integer. Default=2.

For a UNI directional butterworth filter, give
    UNI,<corner>,<n>

For an FFT filter, give
    FFT,<corner>,<tband>
where
    <tband> = width of the transition band, in Hertz.
    Default=0.1.

For an ORMsby filter, give
    ORM,<corner>,<tband>,<ormk>
where
    <ormk> = a fraction of the rolloff ramp width in which to apply parabolic smoothing. Each end of the linear rolloff ramp is tapered with a parabola to avoid discontinuities where the ramp meets the pass band and the stop band.

Default value for <ormk> =0.1.

6.6 CORAVD still needs work:

- FFT and ORM filters need to be tested.
- The ORM filter would be faster for low cut filters if we provided an option for smoothing and decimating the baseline before it was filtered as was done in the PHASE2 program.
- Relative advantages and disadvantages of all the filters need to be evaluated.
- In order to avoid a wrap-around effect from the filters, the actual data is padded with trailing zeros before filtering. (Except when using the UNI filter which doesn't require padding.) We must consider how large an area should be padded with the BI and FFT filters. At present, the FFT filter pads the data enough to extend the number of samples out to the nearest power of 2. The BI filter pads with an area half the length of the actual data. Bill Joyner warns that both these pad lengths can be too short in some situations. Bill has some rules of thumb for choosing pad lengths that have not been incorporated into CORAVD yet.
- The zero padding will create a sharp step in a time series if the series does not begin and end at zero. This step may cause problems if the high-cut filter option is ever used. This same problem needs to be considered with respect to the high-cut filter in HIFRIC too. In Joe Fletcher's programs, he usually uses a tapering function before applying the BI or UNI filters. The function weights a small section of each end of the time series with a cosine taper to bring the value at the first and last point in the series to zero. Another option could provide a narrow section of the pad area (rather than in the actual data as with Joe's scheme) containing a taper from the last point of actual data down to zero. In either case, Bill Joyner suggests that a suitable width for such a taper would be 1.0/(corner frequency of the high-cut filter).

  These two options have been coded, but not tested. To try them, give DATATAPER or PADTAPER among the filter options.

CHAPTER 7

PHASE3, 4, 5, and 6 Programs

PHASE3

PHASE3 calculates response spectra and Fourier amplitude spectra. The Fourier spectrum is calculated (from its definition) at those frequencies used in the response spectrum calculations. The relative velocity response spectrum and the Fourier spectrum appear in the same plot with linear axes. The pseudo-velocity response spectrum is plotted with tripartite log-log axes.

Ask Pete Mork or Virg Perez for instructions.

PHASE3 needs work:
- The program assumes that the initial velocity is zero.
- Program should get run parameters from the command line.

PHASE4

PHASE4 calculates the Fourier spectrum by the Fast Fourier Transform. Results are plotted with linear axes and also with log-log axes.

Ask Pete Mork for instructions.

PHASE4 needs work:
- Roger Borchert, Bill Joyner and Chuck Mueller feel that before applying the FFT the time series data sould be zero-filled to the nearest 2**M points rather than reinterpolating the data to fill completely the 2**M points.
- They also feel that the spectrum should not be smoothed. (Allow alternative options?)
- Program should get its run parameters from the command line rather than by prompting the user.

PHASE5

PHASE5 plots the envelope of the velocity response as a function of time for various periods in the form of contour shaded areas indicating different amplitude levels. A second plot indicates the total duration for each level. See reference [14].

PHASE5 has not been installed at NSMDC yet, but Virg Perez plans to install it in the future and to describe the program in a separate report.

PHASE6

PHASE6 calculates the amplitudes of pseudo-velocity response that are sustained for various cycles and plots these amplitudes with tripartite log-log axes. See reference [13].

Virg Perez plans to describe the PHASE6 program in a separate, as yet unwritten report.

CHAPTER 8

Support Programs

8.1 Time Series Plotter, TSPLOT

TSPLOT plots time series data from DR100, SCALE, HIFRIC, or CORAVD. It cannot plot BUTTER output files yet.

There are four different versions: STSP plots on the teck-tubes or green screens, VTSP on the Versatec, CTSP on the Calcomp, and ITSP plots through the device-independent VIEWER software available at NSMDC.

To run TSPLOT, type the name of the version you wish, followed by a list of file names. The file names specify the data files you wish to have plotted. You may also include the name of a text file that contains a top-of-plot title.

Example:

```
[110,110]STSP [123,50]mydata.r01,.r02,.r03,topplot.ttl
```

The example above uses default scaling and labeling options. Each plot page will show 20 seconds of each curve (one curve for each file given on the command line). The curves are shown in separate strips across the page, each strip with its own y-axis, and the width of the strips (length of the y-axes) depending on the number of curves on the page. The first two significant digits, plus one, of the peak value of a curve are used for the scale on that curve's y-axis. The title at the top of the plot will come from the text file specified in the command line, or if there was no text file, the title will consist of the names of the data files.

To alter the size of the axes, you may include up to five numeric parameters after the file names on the command line. The meaning of these numbers depends on the order in which they are given, so give a null value (two consecutive commas) if you wish to use a default value among other parameters you wish to set. The numeric parameters and the order in which they must be given are: <tbegin>,<tend>,<spp>,<ysize>,<yspace>.

<tbegin> is the time at which the plot will begin.

Default=0.0.

<tend> is the time at which the plot will end. Default= ending time of the longest curve.

<spp> are the number of seconds to appear across each plot page. Default =20.0. If <tend> - <tbegin> is greater than <spp>, more than one page will be plotted.

<ysize> is the size of the y-axes, given as a fraction of the plot page.

<yspace> is the size of the space to be left between the plot strips, given as a fraction of the plot page.

To alter the vertical scale for any curve, you may include the maximum range for that curve's y-axis in parentheses after the file name. The y-axis will range from minus to plus the value in parenthesis. If you wish to use the peak value of the curve, include the letter "p" in the parentheses rather than a number.

To alter the way peaks are labeled, include the keyword "ARROW" or "NOPEAK" in the command line. ARROW will plot a little arrow that points to the peak, and NOPEAK will not put any label at all next to the peak.

To alter the other labels, include one of the following keywords in the command line: "SEB", "NOLABELS", or "AXESONLY". With SEB, the top of plot titles will appear in the format used in the data reports published by the Seismic Engineering Branch. (Talk with Pete Mork to learn more about this format.) NOLABELS and AXESONLY are intended for use with ITSP and Larry's VIEWER programs. With NOLABELS, no labels will be plotted, except perhaps those next to the peaks. AXESONLY is like NOLABELS except that the axes are plotted along with the curves.

More examples:

```
[110,110]STSP mydata.r01(500.),.r02(300.),.r03(500.)  -
             topplot.ttl,SEB, 3.0,10.0,7.0

[110,110]STSP mydata.r01(p),.r02(p),.r03(p),,,,0.18
```

To compare the shapes of curves plotted in several different TSPLOT runs, it is a good idea to provide a value for <ysize> so that all the curves to be compared use a y-axis of the same size. If <ysize> is not specified, TSPLOT will choose a value that depends on the number of lines in the top-of-plot title and on the number of curves to be plotted on the page. <Ysize> values of 0.7, 0.3, 0.2, and 0.14 work well for one, two, three, and four curves per plot page, respectively. To be more precise in choosing a value for <ysize>, divide up the vertical plotting space by the number of curves to be plotted and subtract a bit (0.04) for space between the curves. The vertical space left for plotting after labels and margins are allowed is (0.85 - 0.01*(3 + number of top-of-plot title lines)).

CTSP requires two pens in the Calcomp plotter. The first (right-most) pen is used for the curves and axis lines, the second pen is used for characters. Characters plotted with a felt tip pen make better ZEROX copies than those done with a ball point pen.

The Calcomp plotter won't automatically reposition a new frame to the right-hand side of the paper after three frames have been plotted across the paper. Hence, if you are plotting more than three frames, use ITSP and VIEWERC rather than CTSP.

ITSP and the VIEWER programs allow users to display a TSPLOT plot on a screen, then to choose whether to reproduce the same plot on a hard-copy device. ITSP and VIEWERs also allow users to combine several TSPLOT pages into a new display. ITSP is run just like the other versions of TSPLOT except that the user must respond to prompts asking for VIEWER operating modes. ITSP will generate a disk file named BATCH.PLT that VIEWER will need to access. After running ITSP, run one of the VIEWERs: lb0:[7,11]VIEWER for screen plots, lb0:[7,11]VIEWERV for Versatec plots, LB0:[7,11]VIEWERC for Calcomp plots (and there are other versions too). Once VIEWER begins to prompt you with "viewer>", type "help" to learn how to proceed. If you need more help with the VIEWERs, talk to Larry Baker, Pete Mork, or Chuck Mueller.

TSPLOT still needs work:
==========

- A bug in ITSP or VIEWER causes the Calcomp plotter to leave a blank frame between consecutive plot frames.
- The portion of the page to be left for margins could be specified at run time. Allow four more numeric run parameters to represent the left, right, top and bottom margins in terms of fractions of the plot page. All the code for this is in place except for command line interpretation.
- Need an option to specify the plot scale in centimeters rather than in fractions of the plot page so we can reproduce a trace in the same scale as the original (or an enlarged original). Most of the code for this is in place except for command line interpretation. Maybe if <ysize> is given greater than 1.0, TSPLOT should interpret the <ysize> value as the y-axis length in centimeters and interpret the <spp> value as the length in centimeters for each second along the x-axis.
- Chuck Mueller asks for more control over what does and does not get labeled with options liked NOLABELS and AXESONLY.

8.2 Data File Dumper, BBFILE

BBFILE will display the contents of all header blocks and selected data blocks in a blocked binary data file. A short description of what the value represents is shown alongside each standard header block value. A more complete description of the header block values is given in appendix B.

To use BBFILE, type --

```
     [110,110]BBFILE [blocked binary file]
 or  [110,110]BBFILE <text file>=[blocked binary file],<list>
```

Where:

[blocked binary file] is the name of the blocked binary file to be dumped.

<text file> is the name of the text file in which the blocked binary file's contents will be displayed. The display will appear at the user's terminal if <text file> is not specified.

<list> is an optional list of numbers of the data blocks to be displayed. Dashes may be used to indicate a range of numbers.

Example:

```
     [110,110]BBFILE mylist.tmp=DR2:[123,100]gooddata.r01,1-4,20
```

8.3 Header Block Changing Program, BWRITE

BWRITE allows a user to alter the contents of the header blocks in a blocked binary data file.

To use BWRITE, type --

```
[110,110]BWRITE
```

BWRITE, unlike the other AGRAM programs, will prompt the user for instructions.

BWRITE still needs work:

- Interaction between a user and the BWRITE program is slow and awkward. If a header changing program is going to be used very frequently, other ways of directing the program should be established. One method for providing new or altered header information to a header changing program would be to present an edited BBFILE dump file as input to the header changer (call it CHNGBF).

  Example --

```
[110,110]BBFILE mylist.tmp=DR2:[123,100]gooddata.r01
EDT MYLIST.TMP
     .
     :
*EXIT
[110,110]CHNGBF bestdata.r01=mylist.tmp, -
               DR2:[123,100]gooddata.r01
```

- Someday we should be able to fill in header information with something like this BWRITE program early in the process. Then any AGRAM program that found its run parameters in the header blocks in its input file would not need to get run parameters from the command line.

8.4 Data File Reformatter, REFORM

REFORM reformats data files back and forth between blocked binary data files and card-image text files. REFORM will eventually perform a variety of reformatting functions, but at present it can only perform three:

- reformat BUTTER's card-image output files to blocked binary files that can be input to SCALE
- reformat PHASE1 card-image output files to blocked binary HIFRIC input files
- reformat SCALE, HIFRIC and/or CORAVD blocked binary files to text files suitable for export to the EDIS archive in Colorado.

To use REFORM, type --

```
[110,110]REFORM <outfile>,<usermsg>=[infile] -
                                  [text file type]
```

Where:

<outfile> is the name of the reformatted output file to be created.

<usermsg> is the file to receive diagnostic messages. These messages go to the user s terminal if <usermsg> isn t given.

[infile] is the file that contains the data to be reformatted. There may be several [infile]s when [text file type] = "EDIS". See Pete Mork to learn EDIS requirements.

[text file type] ="BUTTER", "BKY-BUTTER","PHASE1",or "EDIS" to indicate the type of text file to be read or created.

Example --

```
[110,110]REFORM forSCALE.tmp=  -
                DR2:[122,201]elcdamain.but,butter
```

REFORM still needs work:

- Need to reformat blocked binary data files to a compact text file, complete with summary, for use in disseminating the data to Willy Lee's archive system at SLAC similarly to the way it's done for EDIS.
- Need routines to reformat EDIS or Willy Lee archive data back into blocked binary files.
- Need to reformat data from Willy Lee's digitizer downstairs to SCALE input files. Jack Boatwright has a program that will reformat the data to look like BUTTER's output files.

APPENDIX A
Examples

Sample plots and run messages from SCALE, HIFRIC, CORAVD, PHASE3, PHASE4, TSPLOT and BBFILE programs are given in this appendix. The examples process the first component of the 1979 Imperial Valley main shock data recorded at the EDA station (the 70-mm film recorder located near the El Centro Differential Array.) The commands used to generate the examples are listed below. They are followed by a copy of the plots and most of the run messages generated by the commands.

Results from PHASE3 and PHASE4 are included in the examples even though instructions for running these programs are not given in this report. These two programs will be reorganized in the near future and will be discussed in more detail in the next version of the report.

The commands are arranged to be run in batch mode on the PDP 11/70. Batch mode comands on this computer are constructed just like interactive mode commands except that a dollar sign ($) is added as the first character. Comment lines have a dollar sign followed by an exclamation mark ($!) as the first two characters. Comments in this deck indicate the purpose of the commands that follow the comments.

The command deck shows VTSP, the Versatec version of TSPLOT, as the plotting program. VTSP is a good plotting program for batch mode use because it does not require any user interaction, but plots from the Versatec equipment at NSMDC are not as good quality as those from the Calcomp. Better quality plots could be drawn with the Calcomp plotter using CTSP or ITSP and the device-independent VIEWER software. ITSP, VIEWER, and the NSMDC Calcomp equipment require user interaction, however, and they are unique to the NSMDC site, so they aren't used as the plotting method in this appendix.

Sample Command Deck
==========

```
$JOB/TIME:400
$!
$!     DR2:[110,50]APPA.BAT
$!     Illustrate the AGRAM programs for Appendix A of the
$!     User's Guide.
$!     Submit this deck to run in the evening with:
$!     >SUBMIT/AFTER:20:00 =APPA.BAT
$!
$ON WARNING THEN GOTO END
$CWD DR2:[123,5]
$!
$!
$!   Reformat butter output.  (This step won't be necessary once
$!   the BUTTER program is reorganized.)
$!
$!
$[110,110]REFORM temp.ref=dk3:[122,201]elcdamain.but,butter
$!
$!
$!   Run SCALE to get scaled, uncorrected data in three files
$!   (temp.r01, temp.r02, and temp.r03) corresponding to the
$!   three accelerometer traces on the original record.
$!
$!   Note that the first time coordinate in each trace is given
$!   the value of zero.  Since the three traces do not have
$!   precisely the same horizontal coordinate in their first
$!   digitized sample, the three traces are not completely
$!   synchronous.
$!
$!   Plot results for the first trace.
$!
$!
$[110,110]SCALE temp.=temp.ref,0.5,1.92,1.85,1.77
$[110,110]VTSP temp.r01,,,20.0,0.3
$!
$!
$!   Run HIFRIC for the first trace to get interpolated,
$!   high-cut filtered and instrument-corrected data.
$!   Plot results.
$!
$!
$[110,110]HIFRIC temp.c01=temp.r01,.040,.570
$[110,110]VTSP temp.c01,,,,0.3
$!
$!
$!   Compare the effects of the interpolation, instrument
$!   correction and high-cut filter applied in HIFRIC
$!   to uninterpolated, uncorrected data produced by SCALE.
$!   Also compare these to interpolated but uncorrected data.
$!   To do so, rerun HIFRIC with interpolation only, no
```

```
$!   instrument correction or filter, then plot two seconds of
$!     (1) uncorrected (SCALE) data,
$!     (2) interplolated (HIFRIC with INTERP) data, and
$!     (3) filtered, instrument corrected (standard HIFRIC) data
$!   with an exaggerated scale to illustrate how the  peak value
$!   was affected by processing.
$!
$!
$[110,110]HIFRIC temp.i01=temp.r01,INTERP
$[110,110]VTSP temp.r01,.i01,.c01, 5.0,7.0,2.0
$!
$!
$!   Notice the little indentations introduced by the standard
$!   HIFRIC process to the trough and peak at about 5.7 seconds.
$!   The indentations illustrate the effect of approximating
$!   derivatives about a very sharp peak during instrument
$!   correction.  The same difficulty, though more pronounced,
$!   occurred with the older centered-difference differentiation
$!   method.
$!
$!   The BBFILE program can be used to examine the contents of
$!   the data files near these two sharp peaks.  Data for 5.7
$!   seconds is in block 9 of the HIFRIC output file and, if
$!   approximately 600 samples per centimeter were digitized,
$!   data for 5.7 seconds would be in or near block 54 of the
$!   SCALE output file.
$!   (block 9 is determined from 8.9 = 5.7 seconds * 200 samples
$!    per second / 128 samples per block.)
$!   (block 54 is determined from 53.4 = 5.7 seconds * 600
$!    samples per second * 2 items (x and y) per sample
$!    / 128 items per block.)
$!
$!
$[110,110]BBFILE temp.c01,9
$[110,110]BBFILE temp.r01,52-54
$!
$!
$!   Notice that the frequency-domain instrument correcting
$!   algorithm has nearly the same effect as the time-domain
$!   algorithm.
$!
$!
$[110,110]HIFRIC temp2.c01=temp.r01,.040,.570,FDIC
$[110,110]VTSP temp.c01,temp2.c01, 5.0,7.0,2.0
$!
$!
$!   Run CORAVD with linear baseline correction to the velocity
$!   and without long-period filtering.  Plot results.
$!
$!
$[110,110]CORAVD temp.a01=temp.c01
$[110,110]VTSP temp.a01,temp.v01,temp.d01
$!
```

```
$!
$!   Run CORAVD with long-period filtering but no linear
$!   baseline correction and plot results.
$!
$!
$[110,110]SLOWAVD temp2.a01=temp.c01,0,0,BI,.17,2
$[110,110]VTSP temp2.a01,.v01,.d01
$!
$!
$!   Run PHASE3 to calculate relative response spectra, then
$!   plot results.  Let PHASE3 and 4 process the unfiltered
$!   version of acceleration since there is little long-period
$!   noise in this data.
$!
$!
$[110,110]bwrite
temp.a01
q

$PIP temp.a02/nv=temp.a01
$PIP temp.a03/nv=temp.a01
$PIP temp.tit/nv=temptit.sav
$[110,110]PHASE3
temp
temp.ph3
$[110,110]PH3PLT
temp.ph3
$!
$!
$!   Run PHASE4 to calculate FFT spectrum, then
$!   plot results.
$!
$!
$[110,110]PHASE4
temp
temp.ph4
2
1
temp.plt
$pip /nv=1b0:[7,11]VIEWERV.TSK
$VIEWERV
Plot temp.plt
quit
$pip VIEWERV.TSK;*/de
$!
$!
$!   How many temporary files are left?  They should either
$!   be renamed or deleted.
$!
$!
$PIP temp.*,temp2.*/li
$PIP temp.*;*,temp2.*;*/de
$!
```

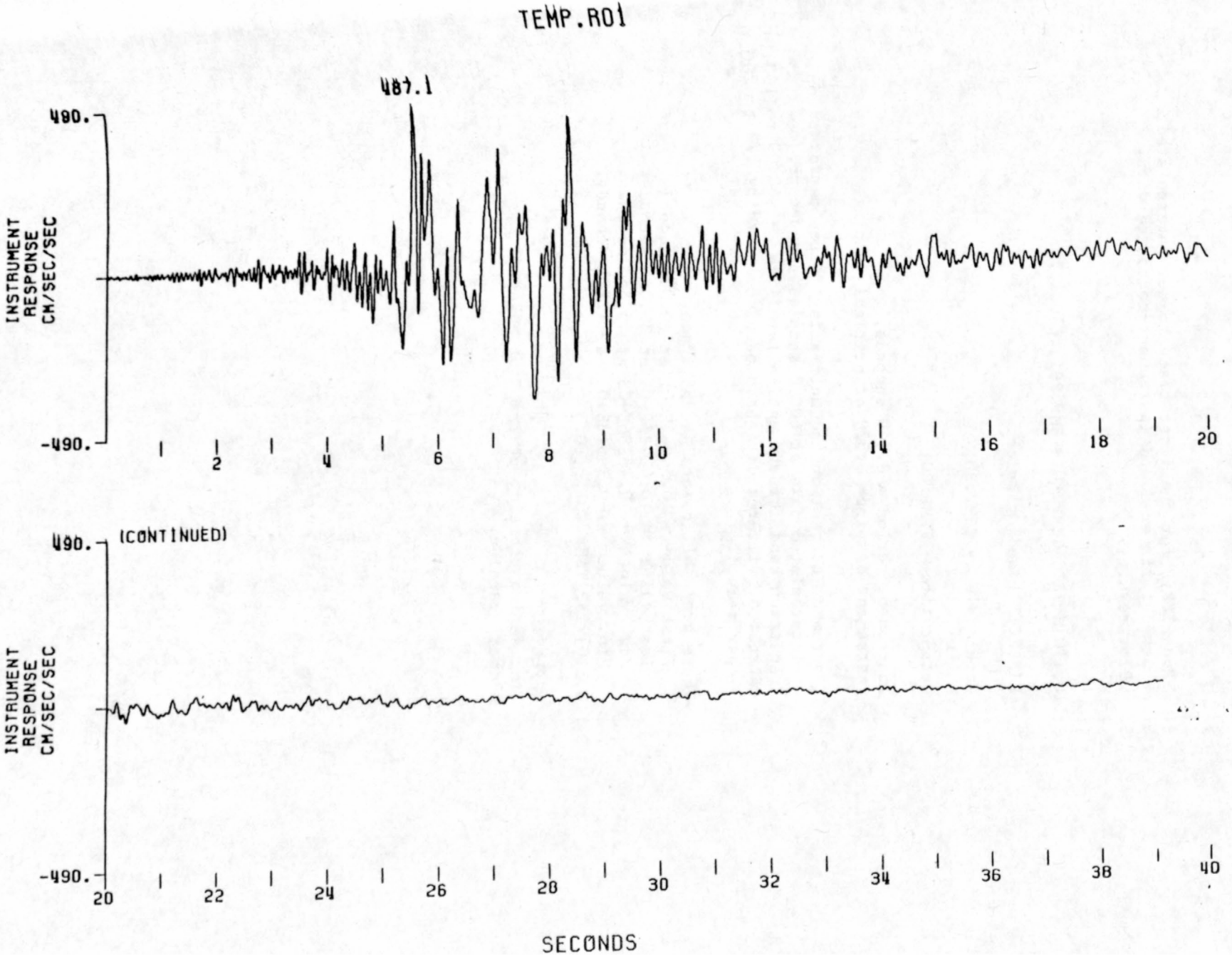
```
$!
$!   end of job.
$!
$END: CONTINUE
$EOJ
```

Output From the Sample Commands

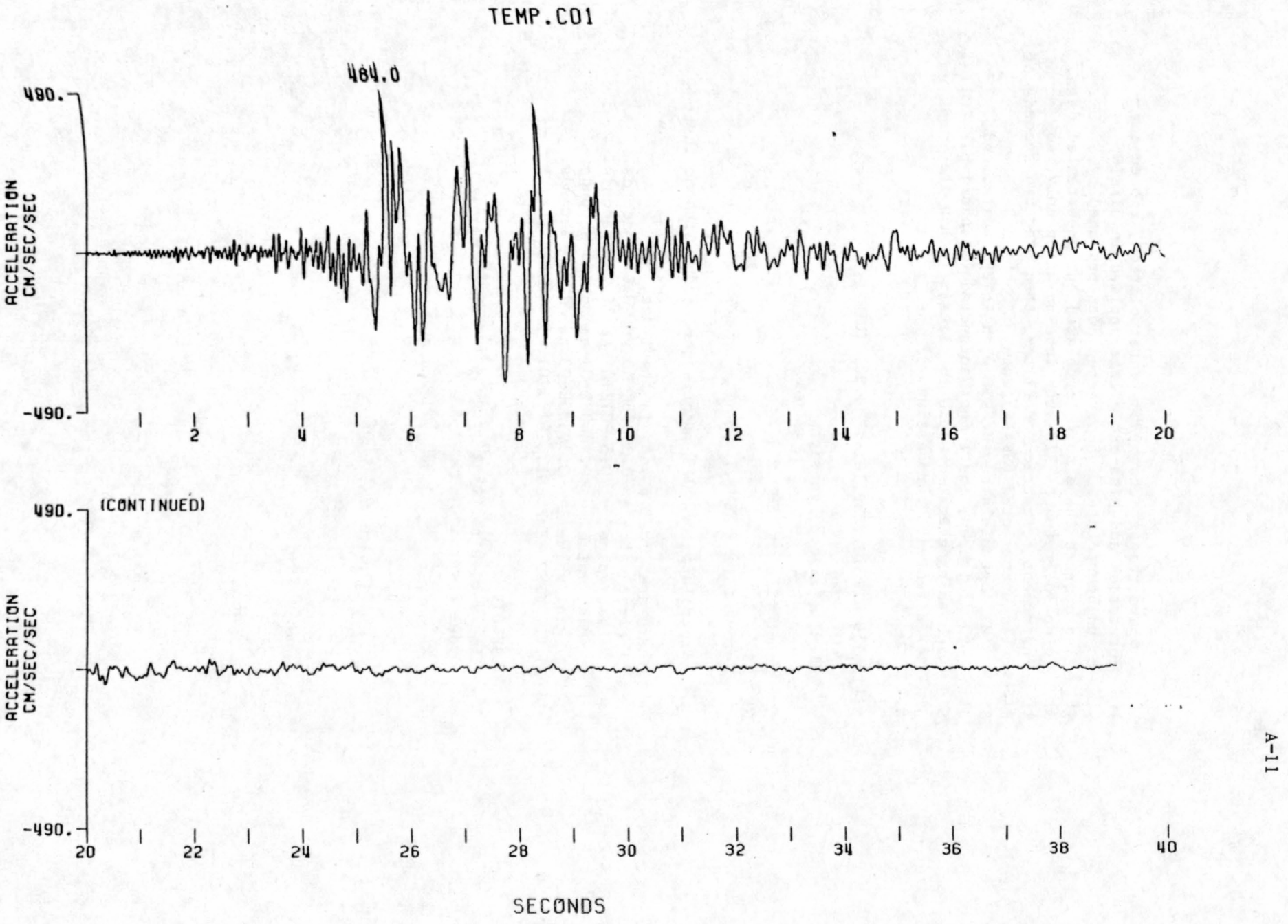
```
$JOB/TIME:400
   ====================================
   User Job -              Terminal VT1:
              UIC = [123,1]
   ====================================

RSX-11M-PLUS V02 BL10   [1,54] System     GSRSX0
$!
$!      DR2:[110,50]APPA.BAT
$!      Illustrate the AGRAM programs for Appendix A of the
$!      User's Guide.
$!      Submit this deck to run in the evening with:
$!      >SUBMIT/AFTER:20:00 =APPA.BAT
$!
$ON WARNING THEN GOTO END
$CWD DR2:[123,5]
DR2:  is  95.7% full (21625. blocks free)
$!
$!
$!   Reformat butter output.  (This step won't be necessary once
$!   the BUTTER program is reorganized.)
$!
$!
$[110,110]REFORM TEMP.REF=DK3:[122,201]ELCDAMAIN.BUT,BUTTER
    The first 3 lines of the butter file are --
    Final BUTTER data traces
    El Centro DA, Main Shock: Tape IV913, frames 32-  Time trace

    Number of time marks=     76
    Component 1
      23458 samples in the data trace, and
       4474 samples in the reference trace.
    Component 2
      22645 samples in the data trace, and
       4474 samples in the reference trace.
    Component 3
      23301 samples in the data trace, and
       4474 samples in the reference trace.
```

```
$!
$!
$!   Run SCALE to get scaled, uncorrected data in three files
$!   (temp.r01, temp.r02, and temp.r03) corresponding to the
$!   three accelerometer traces on the original record.
$!
$!   Note that the first time coordinate in each trace is given
$!   the value of zero.  Since the three traces do not have
$!   precisely the same horizontal coordinate in their first
$!   digitized sample, the three traces are not completely
$!   synchronous.
$!
$!   Plot results for the first trace.
$!
$!
$[110,110]SCALE TEMP.=TEMP.REF,0.5,1.92,1.85,1.77

    SCALE, 11jan83 version.

    Input file = TEMP.REF
       digitization scale = 10000. units/cm.
       time-tick interval = 0.50 seconds.

    Output file = TEMP.R01, recorder sensitivity = 1.9200 cm/g.
       There are 23458 points in the trace.
       Trace length = 39.104 seconds.
       Mean value =  0.37255E+00 cm/sec**2.
       Max. value =  0.48747E+03 at  5.557 seconds.
       Min. value = -0.39278E+03 at  7.755 seconds.
       There are  15 back-stepping points, the largest of which is for
          0.00040 seconds at time =  5.586 seconds.
       The largest difference between two consecutive time-tick
           intervals was 1.216 % at tick number 2.
       The first data point (at 0.000 sec.) occurred before
          the first time tick (at  0.011 sec.).
       The last data point (at 39.104 sec.) occurred after
          the last time tick (at 37.011 sec.).
       The last point in the reference trace (at 39.100 sec.)
          occurred before the last point in the data trace (at 39.104 sec.),
          so the reference line was extended with the slope (= 0.10870E-01)
          of the last two points given in the reference trace.

    Output file = TEMP.R02, recorder sensitivity = 1.8500 cm/g.
DEBUG,  backstep one reference point:
   x,rx1,rx2,ir,irb=   52766.00       52770.00       52776.00        918       1
       There are 22645 points in the trace.
       Trace length = 39.085 seconds.
       Mean value = -0.16772E+02 cm/sec**2.
       Max. value =  0.89331E+03 at  3.564 seconds.
       Min. value = -0.72426E+03 at  3.537 seconds.
       There are 154 back-stepping points, the largest of which is for
          0.00169 seconds at time =  2.541 seconds.
       The largest difference between two consecutive time-tick
```

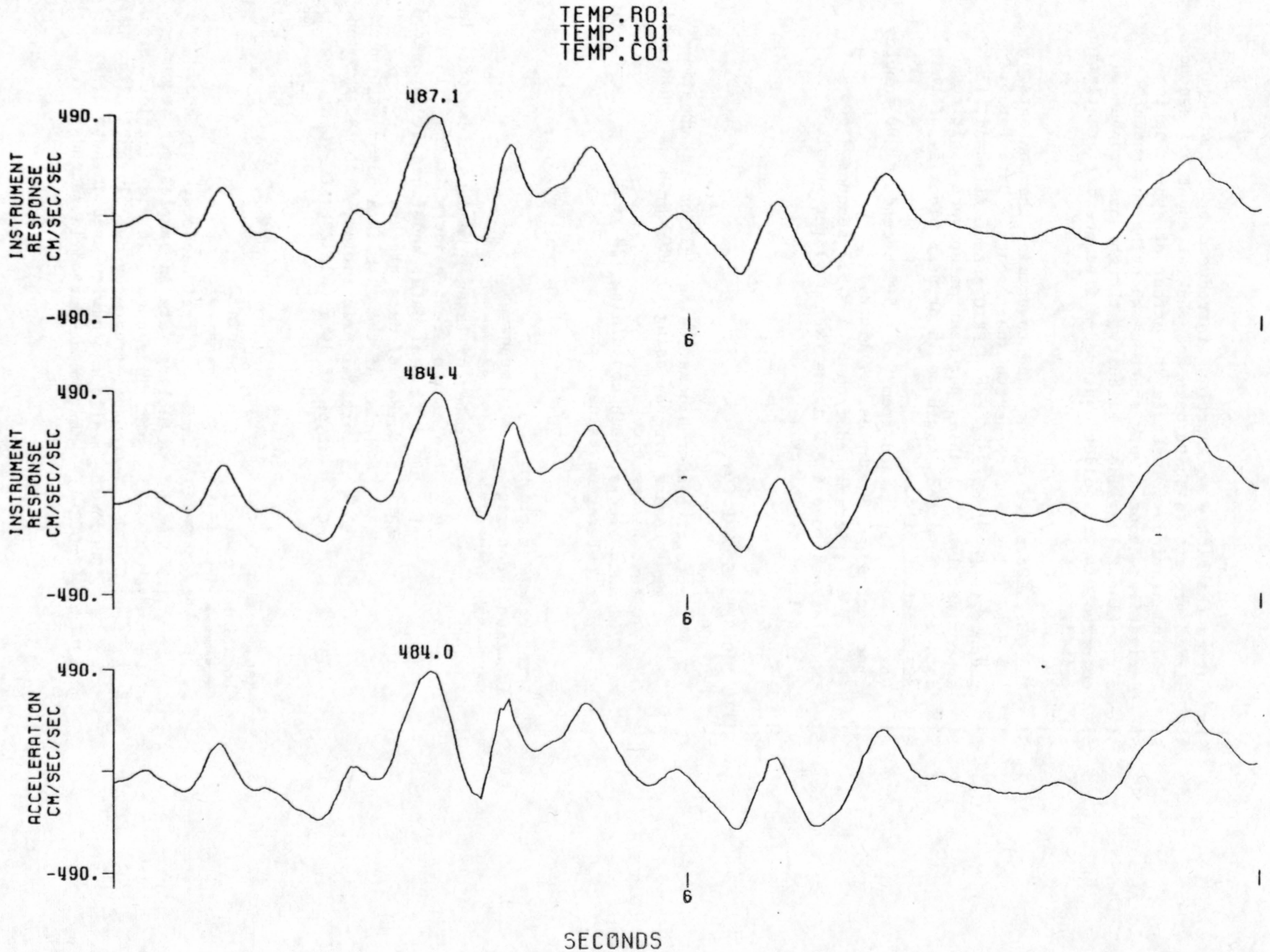
```
          intervals was 1.216 % at tick number 2.
        The last data point (at 39.085 sec.) occurred after
           the last time tick (at 37.000 sec.).

     Output file = TEMP.R03, recorder sensitivity = 1.7700 cm/g.
        There are 23301 points in the trace.
        Trace length = 39.113 seconds.
        Mean value =  0.80066E+00 cm/sec**2.
        Max. value =  0.35460E+03 at  5.648 seconds.
        Min. value = -0.33434E+03 at  9.036 seconds.
        There are  22 back-stepping points, the largest of which is for
           0.00130 seconds at time =  9.012 seconds.
        The largest difference between two consecutive time-tick
            intervals was 1.216 % at tick number 2.
        The last data point (at 39.113 sec.) occurred after
           the last time tick (at 36.991 sec.).
        The last point in the reference trace (at 39.081 sec.)
           occurred before the last point in the data trace (at 39.113 sec.)
           so the reference line was extended with the slope (= 0.10870E-01)
           of the last two points given in the reference trace.
$[110,110]VTSP TEMP.R01,,,20.0,0.3

     TSPLOT.
     Time series plotting program for AGRAM data,
     09dec82 version.
```


TEMP.R01
487.1
490.
-490.
INSTRUMENT RESPONSE CM/SEC/SEC
2
4
6
8
10
12
14
16
18
20
(CONTINUED)
490.
-490.
INSTRUMENT RESPONSE CM/SEC/SEC
20
22
24
26
28
30
32
34
36
38
40
SECONDS

```
$!
$!
$!   Run HIFRIC for the first trace to get interpolated,
$!   high-cut filtered and instrument-corrected data.
$!   Plot results.
$!
$!
$[110,110]HIFRIC TEMP.C01=TEMP.R01,.040,.570

    HIFRIC, 11jan83 version.

    Input file  = TEMP.R01

    Output file = TEMP.C01

    Instrument period = 0.040 seconds,
    instrument damping= 0.570 of critical damping.

    Instrument correction and anti-alias filter performed on densely
       interpolated data at 600.0 samples per second.
    Transition band for the anti-alias filter =  50.00 to 100.00 hz.
    Corrected, filtered data have been decimated by 1/3 to 200.0 samples
       per second.

    There are  7811 points in the trace.
       First value =  0.54643E+01 at  0.000 seconds.
       Last value  =  0.46125E+01 at 39.050 seconds.
       Max. value  =  0.48396E+03 at  5.550 seconds.
       Min. value  = -0.39513E+03 at  7.745 seconds.
$[110,110]VTSP TEMP.C01,,,,0.3

    TSPLOT.
    Time series plotting program for AGRAM data,
    09dec82 version.
```


TEMP.C01
484.0
480.
-490.
ACCELERATION
CM/SEC/SEC
2
4
6
8
10
12
14
16
18
20
(CONTINUED)
490.
-490.
ACCELERATION
CM/SEC/SEC
20
22
24
26
28
30
32
34
36
38
40
SECONDS

```
$!
$!
$!   Compare the effects of the interpolation, instrument
$!   correction and high-cut filter applied in HIFRIC
$!   to uninterpolated, uncorrected data produced by SCALE.
$!   Also compare these to interpolated but uncorrected data.
$!   To do so, rerun HIFRIC with interpolation only, no
$!   instrument correction or filter, then plot two seconds of
$!     (1) uncorrected (SCALE) data,
$!     (2) interplolated (HIFRIC with INTERP) data, and
$!     (3) filtered, instrument corrected (standard HIFRIC) data
$!   with an exaggerated scale to illustrate how the  peak value
$!   was affected by processing.
$!
$!
$[110,110]HIFRIC TEMP.I01=TEMP.R01,INTERP

    HIFRIC, 11jan83 version.

    Input file  = TEMP.R01

    Output file = TEMP.I01

    Interpolate to 200.0 samples per second without instrument correction

    There are  7821 points in the trace.
       First value = -0.35906E+00 at  0.000 seconds.
       Last value  =  0.51901E+01 at 39.100 seconds.
       Max. value  =  0.48438E+03 at  5.555 seconds.
       Min. value  = -0.39305E+03 at  7.755 seconds.
$[110,110]VTSP TEMP.R01,.I01,.C01, 5.0,7.0,2.0

    TSPLOT.
    Time series plotting program for AGRAM data,
    09dec82 version.
```


TEMP.RO1
TEMP.IO1
TEMP.CO1
487.1
490.
INSTRUMENT RESPONSE CM/SEC/SEC
-490.
6
484.4
490.
INSTRUMENT RESPONSE CM/SEC/SEC
-490.
6
484.0
490.
ACCELERATION CM/SEC/SEC
-490.
6
SECONDS

```
$!
$!
$!   Notice the little indentations introduced by the standard
$!   HIFRIC process to the trough and peak at about 5.7 seconds.
$!   The indentations illustrate the effect of approximating
$!   derivatives about a very sharp peak during instrument
$!   correction.  The same difficulty, though more pronounced,
$!   occurred with the older centered-difference differentiation
$!   method.
$!
$!   The BBFILE program can be used to examine the contents of
$!   the data files near these two sharp peaks.  Data for 5.7
$!   seconds is in block 9 of the HIFRIC output file and, if
$!   approximately 600 samples per centimeter were digitized,
$!   data for 5.7 seconds would be in or near block 54 of the
$!   SCALE output file.
$!   (block 9 is determined from 8.9 = 5.7 seconds * 200 samples
$!    per second / 128 samples per block.)
$!   (block 54 is determined from 53.4 = 5.7 seconds * 600
$!    samples per second * 2 items (x and y) per sample
$!    / 128 items per block.)
$!
$!
$[110,110]BBFILE TEMP.C01,9
    TEMP.C01 has
           1 integer header blocks, using  -32768 as "undefined",
           1 real header blocks, using -0.30000E-38 as "undefined",
           0 text header blocks,
    and   62 data blocks, each containing 128 real
             acceleration values.


    Integer header block:
    location   content          meaning
    ========   =======          =======
       1           0       number of integer headers, less one, in the file
       2           0       number of text headers in the file
       4           1       =1 if real data, undefined if integer data
      31          62       number of data blocks.
      32           3       index of the last data point within the last block
      41          90       vertical orientation, 0 to 180 from up.
      43          AC       type of data (BU, IR, AC, VL or DP)


    Real header block:
    location   content          meaning
    ========   =======          =======
       1    0.00000E+00   number of real headers, less one.
       5    0.20000E+03   sampling rate in samples per second
      46    0.10000E+05   digitizer output in digitization units/cm.
      47    0.50000E+02   corner frequency used in anti alias filter
      48    0.50000E+02   rolloff bandwith used in anti alias filter
      49    0.40000E-01   instrument period in seconds
```

```
  50     0.57000E+00   instrument damping
  51     0.19200E+01   SMA recorder sensitivity or DR100 coil constant
  90     0.00000E+00   time of the first data point
  91     0.54643E+01   value of the first data point
  92     0.39050E+02   time of the last data point
  93     0.46125E+01   value of the last data point
  94     0.55500E+01   time of the max. value
  95     0.48396E+03   max. value
  96     0.77450E+01   time of the min. value
  97    -0.39513E+03   min. value
  98     0.37255E+00   data offset for SCALE data
  99     0.00000E+00   begin time for CORAVD llsqf baseline correction
 100     0.39050E+02   end   time for CORAVD llsqf baseline correction
 101    -0.42748E-01   slope of CORAVD linear baseline correction
 102     0.22372E+01   intercept of CORAVD linear baseline correction
 103     0.21000E+01   source of data: 1.0,2.1,2.2,3.0,3.1, or 3.2


Data block  9:
  -0.96203E+02  -0.94562E+02  -0.85350E+02  -0.68872E+02  -0.48027E+02
  -0.27848E+02  -0.42156E+01   0.32427E+02   0.66180E+02   0.92291E+02
   0.10853E+03   0.12237E+03   0.13416E+03   0.12770E+03   0.10711E+03
   0.79117E+02   0.56060E+02   0.35781E+02   0.87723E+01  -0.17698E+02
  -0.41793E+02  -0.61205E+02  -0.73483E+02  -0.85145E+02  -0.90141E+02
  -0.89273E+02  -0.88101E+02  -0.83104E+02  -0.80112E+02  -0.84695E+02
  -0.90655E+02  -0.97345E+02  -0.10356E+03  -0.10954E+03  -0.12044E+03
  -0.13097E+03  -0.14335E+03  -0.15830E+03  -0.17365E+03  -0.18247E+03
  -0.18521E+03  -0.19001E+03  -0.19756E+03  -0.20646E+03  -0.21461E+03
  -0.22534E+03  -0.23270E+03  -0.23276E+03  -0.22679E+03  -0.21462E+03
  -0.20264E+03  -0.18098E+03  -0.14725E+03  -0.12443E+03  -0.88164E+02
  -0.44988E+02  -0.28581E+02   0.30271E+01   0.22837E+02   0.17512E+02
   0.23767E+02   0.13457E+02  -0.13829E+00  -0.42880E+01  -0.22559E+02
  -0.36343E+02  -0.37780E+02  -0.38345E+02  -0.36723E+02  -0.28956E+02
  -0.12704E+02   0.11962E+02   0.45791E+02   0.82265E+02   0.13640E+03
   0.18611E+03   0.22173E+03   0.26299E+03   0.29618E+03   0.32686E+03
   0.36071E+03   0.40300E+03   0.43919E+03   0.45180E+03   0.46394E+03
   0.48091E+03   0.48396E+03   0.47492E+03   0.46833E+03   0.43456E+03
   0.37481E+03   0.32707E+03   0.28942E+03   0.24291E+03   0.16662E+03
   0.97642E+02   0.48270E+02   0.55141E+01  -0.32778E+02  -0.66154E+02
  -0.10105E+03  -0.11070E+03  -0.11255E+03  -0.12952E+03  -0.72724E+02
  -0.19980E+02   0.12801E+02   0.10536E+03   0.16442E+03   0.23808E+03
   0.30642E+03   0.29437E+03   0.32673E+03   0.34687E+03   0.28592E+03
   0.23962E+03   0.21625E+03   0.18340E+03   0.15067E+03   0.12761E+03
   0.10970E+03   0.96466E+02   0.95348E+02   0.96465E+02   0.10026E+03
   0.11349E+03   0.12415E+03   0.13168E+03
```

second noted — peak — first noted

```
$[110,110]BBFILE TEMP.R01,52-54
    TEMP.R01 has
          1 integer header blocks, using  -32768 as "undefined",
          1 real header blocks, using -0.30000E-38 as "undefined",
          0 text header blocks,
    and  367 data blocks, each containing 128 real
             uninterpolated instrument response values.
```

Integer header block:

location	content	meaning
1	0	number of integer headers, less one, in the file
2	0	number of text headers in the file
4	1	=1 if real data, undefined if integer data
31	367	number of data blocks.
32	68	index of the last data point within the last block
41	90	vertical orientation, 0 to 180 from up.
43	IR	type of data (BU, IR, AC, VL or DP)

Real header block:

location	content	meaning
1	0.00000E+00	number of real headers, less one.
46	0.10000E+05	digitizer output in digitization units/cm.
51	0.19200E+01	SMA recorder sensitivity or DR100 coil constant
90	0.00000E+00	time of the first data point
91	0.13492E-01	value of the first data point
92	0.39104E+02	time of the last data point
93	0.54418E+01	value of the last data point
94	0.55574E+01	time of the max. value
95	0.48747E+03	max. value
96	0.77555E+01	time of the min. value
97	-0.39278E+03	min. value
98	0.37255E+00	data offset for SCALE data
103	0.10000E+01	source of data: 1.0,2.1,2.2,3.0,3.1, or 3.2

Data block 52:

0.54770E+01	-0.15782E+02	0.54782E+01	-0.10641E+02	0.54800E+01
-0.44898E+01	0.54812E+01	0.25126E+01	0.54823E+01	0.10077E+02
0.54837E+01	0.18042E+02	0.54848E+01	0.27040E+02	0.54866E+01
0.35348E+02	0.54882E+01	0.45093E+02	0.54891E+01	0.55323E+02
0.54904E+01	0.65998E+02	0.54917E+01	0.77184E+02	0.54930E+01
0.88727E+02	0.54948E+01	0.10020E+03	0.54956E+01	0.11319E+03
0.54970E+01	0.12545E+03	0.54983E+01	0.13913E+03	0.54988E+01
0.15315E+03	0.55006E+01	0.16713E+03	0.55033E+01	0.18068E+03
0.55039E+01	0.19511E+03	0.55050E+01	0.20951E+03	0.55074E+01
0.22377E+03	0.55090E+01	0.23858E+03	0.55107E+01	0.25308E+03
0.55125E+01	0.26815E+03	0.55151E+01	0.28230E+03	0.55167E+01
0.29768E+03	0.55192E+01	0.31129E+03	0.55213E+01	0.32646E+03
0.55238E+01	0.33971E+03	0.55259E+01	0.35510E+03	0.55280E+01
0.36870E+03	0.55300E+01	0.38430E+03	0.55321E+01	0.39775E+03
0.55341E+01	0.41345E+03	0.55363E+01	0.42657E+03	0.55387E+01
0.44182E+03	0.55421E+01	0.45373E+03	0.55457E+01	0.46769E+03
0.55505E+01	0.47979E+03	0.55574E+01	0.48747E+03	0.55678E+01
0.47070E+03	0.55687E+01	0.45959E+03	0.55695E+01	0.45797E+03
0.55705E+01	0.44369E+03	0.55717E+01	0.44195E+03	0.55730E+01
0.42810E+03	0.55737E+01	0.42502E+03	0.55753E+01	0.41239E+03
0.55754E+01	0.40738E+03	0.55771E+01	0.39587E+03	0.55779E+01

0.39014E+03	0.55779E+01	0.37737E+03	0.55788E+01	0.37103E+03
0.55802E+01	0.35881E+03	0.55813E+01	0.35241E+03	0.55825E+01
0.34039E+03	0.55831E+01	0.33308E+03	0.55845E+01	0.32183E+03
0.55845E+01	0.31228E+03	0.55863E+01	0.30205E+03	0.55859E+01
0.29056E+03	0.55884E+01	0.28220E+03		

Data block 53:

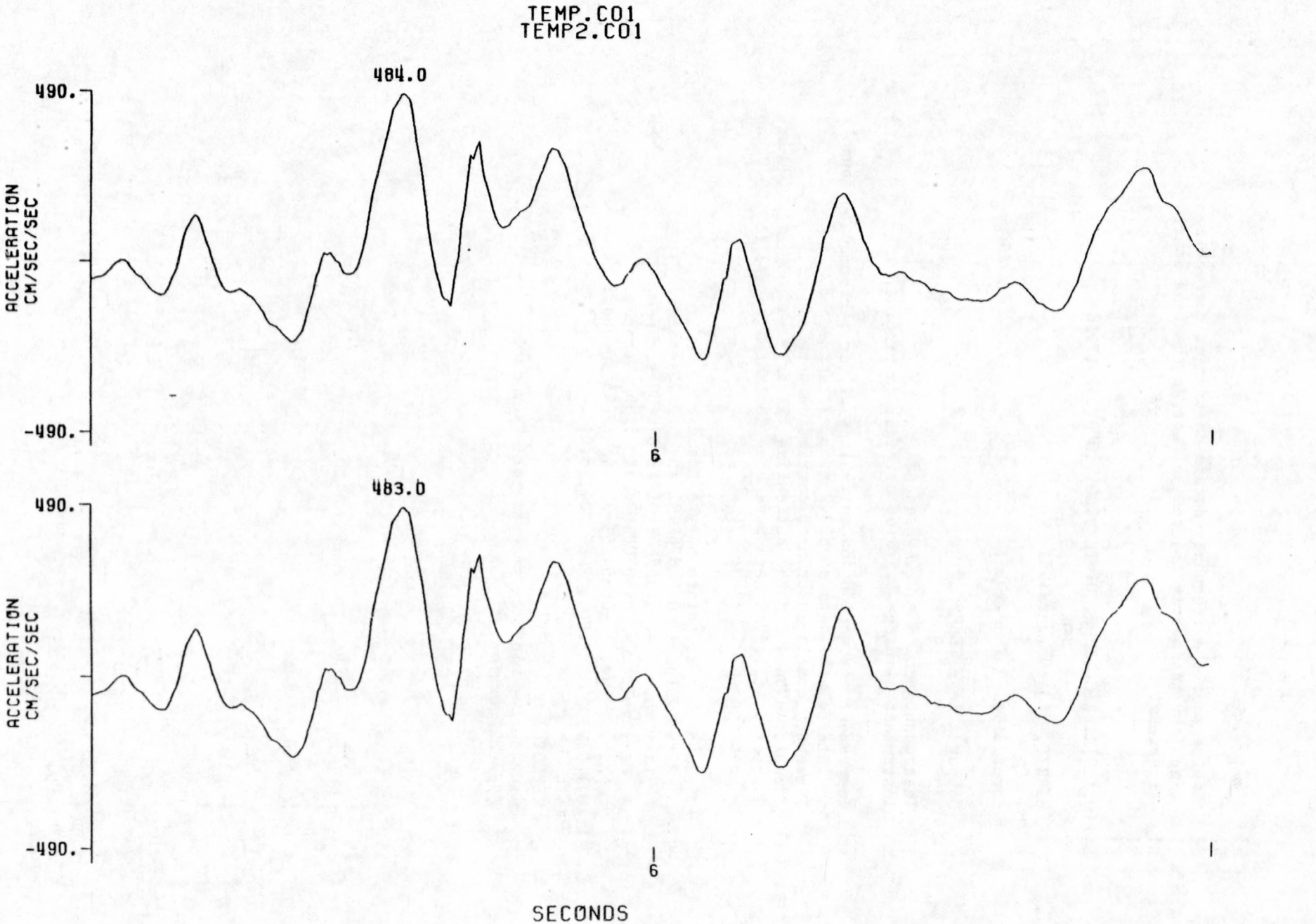
0.55897E+01	0.26760E+03	0.55906E+01	0.25902E+03	0.55916E+01
0.24489E+03	0.55924E+01	0.23611E+03	0.55936E+01	0.21743E+03
0.55946E+01	0.20855E+03	0.55953E+01	0.19017E+03	0.55976E+01
0.18228E+03	0.55981E+01	0.16385E+03	0.55987E+01	0.15170E+03
0.55997E+01	0.13444E+03	0.56015E+01	0.12306E+03	0.56020E+01
0.10575E+03	0.56038E+01	0.95186E+02	0.56047E+01	0.77978E+02
0.56058E+01	0.65726E+02	0.56073E+01	0.49441E+02	0.56080E+01
0.37109E+02	0.56101E+01	0.21858E+02	0.56118E+01	0.10249E+02
0.56130E+01	-0.46102E+01	0.56141E+01	-0.15890E+02	0.56162E+01
-0.29404E+02	0.56173E+01	-0.40464E+02	0.56191E+01	-0.52906E+02
0.56214E+01	-0.62495E+02	0.56229E+01	-0.74217E+02	0.56243E+01
-0.83332E+02	0.56262E+01	-0.93941E+02	0.56276E+01	-0.10271E+03
0.56306E+01	-0.11066E+03	0.56355E+01	-0.11825E+03	0.56406E+01
-0.12654E+03	0.56439E+01	-0.11360E+03	0.56446E+01	-0.99245E+02
0.56453E+01	-0.97154E+02	0.56480E+01	-0.83532E+02	0.56480E+01
-0.78680E+02	0.56501E+01	-0.65003E+02	0.56503E+01	-0.57700E+02
0.56517E+01	-0.42947E+02	0.56529E+01	-0.34679E+02	0.56538E+01
-0.18644E+02	0.56553E+01	-0.99146E+01	0.56570E+01	0.67316E+01
0.56574E+01	0.18172E+02	0.56587E+01	0.35738E+02	0.56597E+01
0.48351E+02	0.56604E+01	0.67145E+02	0.56608E+01	0.81037E+02
0.56631E+01	0.99623E+02	0.56640E+01	0.11489E+03	0.56655E+01
0.13353E+03	0.56670E+01	0.15769E+03	0.56683E+01	0.17091E+03
0.56691E+01	0.18879E+03	0.56714E+01	0.22591E+03	0.56753E+01
0.27739E+03	0.56811E+01	0.31644E+03	0.56889E+01	0.34691E+03
0.56965E+01	0.32513E+03	0.56966E+01	0.31196E+03	0.56963E+01
0.30884E+03	0.56982E+01	0.29729E+03		

Data block 54:

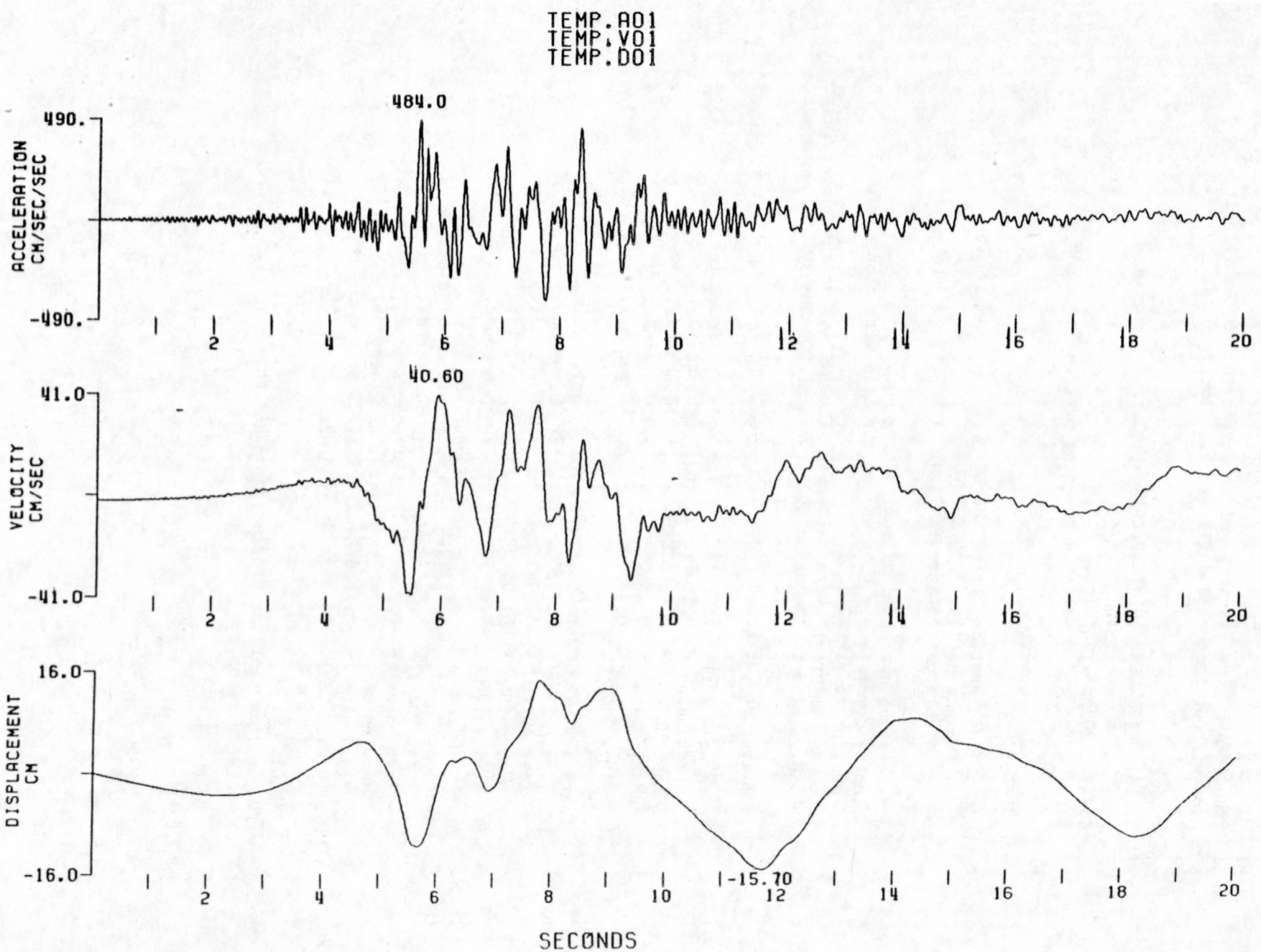
0.56986E+01	0.29234E+03	0.57007E+01	0.28288E+03	0.57008E+01
0.27573E+03	0.57021E+01	0.26664E+03	0.57026E+01	0.25846E+03
0.57044E+01	0.25044E+03	0.57046E+01	0.24094E+03	0.57057E+01
0.23271E+03	0.57067E+01	0.22300E+03	0.57080E+01	0.21488E+03
0.57094E+01	0.20574E+03	0.57104E+01	0.19738E+03	0.57115E+01
0.18825E+03	0.57121E+01	0.17916E+03	0.57137E+01	0.17065E+03
0.57154E+01	0.16265E+03	0.57165E+01	0.15383E+03	0.57175E+01
0.14577E+03	0.57195E+01	0.13833E+03	0.57207E+01	0.13023E+03
0.57224E+01	0.12366E+03	0.57242E+01	0.11673E+03	0.57257E+01
0.11057E+03	0.57274E+01	0.10492E+03	0.57292E+01	0.99880E+02
0.57312E+01	0.96020E+02	0.57334E+01	0.93032E+02	0.57354E+01
0.91573E+02	0.57373E+01	0.90624E+02	0.57390E+01	0.90848E+02
0.57407E+01	0.91378E+02	0.57423E+01	0.92520E+02	0.57439E+01
0.94581E+02	0.57456E+01	0.96337E+02	0.57474E+01	0.98707E+02
0.57488E+01	0.10245E+03	0.57505E+01	0.10589E+03	0.57523E+01
0.11025E+03	0.57538E+01	0.11461E+03	0.57558E+01	0.11862E+03

0.57579E+01	0.12227E+03	0.57595E+01	0.12693E+03	0.57613E+01
0.13094E+03	0.57634E+01	0.13484E+03	0.57655E+01	0.13788E+03
0.57674E+01	0.14184E+03	0.57692E+01	0.14385E+03	0.57714E+01
0.14592E+03	0.57731E+01	0.14814E+03	0.57749E+01	0.15002E+03
0.57766E+01	0.15230E+03	0.57784E+01	0.15418E+03	0.57802E+01
0.15585E+03	0.57820E+01	0.15793E+03	0.57836E+01	0.16114E+03
0.57850E+01	0.16517E+03	0.57867E+01	0.16903E+03	0.57882E+01
0.17413E+03	0.57899E+01	0.17871E+03	0.57915E+01	0.18482E+03
0.57926E+01	0.19118E+03	0.57939E+01	0.19904E+03	0.57957E+01
0.20577E+03	0.57976E+01	0.21389E+03		

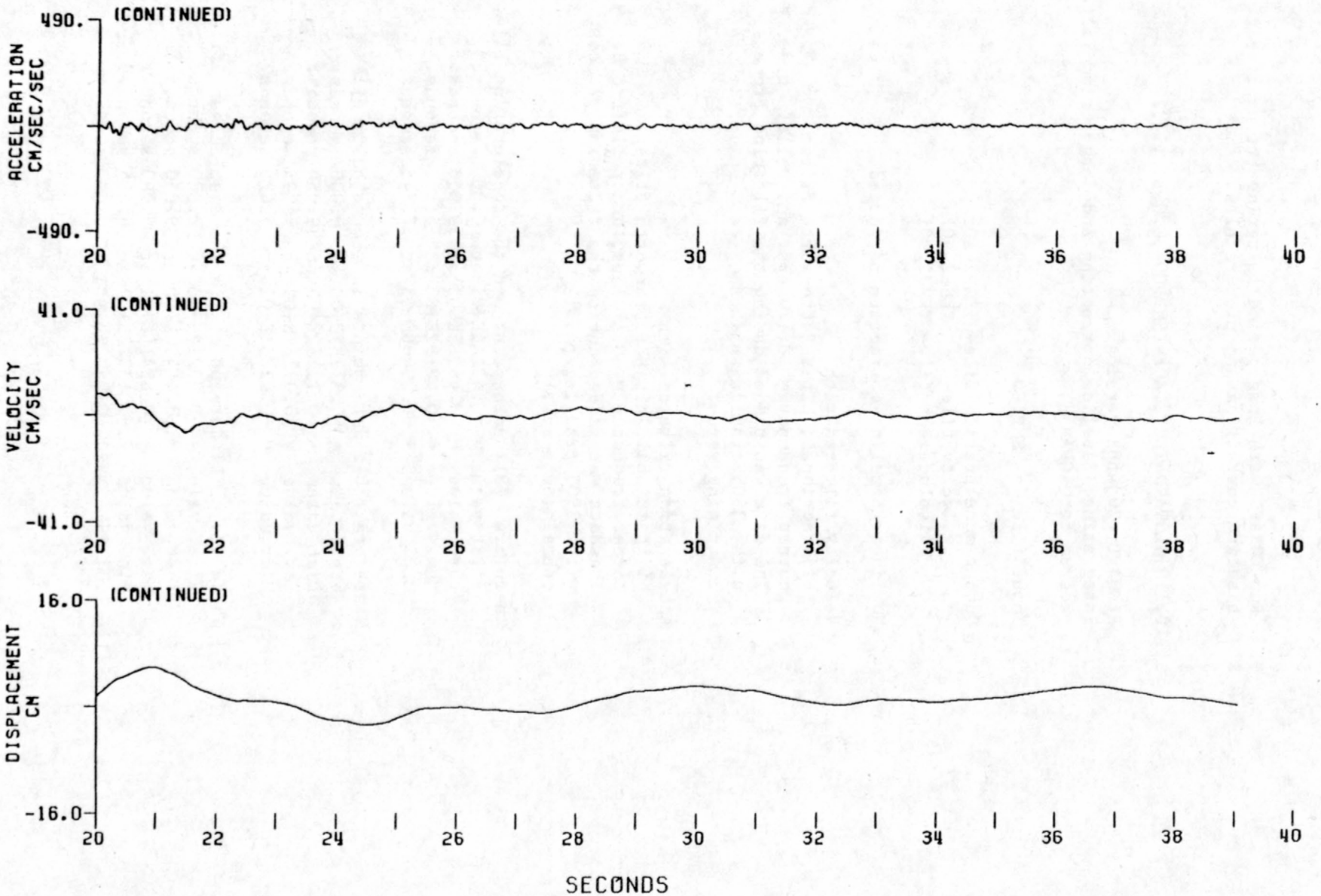
```
$!
$!
$!   Notice that the frequency-domain instrument correcting
$!   algorithm has nearly the same effect as the time-domain
$!   algorithm.
$!
$!
$[110,110]HIFRIC TEMP2.C01=TEMP.R01,.040,.570,FDIC

     HIFRIC, 11jan83 version.

     Input file  = TEMP.R01

     Output file = TEMP2.C01

     Instrument period = 0.040 seconds,
     instrument damping= 0.570 of critical damping.

     Instrument correction and anti-alias filter performed on densely
        interpolated data at 600.0 samples per second.
     Frequency domain instrument correction method used.
     Transition band for the anti-alias filter =  50.00 to 100.00 hz.
     Corrected, filtered data have been decimated by 1/3 to 200.0 samples
        per second.

     There are  7821 points in the trace.
        First value =  0.54676E+01 at  0.000 seconds.
        Last value  = -0.36864E+01 at 39.100 seconds.
        Max. value  =  0.48303E+03 at  5.550 seconds.
        Min. value  = -0.39476E+03 at  7.745 seconds.
$[110,110]VTSP TEMP.C01,TEMP2.C01, 5.0,7.0,2.0

     TSPLOT.
     Time series plotting program for AGRAM data,
     09dec82 version.
```


TEMP.CO1
TEMP2.CO1
484.0
490.
-490.
ACCELERATION
CM/SEC/SEC
6
1
483.0
490.
-490.
ACCELERATION
CM/SEC/SEC
6
1
SECONDS

```
$!
$!
$!   Run CORAVD with linear baseline correction to the velocity
$!   and without long-period filtering.  Plot results.
$!
$!
$[110,110]CORAVD TEMP.A01=TEMP.C01

     CORAVD, 20jan82 version.
     Integrating and Baseline correcting program for AGRAM data.

     Input file  = TEMP.C01

     Output acceleration file= TEMP.A01
            velocity file=     TEMP.V01
            displacement file= TEMP.D01

     Acceleration corrected by subtracting -0.043,
     Velocity corrected by subtracting the line with slope= -0.043
        and intercept=  2.237.
     Slope and intercept were calculated (in HIFRIC) as the linear least
        squares fit to the velocity between  0.000 and 39.050 seconds.

     No filtering performed.

     There are  7811 data points in the acceleration file.
        First value =  0.55071E+01 at  0.000 seconds.
        Last value  =  0.46552E+01 at 39.050 seconds.
        Max. value  =  0.48400E+03 at  5.550 seconds.
        Min. value  = -0.39509E+03 at  7.745 seconds.

     There are  7811 data points in the velocity file.
        First value = -0.22235E+01 at  0.000 seconds.
        Last value  = -0.89635E+00 at 39.050 seconds.
        Max. value  =  0.40599E+02 at  5.895 seconds.
        Min. value  = -0.40472E+02 at  5.475 seconds.

     There are  7811 data points in the displacement file.
        First value = -0.55588E-02 at  0.000 seconds.
        Last value  =  0.22427E-02 at 39.050 seconds.
        Max. value  =  0.14264E+02 at  7.770 seconds.
        Min. value  = -0.15696E+02 at 11.725 seconds.
$[110,110]VTSP TEMP.A01,TEMP.V01,TEMP.D01

     TSPLOT.
     Time series plotting program for AGRAM data,
     09dec82 version.
```

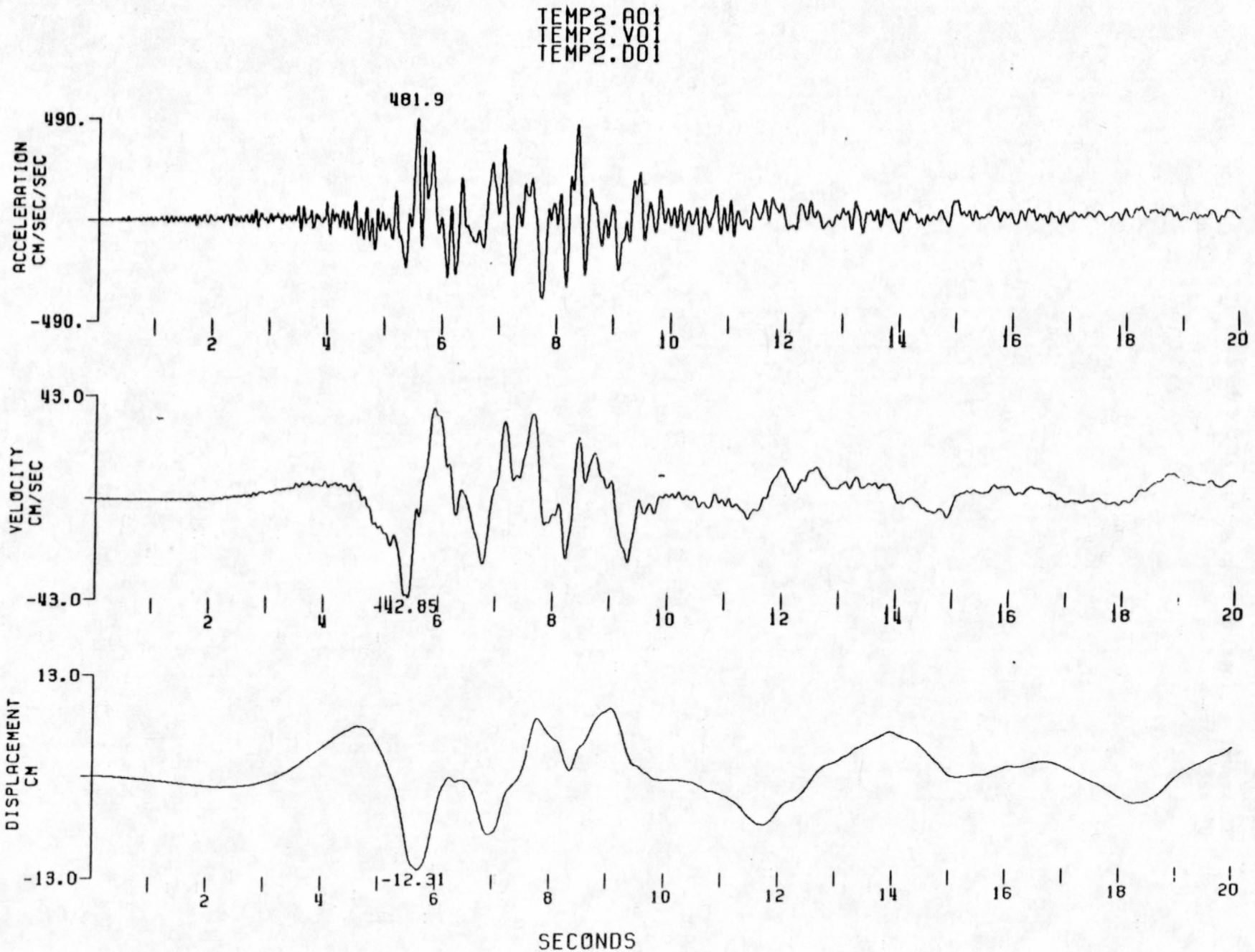

TEMP.A01
TEMP.V01
TEMP.D01
484.0
490.
ACCELERATION
CM/SEC/SEC
-490.
40.60
41.0
VELOCITY
CM/SEC
-41.0
16.0
DISPLACEMENT
CM
-16.0
-15.70
2
4
6
8
10
12
14
16
18
20
SECONDS

TEMP.A01
TEMP.V01
TEMP.D01

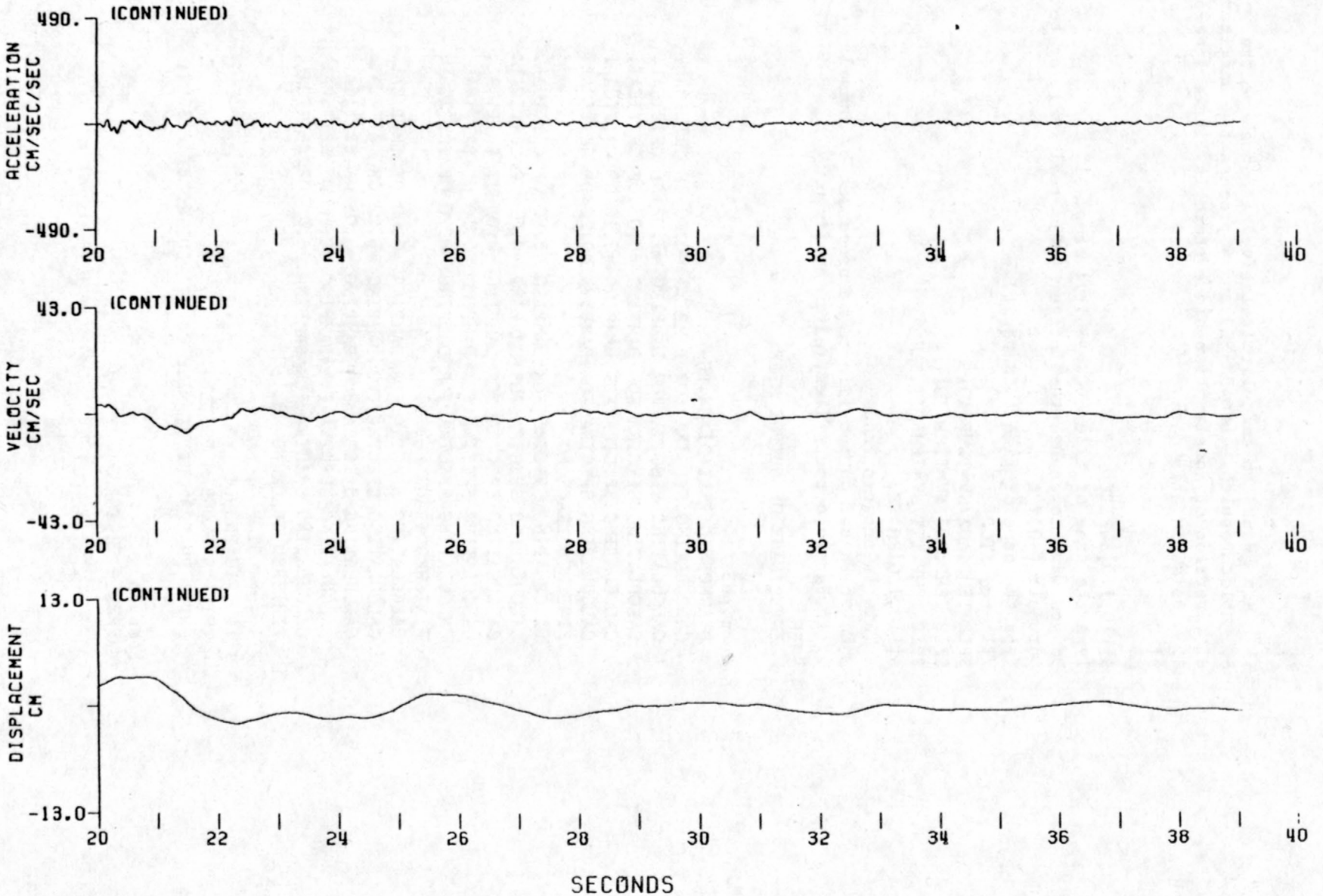


```
$!
$!
$!   Run CORAVD with long-period filtering but no linear
$!   baseline correction and plot results.
$!
$!
$[110,110]SLOWAVD TEMP2.A01=TEMP.C01,0,0,BI,.17,2

     SLOWAVD, 20jan82 version.
     Integrating, Baseline correcting and long-period filtering
     program for AGRAM data.

     Input file  = TEMP.C01

     Output acceleration file= TEMP2.A01
            velocity file=     TEMP2.V01
            displacement file= TEMP2.D01

     No linear baseline correction performed.

     Velocity filtered with
        bidirectional, hipas Butterworth filter,
        corner frequency =  0.170 cps, and rolloff order = 2
        The data was padded during the filtering process
        with  3905 trailing points
        containing zeros.

     Acceleration filtered with
        bidirectional, hipas Butterworth filter,
        corner frequency =  0.170 cps, and rolloff order = 2
        The data was padded during the filtering process
        with  3905 trailing points
        containing zeros.

     There are  7811 data points in the acceleration file.
        First value =  0.47242E+01 at  0.000 seconds.
        Last value  =  0.46585E+01 at 39.050 seconds.
        Max. value  =  0.48192E+03 at  5.550 seconds.
        Min. value  = -0.39066E+03 at  7.745 seconds.

     There are  7811 data points in the velocity file.
        First value = -0.37285E+00 at  0.000 seconds.
        Last value  =  0.12452E+00 at 39.050 seconds.
        Max. value  =  0.37381E+02 at  5.895 seconds.
        Min. value  = -0.42851E+02 at  5.475 seconds.

     There are  7811 data points in the displacement file.
        First value = -0.93212E-03 at  0.000 seconds.
        Last value  = -0.61490E+00 at 39.050 seconds.
        Max. value  =  0.82418E+01 at  9.060 seconds.
        Min. value  = -0.12313E+02 at  5.685 seconds.
$[110,110]VTSP TEMP2.A01,.V01,.D01
```

TSPLOT.
Time series plotting program for AGRAM data,
09dec82 version.

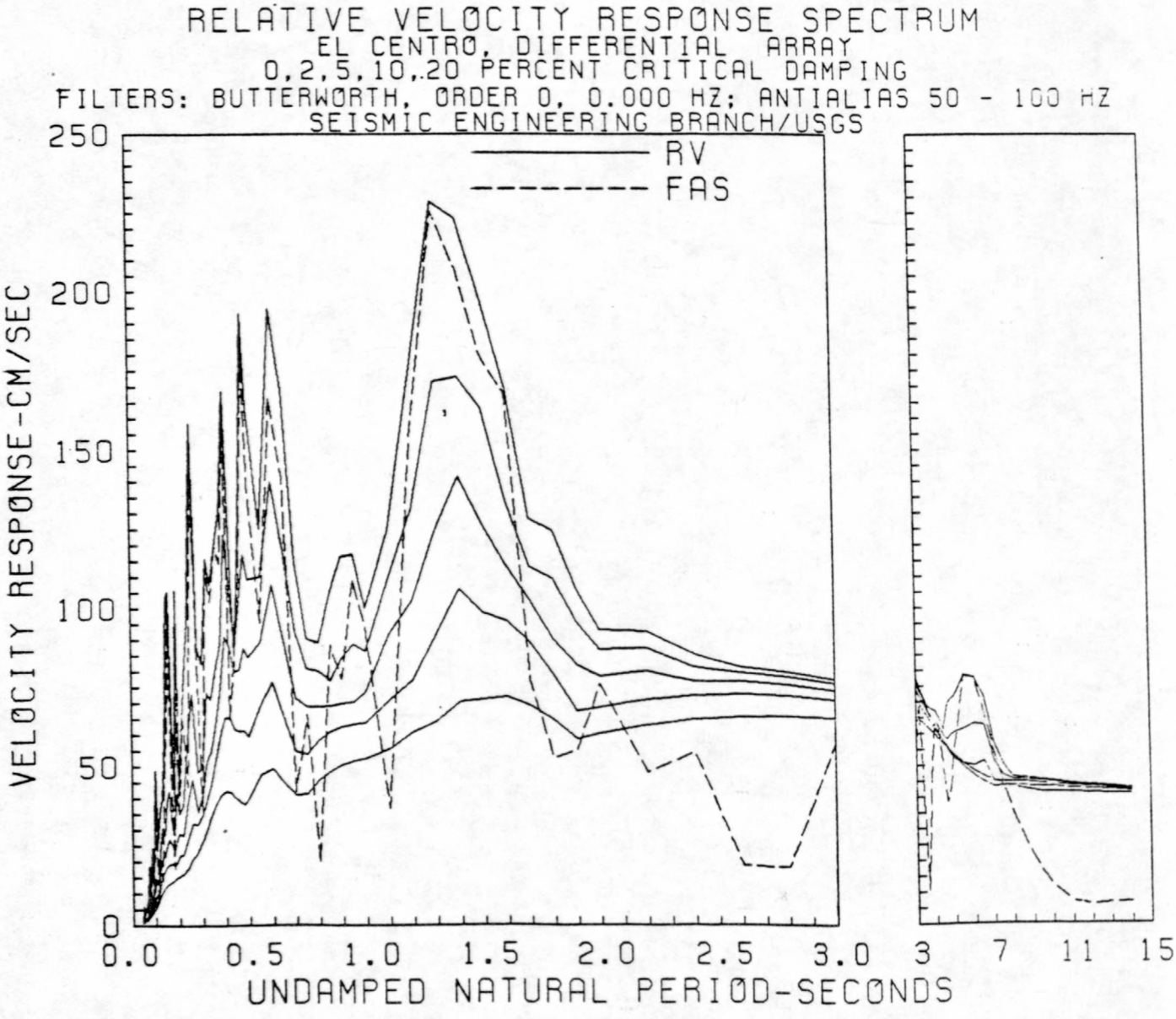

TEMP2.A01
TEMP2.V01
TEMP2.D01
481.9
490.
-490.
ACCELERATION
CM/SEC/SEC
43.0
-43.0
VELOCITY
CM/SEC
-42.85
13.0
-13.0
DISPLACEMENT
CM
-12.31
2
4
6
8
10
12
14
16
18
20
SECONDS

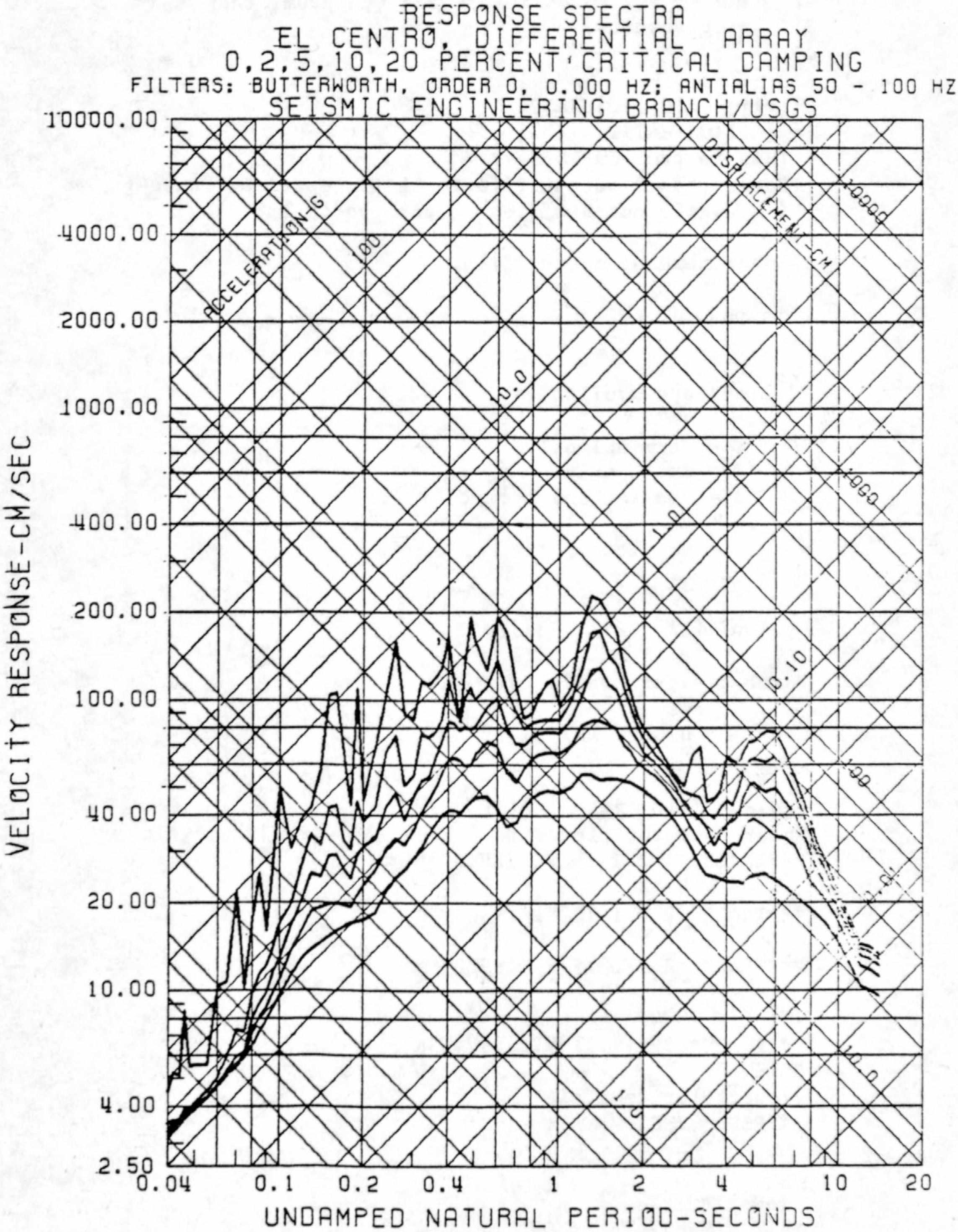
TEMP2.A01
TEMP2.V01
TEMP2.D01



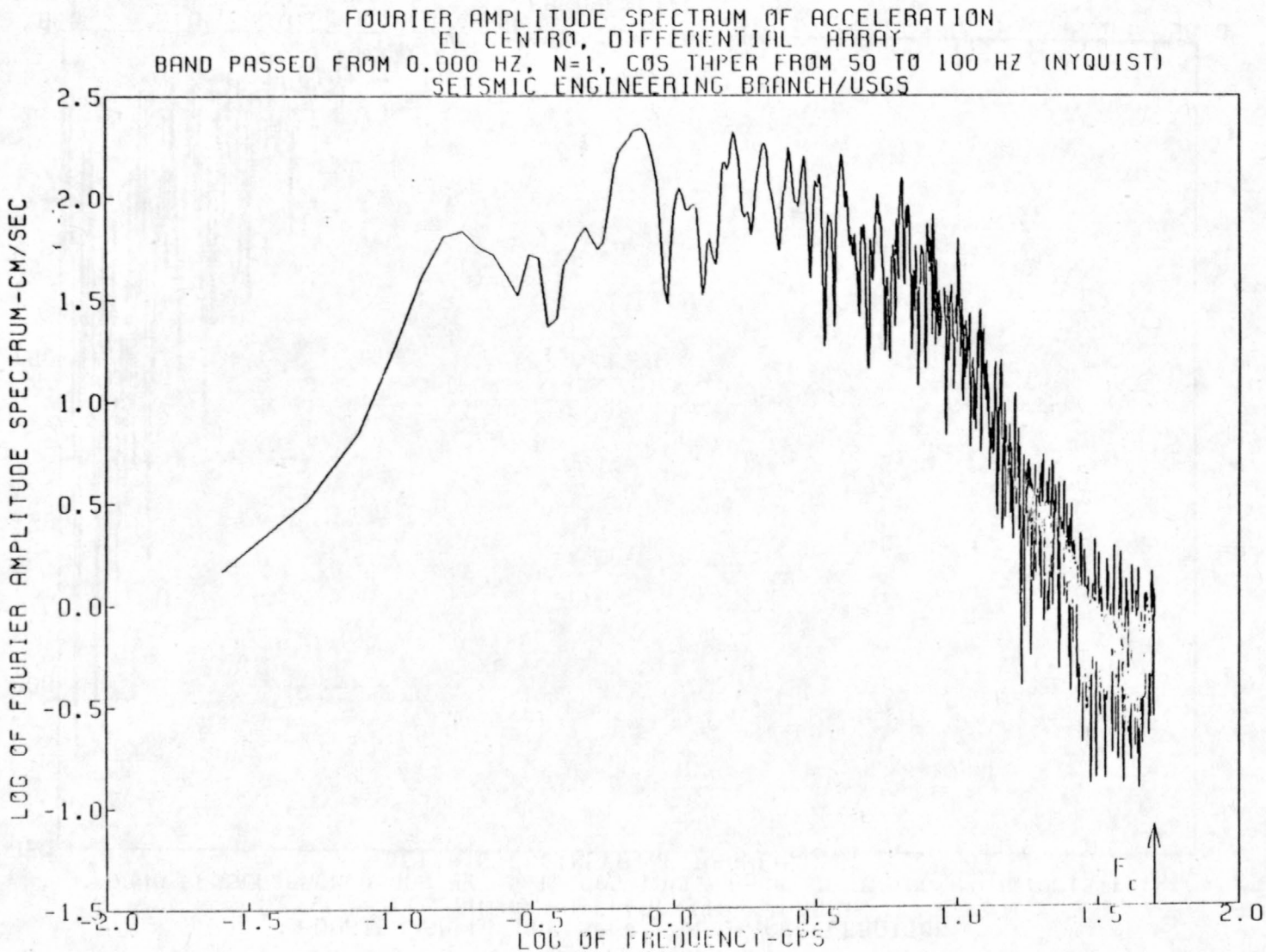
```
$!
$!
$!   Run PHASE3 to calculate relative response spectra, then
$!   plot results.  Let PHASE3 and 4 process the unfiltered
$!   version of acceleration since there is little long-period
$!   noise in this data.
$!
$!
$[110,110]BWRITE
Type the name of file (<CR>=quit): temp.a01
Do you want to change the integer (=I), real (=R), or text (=A) header,
 or quit (=Q)? q
Type the name of file (<CR>=quit):
VT1  --  STOP
$PIP TEMP.A02/NV=TEMP.A01
$PIP TEMP.A03/NV=TEMP.A01
$PIP TEMP.TIT/NV=TEMPTIT.SAV
$[110,110]PHASE3
  This is program Phase3.
  Enter Phase2 output file to be processed by Phase3:
  (If Faze2b output files, enter prefix only.) .
temp
  ENTER NAME OF OUTPUT FILE.

temp.ph3
  DIAGNOSTIC FILE=DIAGN.PH3
  CALCULATING SPECTRA FOR DAMPING =0.00 OF CRITICAL.
  CALCULATING SPECTRA FOR DAMPING =0.02 OF CRITICAL.
  CALCULATING SPECTRA FOR DAMPING =0.05 OF CRITICAL.
  CALCULATING SPECTRA FOR DAMPING =0.10 OF CRITICAL.
  CALCULATING SPECTRA FOR DAMPING =0.20 OF CRITICAL.
  FINISHED TRACE NO. 1.
  CALCULATING SPECTRA FOR DAMPING =0.00 OF CRITICAL.
  CALCULATING SPECTRA FOR DAMPING =0.02 OF CRITICAL.
  CALCULATING SPECTRA FOR DAMPING =0.05 OF CRITICAL.
  CALCULATING SPECTRA FOR DAMPING =0.10 OF CRITICAL.
  CALCULATING SPECTRA FOR DAMPING =0.20 OF CRITICAL.
  FINISHED TRACE NO. 2.
  CALCULATING SPECTRA FOR DAMPING =0.00 OF CRITICAL.
  CALCULATING SPECTRA FOR DAMPING =0.02 OF CRITICAL.
  CALCULATING SPECTRA FOR DAMPING =0.05 OF CRITICAL.
  CALCULATING SPECTRA FOR DAMPING =0.10 OF CRITICAL.
  CALCULATING SPECTRA FOR DAMPING =0.20 OF CRITICAL.
  FINISHED TRACE NO. 3.
VT1  --  STOP
$[110,110]PH3PLT
  THIS IS PROGRAM PH3PLT.
  ENTER PHASE3 OUTPUT FILE TO BE PLOTTED BY PH3PLT:

temp.ph3
MAPPED - VECTOR
```

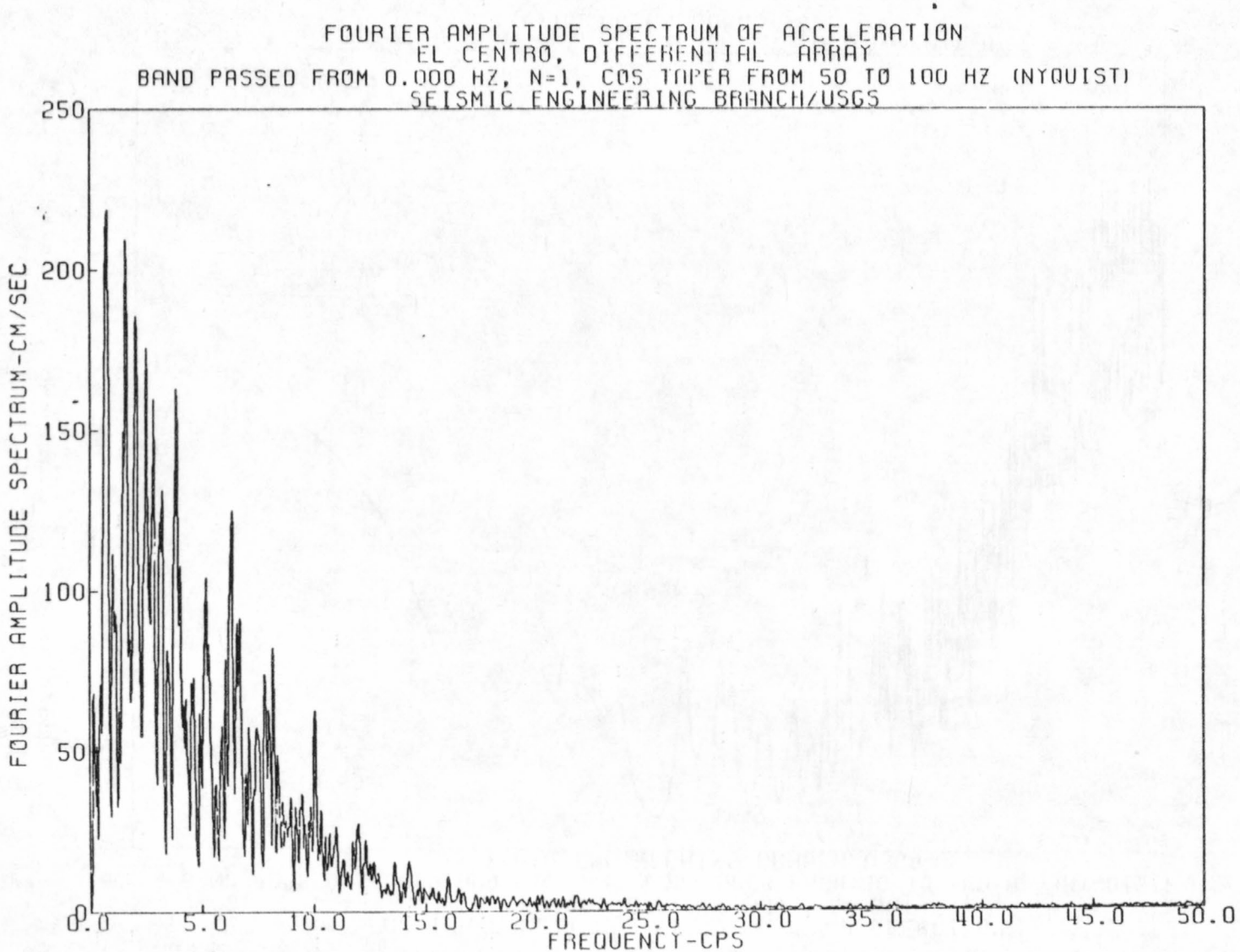
```
  FINISHED TRACE NO. 1.
  FINISHED TRACE NO. 2.
  FINISHED TRACE NO. 3.
VT1  —  STOP
```


RELATIVE VELOCITY RESPONSE SPECTRUM
EL CENTRO, DIFFERENTIAL ARRAY
0,2,5,10,20 PERCENT CRITICAL DAMPING
FILTERS: BUTTERWORTH, ORDER 0, 0.000 HZ; ANTIALIAS 50 - 100 HZ
SEISMIC ENGINEERING BRANCH/USGS
RV
FAS
VELOCITY RESPONSE-CM/SEC
250
200
150
100
50
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3
7
11
15
UNDAMPED NATURAL PERIOD-SECONDS


RESPONSE SPECTRA
EL CENTRO, DIFFERENTIAL ARRAY
0,2,5,10,20 PERCENT CRITICAL DAMPING
FILTERS: BUTTERWORTH, ORDER 0, 0.000 HZ; ANTIALIAS 50 - 100 HZ
SEISMIC ENGINEERING BRANCH/USGS
VELOCITY RESPONSE-CM/SEC
UNDAMPED NATURAL PERIOD-SECONDS
ACCELERATION-G
DISPLACEMENT-CM
10000.00
4000.00
2000.00
1000.00
400.00
200.00
100.00
40.00
20.00
10.00
4.00
2.50
0.04
0.1
0.2
0.4
1
2
4
10
20

```
$!
$!
$!   Run PHASE4 to calculate FFT spectrum, then
$!   plot results.
$!
$!
$! +++ $[110,110]PHASE4
$[110,110]PH4TST
  This is program Phase4.
  Enter Phase2 output file to be processed by Phase4:
  (If Faze2b output files, enter prefix only.)
temp
  Enter name of output file.

temp.ph4


Plot options available:

     1 = Terminal only
     2 = Batch only
     3 = Preview and prompt (1  2)
                                  -
     4 = No plots


Option? [I10;CR=    3]? 2

Batch devices available:

     1 = Disk file


Device? [I10;CR=    1]? 1
Enter name for plot file [CR=SY:BATCH.PLT] temp.plt
***TRACE  1 FFT IS FINISHED***

***TRACE  2 FFT IS FINISHED***

***TRACE  3 FFT IS FINISHED***

VT1  --  STOP
$PIP /NV=LB0:[7,11]VIEWERV.TSK
$VIEWERV
Viewer>Plot temp.plt
MAPPED - VECTOR


Viewer>quit
$PIP VIEWERV.TSK;*/DE
```


FOURIER AMPLITUDE SPECTRUM OF ACCELERATION
EL CENTRO, DIFFERENTIAL ARRAY
BAND PASSED FROM 0.000 HZ, N=1, COS THPER FROM 50 TO 100 HZ (NYQUIST)
SEISMIC ENGINEERING BRANCH/USGS
LOG OF FOURIER AMPLITUDE SPECTRUM-CM/SEC
2.5
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5
2.0
LOG OF FREQUENCY-CPS
F_c


FOURIER AMPLITUDE SPECTRUM OF ACCELERATION
EL CENTRO, DIFFERENTIAL ARRAY
BAND PASSED FROM 0.000 HZ, N=1, COS TAPER FROM 50 TO 100 HZ (NYQUIST)
SEISMIC ENGINEERING BRANCH/USGS
FOURIER AMPLITUDE SPECTRUM-CM/SEC
250
200
150
100
50
0
0.0
5.0
10.0
15.0
20.0
25.0
30.0
35.0
40.0
45.0
50.0
FREQUENCY-CPS

```
$!
$!
$!   How many temporary files are left?  They should either
$!   be renamed or deleted.
$!
$!
$PIP TEMP.*,TEMP2.*/LI


Directory DR2:[123,5]
16-JAN-83 16:18

TEMP.REF;1          1160.       16-JAN-83 15:03
TEMP.R01;1          377.        16-JAN-83 15:08
TEMP.R02;1          377.        16-JAN-83 15:08
TEMP.R03;1          377.        16-JAN-83 15:08
TEMP.C01;1          68.         16-JAN-83 15:09
TEMP.I01;1          68.         16-JAN-83 15:10
TEMP.A01;1          64.      C  16-JAN-83 15:12
TEMP.V01;1          64.      C  16-JAN-83 15:12
TEMP.D01;1          64.      C  16-JAN-83 15:12
TEMP.A02;1          64.      C  16-JAN-83 15:15
TEMP.A03;1          64.      C  16-JAN-83 15:15
TEMP.TIT;1          1.          16-JAN-83 15:15
TEMP.PH3;1          152.        16-JAN-83 15:15
TEMP.PH4;1          1077.       16-JAN-83 16:04
TEMP.PLT;1          439.        16-JAN-83 16:04
TEMP2.C01;1         68.         16-JAN-83 15:10
TEMP2.A01;1         64.      C  16-JAN-83 15:12
TEMP2.V01;1         64.      C  16-JAN-83 15:12
TEMP2.D01;1         64.      C  16-JAN-83 15:12

Total of 4676./4676. blocks in 19. files
$PIP TEMP.*;*,TEMP2.*;*/DE
$!
$!
$!   end of job.
$!
$END: CONTINUE
$EOJ
Connect time:  76    minutes
CPU time used: 4047  seconds
Task total:    80
```

APPENDIX B
Contents of Data File Header Blocks

A blocked binary data file generated by the AGRAM programs contains at least two blocks of auxiliary data at the beginning of the file. The first header block contains integer items, the second contains real items. Most of the locations in the first two header blocks are reserved for specific information as shown in the table below. The locations were assigned according to an organization originally established for DR100 data files with requirements for AGRAM files fitted in later. Many of the locations reserved for DR100 files are not used at all by AGRAM programs and many new locations were assigned in the "scratch" area to accomodate AGRAM data. The table identifies all reserved locations. Those locations that are actually set or read by any of the AGRAM programs are shown with a leading asterisk.

FIRST INTEGER HEADER BLOCK, 256 (2-BYTE) ELEMENTS LONG

*(1)	Number of optional integer header blocks that follow this one.
*(2)	Number of optional ASCII header blocks that follow the real header blocks.
*(3)	Integer value that represents "undefined" (usually = -32768).
*(4)	Real or integer data flag; = 1 if each data block contains 128 real (4-byte) elements, = item (3) if each data block contains 256 integer (2-byte) elements.
(10)	Year of the event
(11)	Julian day
(12)	Hour
(13)	Minute
(14)	Second
(15)	Millisecond
(16)	microsecond.
(17)	Sample number of first time mark
(20)	Serial number
(21)	Event number
(30)	Number of components recorded with (and including) this one.
*(31)	Number of data blocks (without headers).
*(32)	Index of the last sample in the last data block.
(40)	'FB' or 'VT' (fba or velocity transducer)
*(41)	Vertical orientation, a value between 0 and 180 degrees from up. This item will be 90 in data

files representing horizontal motion.
Note that before November 1982, this element was set to 'HR' or 'VR' to indicate whether the time series data represents horizontal or vertical motion.

*(42) Horizontal orientation, a value between 0 and 360 degrees clockwise from north. This item will be 0 in data files representing vertical motion.

*(43) 'AC', 'VL', or 'DP' (acc, vel, or disp) or 'BU' for BUTTER data, 'IR' for SCALE data

(44) This item is no longer used. Before November 1982, it was set to + or - 1 to indicate polarity of vertical motion. Vertical motion is now described by item (41).

*(200-256) Scratch area.

Scratch locations used in (REFORMatted) BUTTER files --

*(200) Number of digitized traces in the file. (There are usually 3 data traces, 1 or 2 reference traces, and 1 time tick trace.)

*(201) Type of trace,
1-> time-tick,
2-> reference line, and
3-> data.

*(202) Number of the first data block for the trace.

*(203) Number of data blocks for the trace.

*(204-?) Repeat items (201),(202), and (203) for each trace.

SECOND HEADER BLOCK, 128 REAL (4-BYTE) ELEMENTS LONG
==========

*(1) Number of optional real header blocks that follow this one.

*(2) Real value that represents "undefined" (usually =-0.3e-38).

*(5) Sampling rate, samples/second. If undefined, then it's output data from BUTTER or SCALE.

(10) Quake or shot lat, deg

(11) " , min

(12) " long, deg

(13) " , min

(14) Depth in kilometers

(15) Magnitude

(16) Moment

(17) Shot weight

(18) Quake or shot origin time

(19) Distance from station to epicenter

(20)	Azimuth of station from epicenter
(21)	Takeoff angle
(40)	Instr latitude, degrees
(41)	" , minutes
(42)	" longitude, degrees
(43)	" , minutes
(44)	Instr altitude
(45)	Voltage input
*(46)	Digitizer output, digitization units/volt for DR100 data, digitization units/cm for SMA data.
*(47)	Anti-alias filter's corner frequency, cps. (in AGRAM files, this is the freqency used with the combined instrument correction and anti-alias filter in HIFRIC. Other high-cut filters may be applied in CORAVD, but those filter characteristics are described in items (105) through (113).)
*(48)	Anti-alias filter's rolloff bandwidth, cps.
*(49)	Natural period of an SMA transducer, in seconds; or natural frequency of a DR100 transducer.
*(50)	Damping coefficient of the transducer, fraction of critical damping.
*(51)	Coil constant (volts/cm/sec/sec) for DR100 data, or Recorder sensitivity (cm/g) for SMA data.
(52)	Gain of a DR100 amplifier (volts/volt).
*(60)	Clock correction, seconds
(70-71)	P and S wave picks
*(90-128)	Scratch area

Scratch locations filled in SCALE, HIFRIC, and CORAVD output files --

*(90)	Time of the first data point, not including item(60), in seconds. (usually =0.0)
*(91)	Value of the first data point, in cm/sec/sec.
*(92)	Time of the last data point.
*(93)	Value "
*(94)	Time of the maximum value.
*(95)	Value "
*(96)	Time of the minimum value.
*(97)	Value "
*(98)	Offset of SCALE data. This value is subtracted from all data items in HIFRIC.
*(103)	Source-of-data indicator: 1.0 if from SCALE; 2.0 if from HIFRIC, with interpolation only; 2.1 if from HIFRIC, with Mike Raugh's instrument correction and anti-alias filter;

2.2 if from HIFRIC, with Bill Joyner's frequency-domain instrument correction;
3.0 if from CORAVD, without long-period filter or linear baseline correction to velocity;
3.1 if from CORAVD, without filter, with linear baseline correction;
3.2 if from CORAVD (or SLOWAVD) with filter and linear baseline correction.

Additional scratch locations filled in CORAVD files --

*(99) Beginning time for the least squares fit baseline correction.
*(100) Ending " "
*(101) Slope of the linear baseline correction.
*(102) Intercept " "

Additional scratch locations filled in SLOWAVD files --

*(104) Low frequency filter cut off, cps.
*(105) High " "
*(106) Low frequency filter rolloff,
*(107) High " ",
cps or integer rolloff order.
*(108) Low frequency filter type,
*(109) High " ",
=1.,2.,3. or 4. to indicate unidirectional Butterworth filter, bidirectional Butterworth filter, Ormsby filter, or FFT filter.
*(110) Low frequency filter data extension identifier (1.=> zero fill, 2.=> cosine fill, ... others, too)
*(111) High f ...
*(112) Low frequency Ormsby filter option,
*(113) High frequency Ormsby filter option,
(not included in the AGRAM package yet).

APPENDIX C
Printouts of Central Subroutines

C.1 Overview

The central, computational subroutines from the AGRAM programs are given in this appendix. Subroutines that perform other functions such as data handling, user-program interaction, or non-standard calculations are not printed here although all the subroutines are available for distribution on magnetic tape. The programs (BUTTER and PHASEs) that have not been reorganized recently are not printed here either. They will be given in future versions of this report, however, once the programs have been revised to isolate their computational processes from the rest of the code.

The contents of several "included files", as they are called in the RSX operating system in use at NSMDC, are given in this appendix too. These "included files" contain sections of code, usually common block specifications or definitions of constants, that are included in more than one subroutine or in more than one program. All the "included file" names have a ".inc" suffix.

Names of the subroutines and "included files" that are printed in this appendix are given in the table that follows. Wherever a subroutine that is not printed in this appendix is mentioned in the table, the purpose of that subroutine is described in parentheses following the subroutine name.

program	"included file" or subroutine	description
SCALE	SCALC	performs SCALE calculations for a single accelerometer trace. Calls READ and NOY.
SCALE	READ	reads the input file for SCALC.
SCALE	NOY	processes time-tick data for SCALC.
HIFRIC	ACALC	interpolates, instrument corrects, anti-alias filters, and decimates an acceleration time series generated by SCALE. Calls SETWTS, OPRTR, AFDIC,

		READXY (to read the input data), and HIOUT (to write output data).
HIFRIC	SETWTS	prepares weights for the combined instrument correction and anti-alias convolution. Calls SX1SIN, SX2COS, and S1COS, all of which are trigonometric functions whose code is given in the same deck with SETWTS.
HIFRIC	OPRTR	applies the weights calculated in SETWTS.
HIFRIC	AFDIC	applies frequency-domain instrument correction and anti-alias filter to acceleration. Calls VFORK (to transform a time series to a frequency-based series) and UFORK (to transform back to time.)
CORAVD	array.inc	This "included file" specifies the working storage array used in CORAVD and SLOWAVD and establishes the major distinction between the two programs. There are two versions: SARRAY.INC declares a standard array for CORAVD and VARRAY.INC declares a virtual array for SLOWAVD. Before subroutines are compiled, either SARRAY.INC or VARRAY.INC is copied to ARRAY.INC.
CORAVD	LLSQF	calculates coefficients of the linear least-squares fit to some portion of a time-series. Calls READY (to read the input file)
CORAVD	AVD	Calculates velocity and displacement and applies optional linear baseline correction. Calls READY (to read the input file) and FILLER (to fill out the last data block in the output files.)
CORAVD	FLTAVD	is an alternative version of AVD that can apply a filter to the acceleration and velocity in addition to the other AVD calculations. Calls FILTER, READY (to read the input file), OUTPUT (to write output files), and NOROOM (to truncate a time series that doesn't fit in the working storage space available).

CORAVD	FILTER	selects one of several filter algorithms. Calls BIHIP, UNIHIP (unidirectional high-pass Butterworth filter), FFTFLT (FFT filter), ORMFLT (Ormsby filter), NOROOM (to truncate a time series that doesn't fit in working storage), SHORTP (to give user notice that working storage space is not large enough to pad the data as much as needed), PAD (to pad the data wih trailing zeros), and NPWR2 (to calculate the nearest power of 2).
CORAVD	BIHIP	applies a bidirectional, high-pass Butterworth recursive filter.
library	RFFT	Computes the finite Fourier transform of a real array. Calls FOUR1.
library	RFFTI	Computes the inverse finite Fourier transform, returning a real array. Calls FOUR1.
library	FOUR1	This subroutine does the FFT for the FFT filter in CORAVD and for PHASE4. It will also eventually replace the FORK subroutine that now does the FFT in HIFRIC. FOUR1 is called through subroutines RFFT and RFFTI. There are two versions of these three subroutines on the library. RFFT, RFFTI and FOUR1 are the versions shown in this appendix; they operate on data in a standard array. The other version (VRFFT,VRFFTI and VFOUR1) operates on data in a virtual array.
library	files.inc	This "included file" defines various constants that may be used in calls to subroutines (e.g. SCLOSE, BIO) that do input/output operations.
library	headerloc.inc	This "included file" defines constants that serve as indices into arrays containing data file header blocks.

Several general support subroutines are called from so many of the subroutines listed in the table above that they are not included in their lists of called subroutines. These support subroutines are not printed in this appendix, but their functions are as follows:

WOE	(traps coding errors.)
BIO	(performs input, output, or positioning operations on blocked binary data files.)
BCLOSE	(closes a blocked binary data file.)
SCLOSE	(closes a standard fortran file.)
IUNDEF	(returns the value used to signify an undefined item in an integer header block of a blocked binary data file.)
RUNDEF	(returns the value used to signify an undefined item in a real header block.)

C.2 Non-Standard, Site-Dependent Code

If the programs are to be conveniently used outside the U.S.G.S., they must run on a variety of computers with a variety of operating systems. For this reason, FORTRAN 77 was chosen as the programming language and most site-dependent code was isolated in separate subroutines so the remainder of the code could be easily transported to other computers. Although transportability is an important goal in the development of these programs, it is rather low on the stack of priorities during early stages of program development and all the existing code still needs a thorough review of its portability.

The programs were written originally for a DEC PDP 11/70 computer with the RSX M-PLUS operating system. Some of the features this system provides that are not included in the ANSI FORTRAN standard are used in the AGRAM programs. The magnetic tape available for distribution that contains a copy of all the programs discused in this report also contains a text file named PROGAGRAM.NTS in which a list of the non-standard features used in the programs is given. Non-standard features used in the subroutines printed in this appendix are:

- "Included files".
  An "included file" is used to insert the same lines of code into several different subroutines. The contents of such a file are incorporated into a subroutine via an INCLUDE statement in the subroutine.

- Virtual arrays.
  A virtual array is a special array whose storage is allocated outside a program's directly addressable main memory. Access to these arrays is considerably slower than to standard arrays but they allow a program to process a long array without needing to break the data into several sections to be processed one at a time. Virtual arrays are used in SLOWAVD and the PHASEs although these programs should be rewritten in the future to use smaller, standard arrays.
  Virtual arrays are declared with a VIRTUAL statement

used in place of a DIMENSION statement.

- Debug statements.
  Statements used while debugging the programs are often left in the code for a long while, just in case they'll be useful again. Non-standard features used in debug statements include:
  . a D in column 1 indicates that the statement is compiled only during debug and is treated as a comment otherwise.
  . TYPE * statements are often used in debug as a convenient form of a WRITE statement.
  . comments following an exclamation sign (!) may be appended to a line of code.

- INTEGER*4 statements.
  On the PDP 11/70, an integer variable is ordinarily only half the length of a real variable. Unless declared differently, real variables are 4 bytes (32 bits) long and integer variables are 2 bytes (16 bits) long. A few integer variables in the programs that can become too large (>32767) to be stored in the two-byte integers have been declared to be 4 bytes long with an INTEGER*4 statement; these integers are usually those that count the number of data samples.

- PARAMETER statements.
  The compiler at NSMDC does not always process parentheses in PARAMETER statements properly. The parentheses have been omitted from almost all parameter statememts in the AGRAM code even though standard FORTRAN 77 requires them.

C.3 Coding Conventions

The following conventions were established to organize the code in a consistent manner:

- Most work space is allocated in a main program and passed to its subroutines through the argument lists. This makes it easy to adjust the size of the work space whenever necessary, as when adding new subroutines, rearranging the overlay structure of a program, or moving the program to another computer having a different amount of memory available to the program.

- Each subroutine statement is followed by declaration statements for the variables received through the argument list. Next are comments describing the subroutine's function and its argument list. The comments are followed by declaration statements for common blocks and for variables

local to the subroutine.

- Symbols defined with parameter statements in "included files" often start with the letter X, Y or Z as a reminder that they are names of constants, not variables. The X, Y or Z serves as a reminder that the constant is used primarily as an index (X), an array length(Y), or an assignment value (Z). Constants defined in local parameter statements often start with a W. Constants defined in HEADERLOC.INC all begin with an H as a reminder that they are header block locations.

- Subroutine WOE is used to abort a run when a coding error has been detected. Some errors that a user might make in specifying the run parameters are trapped with a call to WOE also. All user errors should eventually be given nice diagnostic messages, but a call to WOE serves in the meantime.

- Three asterisks (***) are used in comments in the code to attract a programmer's attention to code that needs to be added or rewritten. Three plus-signs (+++) are used to attract attention to less important problems. For instance, there are a lot of integer*4 declarations commented out with "C +++" in SLOWAVD to remind a programmer of variables that may need to be declared integer*4 once the program is moved to a machine (the VAX) that will allow arrays longer than the maximum of 32K elements allowed in arrays on the PDP 11/70.

- Singly-dimensioned arrays are occasionally used where multiply-dimensioned arrays would have been more straight-forward. There is no reason for doing this with FORTRAN 77, but some of the AGRAM code was written before FORTRAN 77 standardized the order of subscript progression.

C.4 Code Listings

```
      subroutine scalc(luser,ttick,smooth,x0time,dcm,sens,drdel,
     1                 iddin,iddout,
     2                 nrefb,iref0,  ref, lref,
     3                 ntickb,itick0,tick,ltick,
     4                 ndatab,idata0,data,ldata,
     5                 offset,iend,
     6                 vmax,tmax,vmin,tmin,vbeg,tbeg,vend,tend
     7                 )
      logical smooth
      dimension data (ldata), ref(lref), tick(-1:ltick)
cc
c   SCALC: perform SCALE calculations for a single data trace.
c
c   On entry --
c      luser   =logical unit number for the user's teminal or
c               run messages file.
c      ttick   =seconds between time ticks (usually 0.5).
c      smooth  =.true. if time-tick ordinates should be smoothed
c               with a (1/4, 1/2, 1/4) running average.
c               (usually =.true.)
c      x0time  =time of the first data point (usually =0.0)
c      dcm     =digitization units/cm.  IOM-TOWILL digitized data
c                is in microns, so dcm is usually 10000.
c      sens    =instrument sensitivity in cm/g
c      drdel   =vertical difference, in digitization units, between
c               the first point in the data trace and the first point
c               in the reference trace.
c      iddin   =bbfile id for the input file
c      iddout  =bbfile id for the output file.
c
c      nrefb   =number of reference trace blocks in the iddin file.
c      iref0   =number of the block before the first reference trace
c                block on the iddin file.
c      ref      i/o buffer for the reference trace data.
c      lref    =length of ref.
c
c      ntickb  =    ...time-tick trace.
c      itick0           "
c      tick             "
c      ltick            "
c
c      ndatab  =    ...data trace.
c      idata0           "
c      data             "
c      ldata            "
c
c   On return --
c      The iddout file contains SCALE data suitable for input to
c      HIFRIC or TSPLOT.  Data is unevenly sampled and undecimated.
c      Data are in cm/sec**2 and seconds, although abscissas include
c      an offset equal to the mean value of the whole trace.
c      Header blocks still need to be filled.
c
```

```
subroutine SCALC
program SCALE

c      offset,iend,vmax,tmax,vmin,tmin,vbeg,tbeg,vend,tend
c                  have been given values which should be inserted
c                  into the header blocks of the output files.
cc
      include 'files.inc'
      integer*4 ntot
      logical warntl,warnrl
      parameter  huge=1.0e30, tiny=1.0e-30
cc
c   Make sure i/o buffer lengths are even multiples of YFBLK, the size
c     of the blocks in blocked binary data files.
cc
      msized=ldata/yfblk
      msizer=lref /yfblk
      msizet=ltick/yfblk
      if(msized.lt.1 .or. msizer.lt.1 .or. msizet.lt.1) call woe(0)
      ldbuf=msized*yfblk
      lrbuf=msizer*yfblk
      ltbuf=msizet*yfblk
      idtick=iddin
      idref=iddin
cc
c   Vertical scale factor=
c     (cm per sec-squared/g)/((instrument sensitivity in cm/g)
c                             *(digitization units/cm))
cc
      scalef=980.665/(sens*dcm)
cc
c   Get ready.  Read first reference trace block
c               Read first time tick block
c               Read first data trace block
c               Position output file
c               etc.
c   Note that the first point in the reference trace is sometimes
c   digitized several times.  Set IR accordingly.  The time ticks
c   are also sometimes digitized more than once, but the BUTTER
c   program has already removed those extra time tick values.
cc
      irb=1
      call read(idref,nrefb, iref0,irb , ref(1),lrbuf, nr)
         if(nr.lt.4) call woe(0)
      rx1=ref(1)
      ry1=ref(2)
      do 10 ir=4,nr,2
         rx2=ref(ir-1)
         ry2=ref(ir)
         if(rx2.gt.rx1) go to 11
   10 continue
         call woe(0)
   11 continue
      slope1=(ry2-ry1)/(rx2-rx1)
      slopet=0.0
      warnrl=.false.
```

```
      rbig = ref(nr-1)-rx1
cc
      itb=1
      call read(idtick, ntickb,itick0,itb ,tick(1),ltbuf, nt)
         if(nt.lt.4) call woe(0)
         if(smooth .and. nt.lt.6) call woe(0)
      call noy(tick(1),nt)
      tx1=tick(1)
      tx2=tick(2)
      if(tx1.ge.tx2) call woe(0)
      it=2
      if(smooth) tx2=0.5*tx2 + 0.25*(tick(it-1)+tick(it+1))
      tdel=tx2-tx1
      tick(0)=tick(1)-tdel
      tick(-1)=tick(0)-tdel
      warntl=.false.
      ttt=0.0
      hltbuf=ltbuf/2
      fulltb=ttick*float(yfblk/2)
cc
      inone=iundef(iddin)
      rnone=rundef(iddin)
      if(inone.ne. iundef(iddout)) call woe(0)
      if(rnone.ne. rundef(iddout)) call woe(0)
      call bio(iddout,zpos,zwaita,zdata1,zf, data(1),ldbuf, nd)
         if(nd.lt.0) call woe(0)
      idb=1
      call read(iddin, ndatab,idata0,idb  ,data(1),ldbuf, nd)
cc
c   Set diagnostic variables
cc
      ylast=0.0
      tlast=-tiny
      if(x0time.ne.0.0) tlast=x0time*0.9999999
      kbs=0
      bsmax=0.0
      ww=0.0
      dtot=0.0
      ntot=0
      vmin=huge
      vmax=-huge
      tbeg=x0time
      tearly=x0time
      tlate=-huge
cc
c    Calculate TTICK1 as the time at the first tick given in
c      the time-tick trace.  Set other timing variables too.
cc
      x=data(1)
      tfactr= ttick/tdel
      ttick1= x0time - (x-tx1)*tfactr
      timtx1=ttick1
      tref1=  x0time - (x-rx1)*tfactr
```

```
subroutine SCALC
program SCALE

      xtick1=tx1
      xref1 =rx1
      ttickn=0.0
      trefn =0.0
      xtickn=huge
      xrefn =huge
cc
c   Loop over all data points in all blocks for the trace --
cc
      nloops =1 + (ndatab-1)/msized
      do 100 loop=1,nloops
      do 110 idx =1,nd,2
      idy=idx+1
      x=data(idx)
cc
c   a) Find the two time-ticks that bracket the current data point.
c       If the data point falls beyond the digitized time ticks,
c         continue using the nearest time tick interval.
c       Watch that time-tick intervals don't vary much.
c       And smooth time-ticks if requested.
cc
c  *** proof-read paragraphs a) and b) once more
c  *** consider especially whether all is Ok with reread forward
c  *** after having backed up one block.
      if(x.ge.tx1 .and. x.le.tx2) go to 120
  121 if(x.ge.tx1) go to 125
      if(tx1.eq.xtick1) then
         warnt1=.true.
d        type *, 'DEBUG, extend leading ticks.'
         go to 129
      endif
      type *, 'DEBUG, backup one tick:  x,tx1,tx2,it,itb='
      type *, '    ', x,tx1,tx2,it,itb
      tx2=tx1
      it=it-1
      if(it.lt.1) then
      type *, 'Warning: back stepped into the previous tick block.'
      type *, '         Using code I haven t tested yet.  -- April.'
      type *, '         x,tx1,it,itb,timtx1=', x,tx1,it,itb,timtx1
         itb=itb-msizet
         ttt=fulltb*(itb-1)
         if(itb.lt.1) call woe(0)
         call read(idtick, ntickb,itick0,itb, tick(1),ltbuf, nt)
         call noy(tick(1),nt)
         tdel=tick(2)-tick(1)
         tick(0)=tick(1)-tdel
         tick(-1)=tick(0)-tdel
         it=nt
      endif
      tx1=tick(it-1)
      if(smooth .and. tx1.ne.xtick1)
     1   tx1=0.5*tx1 + 0.25*(tick(it-2)+tick(it))
      tdel=tx2-tx1
```

```
      go to 121
  125 continue
      if(x.le.tx2) go to 129
      if(tx2.eq.xtickn) go to 129
      tx1=tx2
      it=it+1
      if(it.ge.nt) then
         if(itb+msizet.le.ntickb) then
            itb=itb+msizet
            ttt=fulltb*(itb-1)
            it=it-nt
            tick(-1)=tick(nt-1)
            tick(0) =tick(nt)
            call read(idtick, ntickb,itick0,itb,  tick(1),ltbuf, nt)
            call noy(tick(1),nt)
         else
            if(xtickn.lt.huge) call woe(0)
            xtickn= tick(nt)
            ttickn= timtx1 + ttick
d            type *, 'DEBUG, extending trailing ticks:'
d            type *,  tx1,tx2,xtickn,ttickn,it,itb,x,idx,idb
         endif
      endif
      tx2=tick(it)
      if(tdel.le.tiny) then
         type *, 'tdel=0.0 ???', tdel,x,idx,idb,tx1,tx2,it,nt,itb
         call woe(0)
      endif
      w=(tx2-tx1-tdel)/tdel
      if(abs(w).gt.abs(ww))then
          ww=w
          nww= it-1  + (itb-1)*YFBLK
      endif
      if(smooth .and. tx2.ne.xtickn)
     1    tx2=0.5*tx2 + 0.25*(tick(it-1)+tick(it+1))
      tdel=tx2-tx1
      go to 125
  129 continue
      if(tx1.ge.tx2) call woe(0)
      timtx1 = ttick1 + ttt + ttick*float(it-2)
      tfactr = ttick/tdel
  120 continue
cc
c   b) Find two reference points that bracket the current data point.
c      If the data point extends beyond the digitized reference
c        trace, continue using the slope of the nearest two
c        reference points.
cc
      if(x.ge.rx1 .and. x.le.rx2) go to 130
  131 if(x.ge.rx1) go to 135
      if(rx1.eq.xref1) then
         warnr1=.true.
d         type *, ' DEBUG, extending leading reference line.'
```

```
subroutine SCALC
program SCALE

         go to 135
      endif
      type *, 'DEBUG,  backstep one reference point:'
      type *, '   x,rx1,rx2,ir,irb=', x,rx1,rx2,ir,irb
      rx2=rx1
      ry2=ry1
      ir=ir-2
      if(ir.ge.4) then
         rx1 = ref(ir-3)
         ry1 = ref(ir-2)
      else
      type *, 'Warning: back stepped into the previous ref. block.'
      type *, '         Using code I haven t tested yet.  -- April.'
      type *, '         x,rx1,rx2,ir,irb=', x,rx1,rx2,ir,irb
         if(irb.le.msizer) call woe(0)
         irb=irb-msizer
         call read(idref, nrefb,iref0,irb ,ref(1),lrbuf, nr)
         ir=nr
         rx1 = ref(ir-1)
         ry1 = ref(ir)
      endif
      go to 131
  135 if(x.le.rx2) go to 139
      if(rx2.eq.xrefn) go to 139
      rx1=rx2
      ry1=ry2
      ir=ir+2
      if(ir.le.nr) then
         rx2=ref(ir-1)
         ry2=ref(ir)
      else
         if(irb+msizer.le.nrefb) then
            irb=irb + msizer
            call read(idref, nrefb,iref0,irb, ref(1),lrbuf, nr)
            ir=2
            rx2=ref(1)
            ry2=ref(2)
         else
            rx1=ref(nr-3)
            ry1=ref(nr-2)
d            type *, 'DEBUG, extending trailing reference line.'
d            type *, '  ',x,idx,nd,idb,rx1,rx2,ry1,ry2,ir,nr,irb
            if(xrefn.lt.huge) call woe(0)
            slopet = (ry2-ry1)/(rx2-rx1)
            trefn  = timtx1 + (rx2-tx1)*tfactr
            xrefn  = rx2
         endif
      endif
      go to 135
  139 continue
      if(rx1.ge.rx2) call woe(0)
  130 continue
cc
```

```
c    c) Subtract reference trace from data trace.
c       Subtract their beginning difference too, just to keep
c         the values near zero.
cc
       r=ry1 +  (x-rx1)*(ry2-ry1)/(rx2-rx1)
       data(idy) = data(idy) -r-drdel
cc
c    d) Convert abscissa to cm/sec**2.
cc
       data(idy)=data(idy)*scalef
cc
c    e) Convert ordinate to seconds.
cc
       t          = timtx1 + (x-tx1)*tfactr
       data(idx) = t
cc
c    f) Count number of back-stepping points.
cc
       if(t.le.tlast)then
          kbs=kbs + 1
          if (tlast-t .gt. bsmax) then
             bsmax=tlast-t
             tbsmax= tlast
          endif
          if(tlast.gt.tlate) tlate=tlast
          if(t.lt.tearly) tearly=t
       endif
cc
c    g) Calculate the mean value of the whole trace
c         so it can be subtracted out by the next program.
c       Get min and max values too
cc
       if(abs(dtot) .gt. huge) call woe(0)
       dtot=dtot + (data(idy) + ylast)*(t-tlast)
       tlast=t
       ylast=data(idy)
       ntot=ntot +1
       if(vmin.gt.data(idy)) then
          vmin=data(idy)
          tmin=data(idx)
       endif
       if(vmax.lt.data(idy)) then
          vmax=data(idy)
          tmax=data(idx)
       endif
cc
c    End loops.
cc
  110 continue
       call bio(iddout,zwrite,zwaita,znextb,zf, data(1),ldbuf, nd)
          if(nd.lt.0) call woe(0)
       idb=idb+msized
       if(loop.eq.1) vbeg=data(2)
```

```
C-14
subroutine SCALC
program SCALE

      if(loop.ne.nloops)
     1call read(iddin, ndatab,idata0,idb   , data(1),ldbuf, nd)
  100 continue
      vend=data(idy)
      tend=data(idy-1)
      iend= mod(idy,yfblk)
      if(tlate.lt.tend )  tlate=tend
      offset =dtot*0.5/(tend-tbeg)
cc
c   Report to user.
cc
      write(luser,1001) ntot,tend,offset,VMAX,TMAX,VMIN,TMIN
      if (abs(bsmax).gt.0.01) write(luser,1000)
      if(kbs.gt.0) write(luser,1002) kbs, bsmax,tbsmax
      if(abs(ww).gt.0.025) write(luser,1000)
      write(luser,1003) ww*100. ,nww
      if(warntl) write(luser,1004) tearly,ttickl
      if(ttickn.ne.0.0) write(luser,1005) tlate,ttickn
      if(warnrl) then
         if(abs(slopel).gt.0.025) write(luser,1000)
         write(luser,1006) trefl,tearly,slopel
      endif
      if(trefn.ne. 0.0) then
         if(abs(slopet).gt.0.025) write(luser,1000)
         write(luser,1007) trefn,tlate,slopet
      endif
      return
cc
 1000 format (5x,'*** Warning:')
 1001 format (8x, 'There are', i6, ' points in the trace.'
     1/8x, 'Trace length =',f7.3, ' seconds.'
     2/8x, 'Mean value = ', e12.5, ' cm/sec**2.'
     3/8x, 'Max. value = ', e12.5,' at',f7.3,' seconds.'
     4/8x, 'Min. value = ', e12.5,' at',f7.3,' seconds.')
 1002 format (8x, 'There are', i4
     1,' back-stepping points, the largest of which is for',
     2/11x, f7.5,' seconds at time =', f7.3,' seconds.')
 1003 format (8x, 'The largest difference between two consecutive'
     1,' time-tick'
     2/11x,' intervals was', f6.3, ' % at tick number',i2,'.')
 1004 format (8x, 'The first data point (at', f6.3
     1, ' sec.) occurred before'
     2/11x, 'the first time tick (at ',f6.3, ' sec.).'  )
 1005 format (8x, 'The last data point (at ', f6.3
     1, ' sec.) occurred after'
     2/11x, 'the last time tick (at ',f6.3, ' sec.).'  )
 1006 format (8x, 'The first point in the reference trace (at '
     1  ,f6.3, ' sec.)'
     2/11x,'occurred after the first point in the data trace (at'
     3  ,f6.3, ' sec.),'
     4/11x,'so the reference line was extended with the slope (='
     5,  f6.3, ')'
     3/11x,'of the first two points given in the reference trace.')
```

```
 1007 format (8x, 'The last point in the reference trace (at '
     1  ,f6.3, ' sec.)'
     2/11x,'occurred before the last point in the data trace (at '
     3  ,f6.3, ' sec.),'
     4/11x,'so the reference line was extended with the slope (='
     5,  e12.5, ')'
     3/11x,'of the last two points given in the reference trace.')
cc
c
ccccccccccccccccccc (end of scalc) ccccccccccccccccccccccccccc
      end
      subroutine read(idbbf, nbtot,ib0,iblock, array, larray, nd)
      dimension array(larray)
cc
c   READ is called from SCALC to read data into a buffer and check
c     for undefined values at the end.
cc
      include 'files.inc'
cc
      nread=larray
      if(iblock + (larray-1)/YFBLK .gt.nbtot)
     1   nread=(nbtot-iblock+1)*YFBLK
      call bio(idbbf,zread,zwaita,ib0+iblock,zf, array(1),nread, nd)
         if(nd.lt.yfblk) call woe(0)
      rnone=rundef(idbbf)
      do 9 j=YFBLK,nd,YFBLK
      if(array(j).ne.rnone) go to 9
      do 10 i=j,1, -1
      if(array(i).ne.rnone) go to 11
   10 continue
      call woe(0)
   11 continue
      nd=i
      go to 12
    9 continue
   12 continue
      return
ccccccccccccccccccc (end of read/scalc) ccccccccccccccccccccc
      end
      subroutine noy(tick,nt)
      dimension tick(1)
cc
c   NOY is called from SCALC to squish y values out of the time-tick
c     array.   (+++ NOY is needed only because I wrote SCALC assuming
c     that y values wouldn't be in the time tick trace, then changed
c     my mind.  If we do continue to give x and y coordinates in the
c     time tick trace, we should rewrite SCALC to get rid of NOY.
c     -- April. +++)
cc
      nt=nt/2
      if(nt.le.1) return
      do 10 i=2,nt
   10 tick(i)=tick(1 + (i-1)*2)
```

```
C-16
subroutine SCALC
program SCALE

      return
ccccccccccccccccccc (end of noy/scalc) ccccccccccccccccccccccccc
      end
```

```
      subroutine ACALC(luser,
     1           ictype,period,sdamp,sps2,ndense,kbands,fc,fz,
     2           iddin,iddout,offset,dr100f,dr100s,
     3           ndblks,ilst,xy,llxy,acc,llacc,buf,lbuf)
      dimension xy(llxy),acc(llacc),buf(lbuf)
cc
c   ACALC:  Interpolate, instrument correct, anti alias filter, and
c            decimate an acceleration time series for program HIFRIC.
c
c   On entry --
c     luser     =logical unit number for dignostics and run messages.
c     ictype    =instrument correction algorithm indicator.
c               =0 if no instrument correction is wanted, just
c                  interpolation.
c               =1 (=wTDIC) if Mike Raugh's time-domain instrument
c                  correction shall be applied.
c               =2 (=wFDIC) if Bill Joyner's frequency-domain
c                  instrument correction shall be applied.
c     period    =recording instrument's period, seconds.
c     sdamp     =recording instrument's damping, as a fraction of
c                critical damping.
c     sps2      =final sample rate, after interpolation and decimation.
c                (usually =200).
c     ndense    =ratio of dense to decimated sample rates. (usually=3).
c                The sample rate at which instrument correction and
c                filter are applied = SPS2*NDENSE, then data are
c                decimated by removing NDENSE-1 out of every NDENSE
c                points.
c     fc,fz     =roll-off band for anti-alias filter.
c                (Usually =50. and 100.).
c     kbands     is used to determine FC and FZ when ICTYPE =wTDIC.
c               =number of equal-width bands in which to divide
c                frequencies from 0 to nyquist.
c                (Nyquist = SPS2*NDENSE/2, KBANDS usually =6, so
c                 bandwidths usually =50hz.)
c     iddin,iddout   blocked binary file identifiers for the input and
c                output data files.
c     offset,dr100f,dr100s    are adjusting factors for the input data.
c                See comments in subroutine readxy.
c     ndblks     Number of data blocks in the input file
c     ilst       Index of the last data item in the last data block.
c     xy(llxy)   space for an input data buffer.  It will contain
c                uninterpolated data, as (x,y) pairs of values.
c     acc(llacc) space for an output buffer.  It will contain
c                interpolated data, as y-values only.
c                LLXY and LLACC should be even multiples of the
c                datafile blocking factor, YFBLK.  If not, the excess
c                just goes unused.  LLXY should be at least 4*YFBLK if
c                the DR100 data is to be read.
c     buf(lbuf)  space for an intermediate data buffer.  It will
c                contain densely sampled, interpolated data.
c                The minimum length for buf depends on which instrument
c                correcting algorigthm is used.
```

```
subroutine ACALC
program HIFRIC

c                For TDIC, LBUF must be at least 61, but the more the
c                better.
c                For FDIC, LBUF must be at least 1154.  That is:
c                       128   (=nlap in subroutine AFDIC)
c                     +   2   (for the 2 extra cells required by the
c                              FFT used in AFDIC)
c                     +1024   (=amount of data processed with each call
c                              to the FFT in AFDIC.)
c
c   On return --
c     The iddout file contains uniformly sampled, decimated,
c     instrument corrected data.
cc
c   Convolution operator arrays and their size are defined here
c     although they may become input arguments someday.
cc
      parameter (nwts=30,  nwts2=2*nwts)
      double precision bn(nwts+1), cn(nwts+1), an(nwts+1)
cc
      include 'files.inc/nolist'
      double precision time, delt1, delt2, dr100t, dr100i
      integer*4 ntot
      parameter (wNOIC=0, wTDIC=1, wFDIC=2)
      parameter (small=1.0e-5)
cc
      nbxy=llxy/YFBLK
      if(nbxy.lt.1) call woe(0)
      lxy=nbxy*YFBLK
cc
      nbacc=llacc/YFBLK
      if(nbacc.lt.1) call woe(0)
      lacc=nbacc*YFBLK
cc
      if(lbuf.lt.nwts2+1) call woe(0)
cc
      ntot=0
      dr100i=1.0
      if(dr100s.ne.0.0) dr100i=dr100i/dr100s
      dr100t=-dr100i
cc
      delt2=sps2
      delt1=delt2
      delt2=1.0/delt2
      delt1=delt1*ndense
      sps1=delt1
      delt1=1.0/delt1
cc
      assign 7011 to lblic
      assign 7002 to label
      m0 = 0
      m2 = 0
cc
c   Distinguish between the 3 alternate instrument correcting
```

```
c   options --
c   a) time-domain instrument correction.
c      Set convolution operator's weights,
c      arrange the first NWTS2 loctions of the intermediate buffer
c      for wrap-around space while applying the weights.
cc
      If (ictype.eq.wTDIC) then
         Call SetWts (nwts, kbands, 2, deltl, an, bn, cn)
         w0 = (2.0*3.1415926536)/period
         if(mod(kbands,2).gt.0 .and. ndense.gt.1) call woe(0)
         if(ndense.ne.1 .and. ndense.ne.kbands/2) call woe(0)
         bandw = spsl/float(2*kbands)
         if(abs(bandw-fc).gt.small) call woe(0)
         if(abs(2.0*bandw-fz).gt.small) call woe(0)
         fc=bandw
         fz=2.0*bandw
         do 10 m0=1,nwts2
   10    buf(m0)= 0.0
         m0 = nwts2
         m2 = -nwts
         nweed = 0
cc
c   b) frequency-domain instrument correction.
cc
      else if(ictype.eq. wFDIC) then
         assign 7012 to lblic
         ndata=lbuf
         ndone=-1
         nstart=1
cc
c   c) no instrument correction, just interpolate.
cc
      else if(ictype.eq.wNOIC) then
         assign 7001 to label
      else
         call woe(0)
      endif
cc
c   Begin outer loop to process each buffer full of data from
c     the input file.
c   The loops process three buffers.
c     IY references the input data buffer, XY(IY),
c     M0 references the intermediate data buffer, BUF(M0),
c     M2 references the output data buffer, ACC(M2).
cc
      nxy=lxy
      nreads=1 + (ndblks-1)/nbxy
      nxylst= ilst + YFBLK*(ndblks-1 -(nreads-1)*nbxy)
      if(nreads.le.0) call woe(0)
      do 200 j=1,nreads
cc
c   Read next buffer full of data.
c   (Note that READXY may reset NXY when reading the last
```

```
subroutine ACALC
program HIFRIC

c     bufferful of DR100 data.)
cc
      if (j.eq.nreads) nxy=nxylst
      call readxy(iddin,offset,dr100f,dr100i,dr100t,
     1            xy,xy,lxy,nxy)
cc
c   If it's the first buffer full, catch the first point.
cc
      If (j.GT.1) then
         iy=0
      else
         iy=2
         xl = xy(1)
         yl = xy(2)
         time = xl
         tfirst=time
      endif
      go to 110
cc
c   Begin inner loop to process each undecimated, interpolated
c     sample within the range of the data in the input buffer.
cc
  100 continue
cc
c   Find the two input points that bracket the time for the next
c     interpolated sample, then interpolate.
cc
      time = time + deltl
      if (xl.gt.time) Go To 120
  110 continue
      iy=iy+2
      if (iy.GT.nxy) then
         if(j.lt.nreads) go to 200
         if(ictype.ne.wFDIC) go to 200
         if(m0.eq.ndone) go to 200
         ndata=m0
         go to 7012
      endif
      x0 = xl
      y0 = yl
      xl = xy(iy-1)
      yl = xy(iy)
      If (x0.eq.xl) then
         Go to 110
      endif
      IF (xl.gt.time) Go To 120
      Go To 110
  120 continue
      v = (time-x0)*(yl-y0)/(xl-x0) + y0
cc
c   If using interpolation without instrument correction, move
c     interpolated value to output buffer.
c   Otherwise, move value to the intermediate buffer.
```

```
cc
      go to label (7001,7002)
 7001 continue
         m2=m2+1
         acc(m2)= v
         if(m2.eq.lacc)
     1    call HIOUT(luser,iddout,ntot,m2,lacc,acc,acc,
     2    tfirst,ictype,period,sdamp,sps2,fc,fz)
         go to 100
 7002 continue
         m0 = m0+1
         buf(m0) = v
cc
c   If using time domain instrument correction algorithm --
c
c   a) Instrument correct, filter, and decimate by applying
c      convolution operator at every ndense-th point, beginning
c      with point 1.  Load results into the output buffer.
cc
      go to 1blic(7011,7012)
 7011 continue
      nweed = nweed+1
      If (nweed .GT. ndense) nweed = 1
      If (nweed .NE. 1 ) Go To 130
      m2 = m2+1
      If (m2 .LT. 1) nweed = 0
      If (m2 .LT. 1) Go To 130
      acc(m2) = Oprtr(m0-nwts, nwts, sdamp, w0, an.bn,cn, buf)
cc
c   b) Whenever the output buffer is full, move buffer's
c      contents to the output file.
cc
      if(m2.eq.lacc)
     1    call HIOUT(luser,iddout,ntot,m2,lacc,acc,acc,
     2    tfirst,ictype,period,sdamp,sps2,fc,fz)
cc
c   c) Whenever the intermediate buffer is full, shift the last
c      NWTS2 values in the buffer to beginning of buffer.
cc
  130 continue
      If (m0 .eq. lbuf) then
         n=lbuf - nwts2
         Do 131 i = 1, nwts2
  131    Buf(i) = buf(n + i)
         m0   = nwts2
      endif
      go to 100
cc
c   If using frequency domain instrument correction, wait
c   until the intermediate buffer is full, then --
c
c   a) apply correction to as much of the data as is now
c      available in the intermediate buffer.
```

```
subroutine ACALC
program HIFRIC

cc
 7012 continue
      if(m0 .ne.ndata) go to 100
      call AFDIC(buf,ndata,ndone,lbuf,deltl,period,sdamp,fc,fz)
cc
c   b) Decimate and, whenever the output buffer is full, write
c       to output file.
cc
      do 150 i=nstart,ndone,ndense
      m2=m2+1
      acc(m2)=buf(i)
      if(m2.eq.lacc)
     1   call HIOUT(luser,iddout,ntot,m2,lacc,acc,acc,
     2   tfirst,ictype,period,sdamp,sps2,fc,fz)
  150 continue
      nstart=nstart + ndense*(1+ (ndone-nstart)/ndense) - ndone
cc
c   c) Shift the data at the end of the buffer that hasn't been
c       completely processed yet to the beginning of the buffer.
cc
      if(ndone.eq.ndata) go to 200
      do 151 i=ndone+1,ndata
  151 buf(i-ndone)=buf(i)
      m0= ndata-ndone
      ndone=0
cc
c   End of loops.
cc
      go to 100
  200 Continue
cc
c   Fill out last data block and header blocks in the output file.
c   Report min, max ,etc. values to user.
cc
      call HIOUT(luser,iddout,ntot,m2,lacc,acc,acc,
     1           tfirst,ictype,period,sdamp,sps2,fc,fz)
ccccccccccccccccccccc (end of ACALC) ccccccccccccccccccccccccccccc
       END
```

```
      Subroutine SetWts
     1 (n, ksegs, ntype, delt, an, bn, cn)
      Double Precision delt, bn(n+1), cn(n+1), an(n+1)

      Double Precision pi, pi2, a, b
cc
c   SETWTS.ftn:
c   Routines for approximation of first and second derivatives
c   by difference operators.
c
c   Written by Michael Raugh of USGS in 1981.
c
c  Ntype = 1 (analytic)  Prepares Fourier coefficients for expansion
c   of   1, x, x**2   with cosine bells for  a < x < b < pi.  Anti-
c   alias filter (i.e. null frequency response) for   b < x < pi.
c  Ntype = 2 (empiric) Prepares weights for instrument correction
c   convolution operator
cc

      pi = 3.1415926536
      pi2 = pi**2

      a =    pi/ksegs
      b = 2.*pi/ksegs

      an(1) = 1.
      bn(1) = 0.
      cn(1) = 0.

      Do 10 i = 1,n
        an(i+1) =-1/pi*(
     1  S1Cos(i,0,a)+S1Cos(i,0,b)
     2       -0.5*(S1Cos(i+ksegs,a,b)+S1Cos(i-ksegs,a,b)))

        bn(i+1) = 1/pi*(
     1  Sx1Sin(i,0,a)+Sx1Sin(i,0,b)
     2       -0.5*(Sx1Sin(i+ksegs,a,b)+Sx1Sin(i-ksegs,a,b)))

        cn(i+1) =-1/pi*(
     1  Sx2Cos(i,0,a)+Sx2Cos(i,0,b)
     2       -0.5*(Sx2Cos(i+ksegs,a,b)+Sx2Cos(i-ksegs,a,b)))

C  Special additional term for coefficient of Cos(Ksegs*x):
      If (i .NE. ksegs) Go To 4
         an(i+1) = an(i+1) + 0.5/ksegs
         cn(i+1) = cn(i+1) + (7./6.)*pi2/ksegs**3

4     If (ntype .EQ. 1) Go To 10
C  Adjust Fourier coefficients for use as convolution weights
C   in instrument correction operator
      an(i+1) = - 0.5*an(i+1)
      bn(i+1) =   0.5*bn(i+1)/delt
```

```
subroutine SETWTS
program HIFRIC

      cn(i+1) =   0.5*cn(i+1)/delt**2
      an(1) = an(1)-2.*an(i+1)
      cn(1) = cn(1)-2.*cn(i+1)
10    Continue

      Return
      End

      Double Precision Function SlCos (n, a, b)
      Double Precision a, b, xn
      SlCos = snl(n, a, b)
      Return
      End

      Double Precision Function SxlSin (n, a, b)
      Double Precision a, b, xn
      SxlSin = sn2(n, a, b) - cnl(n, a, b)
      Return
      End

      Double Precision Function Sx2Cos (n, a, b)
      Double Precision a, b, xn
      Sx2Cos = cn2(n, a, b) + sn3(n, a, b)
      Return
      End

      Double Precision Function cnl (n, a, b)
      Double Precision a, b, xn
      cnl = 0
      If (n .EQ.0) Return
      xn = n
      cnl =-(a*dcos(n*a)-b*dcos(n*b))/xn
      Return
      End

      Double Precision Function cn2 (n, a, b)
      Double Precision a, b, xn
      cn2 = 0
      If (n .EQ.0) Return
      xn = n**2
      cn2 =-2.*(a*dcos(n*a)-b*dcos(n*b))/xn
      Return
      End

      Double Precision Function snl (n, a, b)
      Double Precision a, b, xn
      snl = 0
      If (n .EQ.0) Return
      xn = n
      snl =-(dsin(n*a)-dsin(n*b))/xn
      Return
      End
```

```
Double Precision Function sn2 (n, a, b)
Double Precision a, b, xn
sn2 = 0
If (n .EQ.0) Return
xn = n**2
sn2 =-(dsin(n*a)-dsin(n*b))/xn
Return
End

Double Precision Function sn3 (n, a, b)
Double Precision a, b, xn
sn3 = 0
If (n .EQ.0) Return
xn = n
sn3 =-((xn**2*a**2-2.)*dsin(n*a)-(xn**2*b**2-2.)*dsin(n*b))/xn**3
Return
End
```

C-26
subroutine OPRTR
program HIFRIC

```
      Function  Oprtr(i, nwts, sdamp, w, an. bn, cn, data)
      dimension data(1)
      Double Precision delt, an(1), bn(1), cn(1)

      Double Precision asym, sym,  d0,d1,d2

      d0 = an(1)*data(i)
      d1 = 0
      d2 = cn(1)*data(i)

      Do 10 j = 1, nwts
      ipj = i+j
      imj = i-j
      sym  = data(ipj) + data(imj)
      asym = data(ipj) - data(imj)
      d0 = d0 + an(j+1)*sym
      d1 = d1 + bn(j+1)*asym
      d2 = d2 + cn(j+1)*sym
10    Continue

      Oprtr = d0 + 2.0*sdamp/w*d1 + d2/w**2

      Return
      END
```

```
      subroutine afdic  (x,ndata.ndone,lx,delt,tins,h,fh1,fh2)
      dimension x(lx)
cc
c   AFDIC applies Bill Joyner's frequency-domain instrument correcting
c         and anti-alias filtering algorithm to acceleration data.
c   WARNING:  Bill chose the NLAP value (see below) as something very
c        much longer than the time it would take the instrument
c        response from a single pulse to decay to (close to) zero.  If
c        very different instrument or filter characteristics are given
c        than those that are usually given, the NLAP value may not be
c        appropriate.  Until more analysis and experiments are
c        performed with this algorithm, Bill recommends the following.
c            fh2  <  1.0/(2*delt)
c            fh1  <= fh2/2
c        and tins <  1.0
c
c   On entry --
c    x()      contains acceleration time series data.
c    ndata =  number of values in x().
c             if ndata < lx, then ***
c             Excess beyond NDATA will be filled with zeros.
c    ndone =  location of the last x-value to have been completely
c             processed during a previous call to FDIC.
c             NDONE must= -1 in the first call to FDIC.
c             NDONE will be reset during the current call. Caller
c             may dispose of x(1 through NDONE), shift the uncompleted
c             x-values to the beginning of x(), reset NDONE, then
c             add more x-values into the end of x() for processing
c             in a subsequent call to FDIC.
c    lx     = length of the array x().  Must be NDONE+ 2 + (an even
c             multiple of) 1024.
c    delt   = time interval between x-values, in seconds, usually 1/600.
c    tins   = instrument period, in seconds, usually about 0.06.
c    h      = instrument damping as fraction of critical damping,
c             usually about 0.6
c    fh1    = beginning-of-taper frequency for cosine taper in the high
c             cut filter, hz, usually 50.
c    fh2    = end-of-taper frequency for cosine taper, hz, usually 100.
c
c   On return --
c    x(1 to ndone)    contains intrument corrected time series data.
c    ndone =  location of the last x-value completely processed during
c             the current call to FDIC.
cc
c   Use "overlap-add method" for fitting separately filtered
c        segments of the data back together.  See section 3.8,
c        pages 110 to 113 in 'Digital Signal Processing' by
c        A.V. Oppenheim and R.W. Schafer; Prentice-Hall,1975.
c
c   Illustration of "overlap-add" method:
c
c   1) Divide x() into equal-lengthed segments:
c
```

```
subroutine AFDIC
program HIFRIC

c              last seg><   current segment   >< next segment
c     x():     ...-----=======================================-...
c     length:            <      nxseg          >< nzeros     >
c       "          <nlap><nlap>< nxseg-nlap   ><nlap><nlap><2>
c     name:        < A  >< B  ><        C      >< D >< E >
c
c   2) Before transforming each segment to the frequency domain,
c      save x-values from the nzeros area of the next segment
c      in XSAVE(), then set the area to 0.0.
c
c   3) Transform B,C,D,E area to frequency domain, instrument
c      correct and filter, then transform back to time domain.
c      Resulting sequence represents one cycle of a periodic sequence
c      and is longer than the original NXSEG points, extending into
c      the NZEROS area from both directions.
c
c   4) Add results in overlapping areas after transforming back to
c      the time domain.
c                A=A+E   B=B+ENDLAP   C=C        D is saved in ENDLAP
c                                                to be added into B
c                                                area of the next seg.
c
c   5) Restore XSAVE back into the NZEROS area, then repeat 2) through
c      5) for the next segment.
cc
c   Sizes used in the overlap method are defined here although NTOT2
c   and NLAP may become input parameters someday.
c     nxseg  =number of x-values in each segment
c     nzeros =number of trailing zeros appended to the x-values
c             before filtering
c     nlap   =length of overlap extending on each side of
c             the NXSEG values after filtering.
c             NZEROS includes space for 2*NLAP (unlike the 1*NLAP
c             used in the example in the textbook cited) because
c             the instrument response function has non-zero values
c             on both sides of zero.
c     ntot2  =number of real, time-domain, values that can be
c             accepted by UFORK.
c     nf+1   =number of complex, frequency-domain, values returned
c             by UFORK.  The first pair are for zero-frequency.
cc
      parameter (m=9)
      parameter (nf=2**m)
      parameter (ntot2=2*nf)
      parameter (nlap=128)
      parameter (n2lap=nlap*2)
      parameter (nzeros=n2lap+2)
      parameter (nxseg =ntot2-n2lap)
      logical first,last
cc
c   Save info. between calls --
cc
      common /svfdic/ init,last,fun,nh1,nh2,
```

```
     1       xsave(nzeros),endlap(nlap)
cc
      if(ndone.ge.0) then
         next0=ndone+nlap
         if(init.ne.-1234) call woe(0)
         if(last) call woe(0)
         first=.false.
      else
         ndone=0
         first=.true.
         last=.false.
         next0=0
         init=-1234
         do 302 i=1,nzeros
  302    xsave(i)=x(i)
         do 303 i=1,nlap
  303    endlap(i)=0.0
         t=float(ntot2)*delt
         fun=1.0/t
         nh1=ifix(fh1*t)
         nh2=ifix(fh2*t)
         if(nh2.gt.nf) call woe(0)
      endif
      nsegs=(ndata-ndone)/nxseg
      if(ndata.lt.lx) then
         last=.true.
         nsegs=1+(ndata-ndone-1)/nxseg
         do 301 i=ndata+1,lx
  301    x(i)=0.0
         if(ndata.le.next0) then
c           *** or is there stuff left in endlap(?) ***
            ndone=ndata
            return
         endif
      endif
      nn= next0 + nsegs*nxseg + nzeros
      if(nn.gt.lx) call woe(0)
      if(ndata.gt.lx) call woe(0)
cc
c   Loop for each segment of x, x(now0+1) through x(next0)
cc
      DO 102 iseg=1,nsegs
      now0=next0
      next0=next0+nxseg
cc
c   Save the part of x in the next segment that needs to be
c     zeroed out for the overlap method.  Also replace
c     saved x from the last segment.
cc
      do 103 i=1,nzeros
      x(now0+i) =xsave(i)
      xsave(i)  =x(next0+i)
  103 x(next0+i)=0.0
```

```
C-30
subroutine AFDIC
program HIFRIC

cc
c   Transform current segment of x along with its trailing zeros
c     from time to frequency.  Odd locations in the returned x
c     are real, even locations are imaginary.
cc
      CALL vfork(x(now0+1),M)
cc
c   Instrument correct.
cc
      do 3 i=1,nh2
      j=2*i+1
      f=fun*float(i)
      fr=1.0-f*f*tins*tins
      fi=2.0*h*f*tins
      TEMP1=FR*X(now0+J)-FI*X(now0+J+1)
      TEMP2=FR*X(now0+J+1)+FI*X(now0+J)
      X(now0+J)=TEMP1
      X(now0+J+1)=TEMP2
cc
c   Apply cosine taper from frequency = fh1 to fh2.
cc
      if(i.le.nh1) go to 3
      factor=0.5*(1.0+cos(3.14159*(f-fh1)/(fh2-fh1)))
      X(now0+J)=FACTOR*X(now0+J)
      X(now0+J+1)=FACTOR*X(now0+J+1)
    3 continue
cc
c   filter gain =0.0 for frequencies above fh2.
cc
      if(nh2.ge.nf) go to 5
      nn=1+2*(nh2+1)
      do 4 i=nn,ntot2+2
    4 X(now0+i)=0.0
    5 continue
cc
c   transform back to time.
cc
      CALL ufork(x(now0+1),M)
cc
c   Overlap these results with results from the last segment.
cc
      do 304 i=1,nlap
      ii=now0+i
      x(ii)=x(ii) + endlap(i)
  304 endlap(i)=x(next0+i)
      if(.not. first) then
         nn=now0 -nlap
         mm=next0+nlap
         do 305 i=1,nlap
         ii=nn+i
  305    x(ii)=x(ii)+x(mm+i)
      endif
  102 CONTINUE
```

```
      ndone =next0-nlap
cc
c   If the last segment processed is the last in the time series,
c     set x() to zero beyond ndata and set ndone=ndata.
cc
      if(last) then
         if(next0.le.ndata) call woe(0) !if so, I'm confused.
         ndone=ndata
         do 310 i=ndata+1,lx
  310    x(i)=0.0
      endif
      return
ccccccccccccccccccc (end of afdic) ccccccccccccccccccccccccccccccc
      end
```

include files
program CORAVD

```
c --- begin sarray.inc ---
      dimension array(larray)
c --- end   sarray.inc ---
```

```
c --- begin varray.inc ---
      virtual array(larray)
c --- end   varray.inc ---
```

```
      subroutine llsqf(iddin,intype ndblks,ilst,
     1                 array,larray,
     2                 dr100f,delt,tbegin.fitbeg,fitend,taper,
     2                 nreads,slope,y0)
      include 'array.inc'
      double precision delt
cc
c   LLSQF
c   Calculate the coefficients Y0 and SLOPE for the line
c        Y = Y0 +SLOPE*X
c   which best fits the integral (=velocity) of the points on the
c   input file in the least squares sense.
c   If intype=2, then fit the input data itself, don't integrate
c   first.
c
c   on entry --
c     iddin   =blocked binary file identifier for the input file.
c     intype  =1 if the input data is acceleration,
c             =2 if the input data is velocity.
c     ndblks  = number of data blocks on the input file.
c     ilst    = index of the last data item in the last data block.
c     array() = array into which the data will be read.
c     larray  = length of array
c     dr100f  = DR100 data conversion factor, = 0.0 if the data comes
c                 from an AGRAM program rather than from DR100.
c     delt    = time increment, in seconds, between data samples.
c     tbegin  =time corresponding to the first point in the data file.
c     fitbeg  =time of the first point to inclue in the fit.
c     fitend  =time of the last point to include in the fit.
c              *** tbegin doesn't include the syncronizing time offset
c                  that may have been given in the header.  Will fitbeg
c                  and fitend??  Assume not, for now. ***
c     taper   =portion of the fitrange in which to apply a cosine
c               taper.  (0.0 <= taper >=0.5)
c   on return --
c     nreads  =1 if all the input data is in array, so subsequent
c              subroutines need'nt reread it.
c     slope   =slope of the llsqf line.
c     y0      =intercept of the llsqf line.
cc
      double precision x,y, time0,tabuf
      double precision d,sumx,sumxx,sumy,sumxy
      include 'files.inc/nolist'
c +++ integer*4 npts
cc
      pi    =3.1415926536
      sumx  =0.0
      sumxx =0.0
      sumy  =0.0
      sumxy =0.0
cc
      nbab   = larray/YFBLK
      la     = YFBLK*nbab
```

```
subroutine LLSQF
program CORAVD

      tabuf  = delt * la
      time0  = tbegin -delt
      y      = 0.0
      alast  = 0.0
      hdt    = 0.5*delt
      t1     = fitbeg-hdt
      t2     = fitend+hdt
      npts   = 0
      points =0.0
      ttaper =taper*(fitend-fitbeg)
      tapere =fitbeg + ttaper
      taperb =fitend - ttaper
      assign 20 to JMP
      if(taper.gt.0.0 .and. taper.le.0.5) assign 21 to JMP
      assign 302 to JUMP
      if(intype.eq.2) assign 301 to JUMP
cc
c   Loop for each bufferful of data on the input file, then
c   Loop for each data point in the buffer.
cc
      nreads = 1 + (ndblks-1)/nbab
      if(nreads.le.0) call woe(0)
      nlast  = ilst + YFBLK*(ndblks-1 -(nreads-1)*nbab)
      n =la
      if(nbab.ge.ndblks) n=ndblks*YFBLK
      do 100 ibuf=1,nreads
      call ready(iddin,array,larray, n ,dr100f)
      if(ibuf.eq.nreads) n=nlast
      do 101 i=1,n
      go to JUMP,(301,302)
  301 continue
         y=array(i)
         go to 303
  302 continue
         y= y +  hdt*(alast + array(i))
         alast =array(i)
c +++    would be faster to get space for acc(0) instead of using alast
  303 continue
      x= delt*float(i) + time0
      if(x.lt.t1) go to 101
      if(x.gt.t2) go to 102
cc
c   a) taper the effect of the points near the end of the fit range.
cc
      go to JMP,(20,21,22,23)
   21 continue
      wt=(x-fitbeg)/ttaper
      if(x.le.tapere) go to 103
      assign 22 to JMP
   22 continue
      if(x.lt.taperb) go to 20
      assign 23 to JMP
   23 continue
```

```
      wt=(fitend-x)/ttaper
  103 continue
      wt=0.5 *(1.0 - cos(pi*wt))
      points= points +wt
      sumx  = sumx  + x     *wt
      sumxx = sumxx + x*x   *wt
      sumy  = sumy  + y     *wt
      sumxy = sumxy + y*x   *wt
      go to 101
cc
c   b) Weight the points in the middle of the fit range evenly.
cc
   20 continue
      npts=npts + 1
      sumx  = sumx  + x
      sumxx = sumxx + x*x
      sumy  = sumy  + y
      sumxy = sumxy + y*x
  101 continue
      time0 = time0 + tabuf
  100 continue
  102 continue
cc
c   end loop.
cc
      points = points + float(npts)
      d    = points*sumxx   - sumx*sumx
      y0   = (sumy*sumxx    - sumx*sumxy)/d
      slope= (points*sumxy  - sumx*sumy)/d
      return
ccccccccccccccccccc (end of llsqf) ccccccccccccccccccccccccccc
      end
```

```
subroutine AVD
program CORAVD

      subroutine AVD (
     1       acc.vel,dis,lla,llv,lld,
     2       dr100f,tbegin,delt,slope,y0,
     2       intype,iddin,ndblks,ilst,inmem,
     3       iddout,ndbout,ilout,
     3       vbeg,vend,vmin,vmax,
     4       tbeg,     tmin,tmax)
      dimension acc(lla), vel(llv), dis(lld)
      double precision  delt
      parameter nout=3
      dimension iddout(nout),ndbout(nout),ilout(nout)
      dimension vbeg(nout),tbeg(nout),vend(nout)
      dimension vmin(nout),vmax(nout), tmin(nout),tmax(nout)
cc
c   AVD:
c   Calculate velocity and displacement from acceleration.
c   If requested, subtract a line (the llsqf of some portion of the
c     velocity) from the velocity and a constant (the slope of the
c     llsqf line) from the acceleration.
c
c   AVD if an alternative version of FLTAVD that does no filtering,
c      and will NOT truncate the data even if it won't fit in
c      working storage space all at once.
c
c   On entry --
c     acc()       = i/o buffer for acceleration data.
c     lla         = length of acc().
c     vel()       = i/o buffer for velocity data.
c     llv         = length of vel().
c     dis()       = output buffer for displacement data.
c     lld         = length of dis().
c     dr100f      = DR100 data conversion factor. (=0.0 if the data
c                   comes from an AGRAM program rather than from
c                   DR100.)
c     tbegin      = time corresponding to the first sample in the
c                   data file.
c     delt        = time increment between data samples, in seconds.
c     slope       = slope of the llsqf to velocity.
c     y0          = intercept  "
c     intype      = 1 if input file contains acceleration,
c                 = 2 if input file contains velocity.
c     iddin       = blocked binary file identifier for the input file.
c     ndblks      = number of data blocks on the input file.
c     ilst        = index of the last data sample in the last data
c                   block on the input file.
c     inmem       = 1 if the data has already been read into array and
c                     it all fits.
c     iddout()    = blocked binary file identifier for the three output
c                   files.
c                   (1) => acceleration file,
c                   (2) => velocity,
c                   (3) => displacement.
c
```

```
c   on return --
c     ndbout(),ilout()  = ndblks and ilst for the three output files.
c     vbeg(),vend(),vmin(),vmax(),tbeg(),tmin(), and tmax()
c                   have been given values which should be inserted
c                   into the header blocks of the output files.
cc
      include 'files.inc/nolist'
      double precision dpv,dpd, hdt
      double precision a0time, v0time, d0time
      double precision tabuf,  tvbuf,  tdbuf
      parameter huge = 1.0e30
cc
c   Worry about buffer siZes
cc
      ld=(lld/YFBLK)*YFBLK
      lv=(llv/YFBLK)*YFBLK
      nbab = lla/YFBLK
      la=nbab*YFBLK
cc
c   initialize
cc
      do 10 j=1,nout
      vbeg(j)=0.0
      vend(j)=0.0
      vmin(j)= huge
      vmax(j)=-huge
      tmin(j)=0.0
      tmax(j)=0.0
      ndbout(j)=ndblks
      ilout(j) =ilst
      tbeg(j)  =tbegin
   10 continue
      a0time =-delt     +tbegin
      v0time = a0time
      d0time = a0time
      tabuf  = delt*la
      tvbuf  = delt*lv
      tdbuf  = delt*ld
      idacc  =iddout(1)
      idvel  =iddout(2)
      iddis  =iddout(3)
      iv=0
      id=0
      dpv =0.0
      dpd =0.0
      hdt=0.5*delt
      alast=0.0
      vlast=0.0
      assign 302 to JUMP
      if(intype.eq.2) assign 301 to JUMP
cc
c   Loop for each bufferful of data on the input file, then
c   Loop for each data point in the buffer.
```

```
subroutine AVD
program CORAVD

cc
      nreads = 1 + (ndblks-1)/nbab
      if(nreads.le.0) call woe(0)
      nlast  = ilst + YFBLK*(ndblks-1 -(nreads-1)*nbab)
      n =1a
      if(nbab.ge.ndblks) n=ndblks*YFBLK
      do 100 ibuf=1.nreads
      if(inmem.ne.1) call ready(iddin,acc,11a, n ,dr100f)
      if(ibuf.eq.nreads) n=nlast
      do 101 i=1,n
cc
c   Integrate input acceleration for velocity,
c     or, if input data is velocity, just move the data from
c     input array to vel array.
c   Then subtract linear correction from velocity and
c     subtract a constant, =slope of the line, from acceleration.
cc
      iv=iv+1
      go to JUMP,(301,302)
  301 continue
         dpv= acc(i)
         go to 303
  302 continue
         dpv=dpv + (acc(i) + alast) *hdt
         alast=acc(i)
         acc(i)=acc(i)-slope
  303 continue
      vel(iv)=dpv
     1        - y0 -slope*(v0time + float(iv)*delt)
cc
c   integrate corrected velocity for displacement.
cc
      id=id+1
      dpd =dpd + (vel(iv) + vlast)*hdt
      vlast=vel(iv)
      dis(id)=dpd
cc
c   find min and max values of each curve
cc
      if(acc(i) .gt.vmax(1)) then
         vmax(1)=acc(i)
         tmax(1)=delt*float(i) + a0time
      endif
      if(acc(i).lt.vmin(1)) then
         vmin(1)=acc(i)
         tmin(1)=delt*float(i) + a0time
      endif
      if(vel(iv) .gt.vmax(2)) then
         vmax(2)=vel(iv)
         tmax(2)=delt*float(iv) + v0time
      endif
      if(vel(iv).lt.vmin(2)) then
         vmin(2)=vel(iv)
```

```
         tmin(2)=delt*float(iv) + v0time
       endif
       if(dis(id) .gt.vmax(3)) then
          vmax(3)=dis(id)
          tmax(3)=delt*float(id) + d0time
       endif
       if(dis(id).lt.vmin(3)) then
          vmin(3)=dis(id)
          tmin(3)=delt*float(id) + d0time
       endif
cc
c   Write to output files whenever buffers are full
cc
       if(iv .eq. lv) then
          call bio(idvel,ZWRITE,zwaita,ZNEXTB,zf,vel,lv,ntrans)
          if(ntrans.lt.lv) call woe(0)
          if(v0time.lt.tvbuf-delt) vbeg(2) = vel(1)
          iv=0
          v0time = v0time + tvbuf
       endif
       if(id.eq.ld) then
          call bio(iddis,ZWRITE,zwaita,ZNEXTB,zf,dis,ld,ntrans)
          if(ntrans.lt.ld) call woe(0)
          if(d0time.lt.tdbuf-delt) vbeg(3) = dis(1)
          id=0
          d0time = d0time + tdbuf
       endif
  101  continue
       if(ibuf.lt.nreads .and. intype.eq.1) then
          call bio(idacc,ZWRITE,zwaita,ZNEXTB,zf,acc,la,ntrans)
          if(ntrans.lt.la) call woe(0)
          if(ibuf.eq.1) vbeg(1)=acc(1)
          a0time =a0time + tabuf
       endif
  100  continue
cc
c   Write out the last bufferful for each data file
cc
       vend(1)=acc(n)
       vend(2)=vel(iv)
       vend(3)=dis(id)
       if(intype.eq.1)
      lcall filler(idacc,acc,nlast,la)
       call filler(idvel,vel iv,lv)
       call filler(iddis,dis,id.ld)
       return
ccccccccccccccccccc (end of AVD) ccccccccccccccccccccccc
       end
       subroutine filler (idbbf,array,last,lend)
       dimension array(lend)
       include 'files.inc/nolist'
cc
       if(last.eq.0) return
```

```
subroutine FILLER/AVD
program CORAVD

      l= (l + (last-l)/YFBLK)*YFBLK
      if(l.gt.lend) call woe(0)
      if(l.eq.last) go to 11
      rnone =rundef(idbbf)
      DO 10 I= last+l,L
   10 ARRAY(I)=rnone
   11 continue
      call bio(idbbf,zwrite.zwaita,ZNEXTB.zf.array,l,ntrans)
      return
ccccccccccccccccccc (end of filler/AVD) ccccccccccccccccccc
      end
```

```
      subroutine FLTAVD (luser,kfiltr,fcornr,framp,noptn,option,ixtend,
     1       array,larray,
     2       dr100f,tbegin,delt,slope,y0,
     2       intype,iddin,ndblks,ilst,inmem,
     3       iddout,ndbout,ilout,
     3       vbeg,vend,vmin,vmax,
     4       tbeg,tend,tmin,tmax)
c +++ integer*4 larray
      include 'array.inc'
      double precision delt
      dimension kfiltr(2),fcornr(2),framp(2)
      dimension noptn(2),option(2),ixtend(2)
      parameter nout=3
      dimension iddout(nout),ndbout(nout), ilout(nout)
      dimension vbeg(nout),tbeg(nout),vend(nout)
      dimension tend(nout)
      dimension vmin(nout),vmax(nout), tmin(nout),tmax(nout)
cc
c   FLTAVD:
c   Calculate velocity and displacement from acceleration.
c   If requested, subtract a line (the llsqf of some portion of the
c     velocity) from the velocity and a constant (the slope of the
c     llsqf line) from the acceleration.
c   If requested, filter acceleration and velocity.
c
c   FLTAVD is an alternative version of AVD that can also apply a
c     filter to the accleration and velocity.  Unlike AVD, however,
c     FLTAVD will truncate a time series if all the data and necessary
c     scratch space will not fit in array all at once.
cc
c
c   on entry --
c     luser       = LUN for diagnostic and run messages.
c     kfiltr()    = filter parameters, see subroutine filter.
c     fcornr()      "
c     framp()       "
c     noptn()       "
c     option()      "
c     ixtend()      "
c     array()     = array to contain the data to be manipulated
c                   (in locations 1 through ndata) plus working
c                   space (in locations ndata+1 through larray).
c                   The amount of work space required depends on
c                   the type of filter used.
c     larray      = length of array.
c     dr100f      = DR100 data conversion factor. (=0.0 if the data
c                   comes from another  AGRAM program rather than from
c                   DR100.)
c     tbegin      = time corresponding to the first sample in the
c                   data file.
c     delt        = time increment between data samples, in seconds.
c     slope       = slope of the llsqf to velocity.
c     y0          = intercept  "
```

```
subroutine FLTAVD
program CORAVD

c     intype      = 1 if input file contains acceleration,
c                 = 2 if input file contains velocity.
c     iddin       = blocked binary file identifier for the input file.
c     ndblks      = number of data blocks on the input file.
c     ilst        = index of the last data sample in the last data
c                   block on the input file.
c     inmem       = 1 if the data has already been read into array and
c                     it all fits.
c     iddout()    = blocked binary file identifier for the three output
c                   files.
c                   (1) => acceleration file,
c                   (2) => velocity,
c                   (3) => displacement.
c
c   on return --
c     ndbout(),ilout()  = ndblks and ilst for the three output files.
c                   Note that the output files will be shorter than the
c                   input file if the data were truncated here or in
c                   subroutine filter.
c     vbeg(),vend(),vmin(),vmax(),tbeg(),tend(),tmin(), and tmax()
c                   have been given values which should be inserted
c                   into the header blocks of the output files.
cc
      include 'files.inc/nolist'
cc
      double precision dpv,dpd
c +++ integer*4 npts, nn, nread, need  nptwas
cc
      do 10 j=1,nout
      vbeg(j)=0.0
      vend(j)=0.0
      vmin(j)= huge
      vmax(j)=-huge
      tmin(j)=0.0
      tmax(j)=0.0
      ndbout(j)=ndblks
      ilout(j) =ilst
      tbeg(j)  =tbegin
   10 continue
      a0time =-delt     +tbegin
      v0time = a0time
      d0time = a0time
      idacc  =iddout(1)
      idvel  =iddout(2)
      iddis  =iddout(3)
      dpv =0.0
      dpd =0.0
      hdt=0.5*delt
      alast=0.0
      vlast=0.0
cc
c   is array big enough for the data?
cc
```

```
      nread=ndblks
      nread=nread *YFBLK
      npts= nread -YFBLK + ilst
      nptwas=npts
      if(nread.gt.larray) then
         nread= (larray/YFBLK) *YFBLK
         call noroom(luser,0,npts,nread,delt)
      endif
cc
c   read input data into array unless subroutine llsqf has
c     already put it there.  Data is either acceleration or
c     velocity.
cc
      if(inmem.ne.1) call ready (iddin,array,larray,nread,dr100f)
cc
c   Integrate input acceleration for velocity.
cc
      if (intype.eq.1) then
         do 110 nn=1,npts
         dpv=dpv + (array(nn) + alast) *hdt
         alast = array(nn)
         array(nn)=dpv
  110    continue
      endif
cc
c   Subtract linear correction from velocity.
cc
      if (slope.ne.0.0 .or. y0 .ne.0.0) then
         do 111 nn=1,npts
         array(nn)=array(nn)
     1        - y0 -slope*(v0time + float(nn)*delt)
  111    continue
      endif
cc
c   filter baseline corrected velocity
cc
      if(kfiltr(1).ne.0 .or.kfiltr(2).ne.0)
     1   call filter(luser,2,array npts,larray delt,
     2               kfiltr,fcornr,framp,noptn,option,ixtend)
cc
c   write corrected, filtered velocity
cc
      call output(idvel,array,nread,npts,nptwas v0time,delt,
     1            vmax(2),tmax(2),  vmin(2),tmin(2),
     2            vbeg(2)           vend(2),tend(2),
     3            ndbout(2),ilout(2) )

cc
c   integrate corrected, filtered velocity for displacement.
cc
      vlast=0.0
      do 120 nn=1 npts
      dpd =dpd + (array(nn) + vlast)*hdt
```

```
subroutine FLTAVD
program CORAVD

      vlast= array(nn)
      array(nn)=dpd
  120 continue
cc
c   write displacement
cc
      call output(iddis.array,nread,npts,nptwas,d0time,delt,
     1            vmax(3),tmax(3),  vmin(3),tmin(3),
     2            vbeg(3),          vend(3),tend(3),
     3            ndbout(3).ilout(3) )
cc
c   reread input acceleration.
c +++ Or would it be better to use two arrays so we don't have to
c     read the acceleration twice?? +++
cc
      if(intype .ne.1) go to 200
      call vbio(iddin,ZPOS, zwaita,ZDATA1, ZF ,array,yfblk ntrans)
      call ready (iddin,array,larray,nread,dr100f)
cc
c  subtract a constant, =slope of the line, from acceleration
cc
      do 130 nn=1,npts
  130 array(nn)=array(nn)-slope
cc
c   filter corrected acceleration
cc
      if(kfiltr(1) ne.0 .or.kfiltr(2).ne.0)
     1   call filter(luser,1,array,npts,larray,delt,
     2               kfiltr,fcornr framp,noptn option,ixtend)
cc
c   write filtered, corrected acceleration.
cc
      call output(idacc,array,nread,npts,nptwas a0time,delt,
     1            vmax(1),tmax(1),  vmin(1),tmin(1),
     2            vbeg(1),          vend(1),tend(1),
     3            ndbout(1),ilout(1) )
cc
  200 continue
      return
ccccccccccccccccccc (end of FLTAVD) ccccccccccccccccccccccccccccccc
      end
      subroutine output(idbbf,array,larray,npts,nptwas time0.delt,
     1                  vmax,tmax.vmin.tmin,vbeg,vend,tend,
     2                  ndbout,ilout)
      double precision delt
c +++ integer*4 nread,npts,nptwas
      include 'array.inc'
cc
c   output subroutine for FLTAVD.
cc
      include 'files.inc/nolist'
c +++ integer*4 nn, nwrite
cc
```

```
      nread=larray
      nwrite=nread
      if(npts.lt.nptwas) nwrite=YFBLK*(1 + (npts-1)/YFBLK)
      if(nwrite.gt.npts) then
         rnone=rundef(idbbf)
         do 10 nn = npts+1,nwrite
   10    array(nn)=rnone
      endif
      call vbio(idbbf,ZWRITE.zwaita.ZNEXTB,zf,array,nwrite,ntrans)
      if(ntrans.lt.nwrite) call woe(0)
      if(npts.ne.nptwas) then
         tend =time0 + float(npts)*delt
         ndbout=1+ (npts-1)/YFBLK
         ilout = npts - YFBLK*(ndbout-1)
      endif
      vbeg = array(1)
      vend = array(npts)
      do 111 nn=1,npts
      if(array(nn) .gt.vmax) then
         vmax=array(nn)
         tmax=delt*float(nn) + time0
      endif
      if(array(nn).lt.vmin) then
         vmin=array(nn)
         tmin=delt*float(nn) + time0
      endif
  111 continue
      return
ccccccccccccccccccc (end of output/FLTAVD) ccccccccccccccccccccc
      end
```

```
subroutine FILTER
program CORAVD

      subroutine filter (luser,kind,array,ndata.larray,delt,
     1                   kfiltr,fcornr,framp.noptn,option,ixtend)
      double precision delt
      dimension kfiltr(2),fcornr(2),framp(2)
      dimension noptn(2).option(2),ixtend(2)
      include 'array.inc'
c +++ integer*4 ndata,larray
cc
c   FILTER is called from FLTAVD to filter acceleration and
c     velocity (in 2 separate calls) according to kfiltr option.
cc
c   on entry --
c     luser   = lun for diagnostics and run messages.
c     kind      indicates the type of data in array; its only use is
c               for labeling the run messages.
c               =1=> acceleration,
c               =2=> velocity,
c               =3=> displacement.
c     array() = array containing the data to be filtered (in locations
c               1 through ndata) plus working space (in locations
c               ndata+1 through larray).  The amount of work space
c               required depends on the type of filter used.
c     ndata   = number of data items in array.
c     larray  = length of array.
c     delt    = time increment between data items, in seconds.
c     kfiltr()= filter type
c              =1 => unidirectional, high-pass Butterworth filter,
c               2 => bi "
c               3 => Ormsby filter.
c               4 => FFT filter,
c              -1 => debuggery.
c     fcornr()= corner frequency for the filter, cps.
c     framp() = width of transition band extending from fcornr to
c               frequency at which filter gain =0.0.
c     noptn() = integer option used with some of the filters.
c               For Butterworth filters  noptn = rolloff order which is
c               used rather than framp to specify the width of the
c               transition band.
c     option()= real option used with some of the filters.
c               For Ormsby filter, option = the ormk variable used in
c                  calculating the convolution weights.
c     ixtend()= data padding option needed for most of the filters.
c               The data will be extended to npoints=nearest power of 2
c               for the FFT filter; to half again its original length
c               for the bidirectional Butterworth filter.  For Ormsby
c               filter, the data is not actually padded, but the
c               padding effect is incorporated during the convolution.
c              =1 => extend data with zeros (this may leave a sharp step
c                    between the data and the pad.)
c              =2 => taper a small section at the beginning of the pad
c                    area from the last data point down to zero.
c              =3 => taper a small section at both ends of the data to
c                    meet the zeroed pad area.
```

```
c
c     index for kfiltr,fcornr,framp,noptn.option, and ixtend
c             =1 for low frequency filter (="low cut", ="high-pass")
c             =2 for high frequency filter (="high cut", ="low-pass")
c
c   on return --
c     ndata   may have become smaller than it was on entry if there
c               wasn't enough working storage space.
c     array(1) through array(ndata) contain filtered data.
c     array(ndata+1) through array(larray) contain garbage.
cc
c +++ integer*4 npwr2, nn, ll, npts,lcopy.lwts
      parameter extend=0.5 !*** Bill Joyner has a formula for extend.
cc
      if(kind.eq.1) write(luser,1001)
      if(kind.eq.2) write(luser,1002)
      if(kind.eq.3) write(luser,1003)
cc
c   prepare for a low cut filter (nowf=1),
c     but allow for concurrent high and low cut FFT filter (nowf=3)
c     if the options for both are the same.
cc
      nextf=2
      if(kfiltr(1).eq.0) go to 10
      nowf=1
      kindf=kfiltr(1)
      ipad=ixtend(1)
      if(kindf.eq.4 .and.
     1   kfiltr(2).eq.kfiltr(1) .and.
     2   ixtend(2).eq.ixtend(1) )then
         nowf=3
         nextf=0
      endif
      go to 20
cc
c   prepare for a high cut filter (nowf=2) if it wasn t performed
c     along with the low cut filter.
cc
   10 continue
      if(nextf.ne.2 .or. kfiltr(2).le.0) return
      nextf=0
      nowf=2
      kindf=kfiltr(2)
      ipad=ixtend(2)
cc
c   choose the filter
cc
   20 continue
      if(ipad.le.0) ipad=1
      go to (100,200,300,400), kindf
cc
c   debug pad options
cc
```

```
subroutine FILTER
program CORAVD

      type *, ' filter, debug option'
      npts = ifix(float(ndata)*(1.0+extend))
      if(npts.gt.larray) call shortp(luser,npts,larray)
c --- play with seconds 3.0-5.0. fillin 0.0 to 3 and 5.0to 6.0
      ndata =ifix(200. * 5.0)
      npts = ifix(200. * 6.0)
      ll=    ifix(200. * 3.0)
      call pad(luser,npts, ipad,array,ll,ndata,larray)
      call pad(luser,1   , ipad,array,ll,ndata,larray)
c -----
      ndata =npts
      return
cc
c   unidirectional, high-pass Butterworth filter
c     (from Mike Raugh's PHB program)
cc
  100 continue
      if(nowf ne.1) then
         write(luser,1011)
         call woe(0)
      endif
      write(luser,1010) fcornr(1),noptn(1)
      call UNIhip(array,ndata,fcornr(1),delt noptn(1),larray,0)
      go to 10
cc
c   bidirectional, high-pass Butterworth filter
c     (from Mike Raugh's PHB program)
cc
  200 continue
      if(nowf.ne.1) then
         write(luser,1021)
         call woe(0)
      endif
      write(luser,1020) fcornr(1),noptn(1)
      npts =ndata
         npts = ifix(float(ndata)*(1.0+extend))
         if(npts.gt.larray) call shortp(luser,npts larray)
      ll=1
      call pad(luser,npts, ipad,array,ll,ndata,larray)
      call BIhip(array,npts,dble(fcornr(1)),delt,noptn(1),larray)
      go to 10
cc
c   Ormsby filter
cc
  300 continue
      call ormflt(luser,delt,nowf,ipad,fcornr(nowf),framp(nowf),
     1            option(nowf), array,ndata,larray)
      go to 10
cc
c   FFT filter.  Use Jon Raggett's filter from DRAWSMR.
c   Separate array space into:
c     .data  (1 thru ndata)
c     .pad out data to next power of 2 points (ndata+1 thru nn)
```

```
c     .2 extra cells needed by RFFT (nn+1 and nn+2)
cc
  400 continue
      fl=fcornr(1)
      fh=fcornr(2)
      if(nowf.eq.1) then
         fh=1.0/delt
         write(luser,1040) fl,framp(1)
      else if(nowf.eq.2) then
         fl=0.0
         write(luser.1041) fh,framp(2)
      else
         write(luser,1042)fl, fh,framp(1).framp(2)
      endif
      type *, ' *** WARNING: the FFT filter hasn t been tested yet'
      nn = npwr2(ndata)
      if(nn .gt.larray-2) call noroom(luser,1,nn,larray-2,delt)
      nn=nn+2
      ll=1
      call pad(luser,nn-2,ipad,array,ll,ndata,larray)
      call FFTflt(FL,framp(1),FH,framp(2),delt,
     1            array,nn,larray)
      go to 10
cc
 1001 format(/5x,'Acceleration filtered with')
 1002 format(/5x,'Velocity filtered with')
 1003 format(/5x,'Displacement filtered with')
 1010 format(8x,
     1'unidirectional, high-pass Butterworth filter,'
     2/8x.'corner frequency =', f7.3,
     3   ' cps  and rolloff order =',i2)
 1011 format(
     1' *** Sorry  the unidirectional high cut Butterworth filter'
     2' doesn t work yet. ***')
 1020 format(8x,
     1'bidirectional, high-pass Butterworth filter,'
     2/8x,'corner frequency =', f7.3,
     3   ' cps, and rolloff order =',i2)
 1021 format(
     1' *** Sorry, the bidirectional high cut Butterworth filter'
     2' doesn t work yet. ***')
 1040 format(8x,
     1'low cut FFT filter, corner frequency =', f7.3, ' cps'
     2/8x, 'and rolloff band =',f7.3)
 1041 format(8x,
     1'high cut FFT filter, corner frequency =', f7.3, ' cps'
     2/8x, 'and rolloff band =',f7.3)
 1042 format(8x,
     1'FFT filter, corner frequencies =', 2f7.3, ' cps'
     2/8x  'and rolloff bands =',2f7.3)
ccccccccccccccccccc (end of filter ) ccccccccccccccccccccccc
      end
      subroutine shortp(luser,nold,nnew)
```

```
subroutine FILTER
program CORAVD

      write(luser,1000) nold
      nold=nnew
      return
 1000 format (/
     1' *** WARNING: work space is too small to pad the data to',
     2 i7,' points. ***')
      end
ccccccccccccccccccc (end of shortp/filter) ccccccccccccccccccc
      subroutine noroom(luser,iquit,nold,nnew,delt)
cc
c   iquit=0:  truncate data and warn user,
c         1:  print diagnostic and stop.
cc
      told=delt*float(nold)
      tnew=delt*float(nnew)
      if(iquit.gt.0 .or. nnew.le.200) then
         if(nnew.gt.0) write (luser,1001) tnew.nnew,told.nold
         if(nnew.le.0) write (luser,1000)
         call woe(0)
      else
         write(luser,1002) told,nold,tnew,nnew
         nold=nnew
      endif
      return
cc
 1000 format (/
     1' *** Program needs more work space. ***')
 1001 format (/
     1 ' *** Program needs more work space.'
     2/5x,'Have enough space for', f6.3,'seconds (',i7,' points),'
     3/5x,'need enough space for', f6.3,'seconds (',i7,' points).'
     4  ,' ***')
 1002 format (/
     1' *** WARNING '
     2/5x,'Data and temporary variables won t fit in'
     2, ' the work space available.'
     3/5x,'Data has been truncated from',f7.3,
     4    ' seconds (', i7,' points)'
     5/5x,'to', f7.3, ' seconds (', i7,' points). ***')
ccccccccccccccccccc (end of noroom/filter) ccccccccccccccccccc
      end
      function npwr2(n)
c +++ integer*4 npwr2, n, nm
cc
c     return next largest power of 2.
cc
      p= log(float(n-1))/alog(2.0)
      i= 1 + ifix(p)
   10 continue
      npwr2= 2**i
cc
c   now, in case of precision problems --
cc
```

```
      nm=2**(i-1)
      if(n.gt.nm .and. n.le.npwr2) return
      if(n.le.nm) i=i-1
      if(n.gt.npwr2) i=i+1
      go to 10
ccccccccccccccccccc (end of npwr2/filter) ccccccccccccccccccccc
      end
```

```
subroutine BIHIP
program CORAVD

      subroutine BIhip(array,ndata,fcut,dt,nfs,larray)
      include 'array.inc'
      double precision dt,fcut
cc
c   BIhip:  Bidirectional, high-pass Butterworth recursive filter.
c
c   Written by Keith McCamy of Lamont-Doherty Geophysical Observatory
c                           of Columbia University.
c
c   Installed as subroutine BUTWOR in the NSMDC user library by
c       Jon Fletcher.
c
c   Modified by Mike Raugh in 1981:
c     - working variables are in double precision and are dimensioned
c       as (11) rather than (8).
c     - uses data in a virtual array
c     - wrote messages to lun=5.  (which April took back out again.)
c
c   Modified by April Converse in 1982:
c     - renamed it to BIhip  so it can be distinguished from the BUTWOR
c       subroutine on USERLIB.
c     - made dt double precision since it is so in the calling
c       subroutine.
c     - renamed s() to array() and isiz to larray so array can be
c       declared as a standard or a virtual array, depending on the
c       contents of array.inc.
cc
c   On entry --
c     array()    contains data to be filtered.
c     ndata    = number of data points, array(1) through array(ndata)
c     larray   = length of array.
c     fcut     = corner frequency in cps.
c                This "corner" is the frequency at which the filter
c                gain is down by 6 decibels.
c     dt       = time increment between data points, seconds.
c     nfs      = rolloff order.
c                rolloff = nfs*24 decibels/octave.
c
c   On return --
c     array()    contains filtered data.
cc
c
c was:      SUBROUTINE BUTWOR (S, NDATA, FCUT, TS, NFS)
c was:C+
c was:c    BUTWOR is a high pass Butterworth filter.  It does
c was:c not shift frequency in the time domain (phase = 0).
c was:c
c was:c    Call BUTWOR (s, ndata, fcut, ts, nfs)
c was:c
c was:c            s = the array of data to be filtered
c was:c            ndata = the number of data points
c was:c            fcut = the corner frequency
c was:c            ts = the time interval (dt)
```

```
c was:c             nfs = the order of the rolloff
c was:c
c was:c-
c was:c
c was:      DIMENSION  S(NDATA), F(8, 3), A(8), B(8), C(8)
cc
      Double Precision CS,  PI, TEMP, TS, WCP
      Double Precision F(11,3),A(11),B(11),C(11)

      NFS1=NFS+1
      if(nfs1.gt.11) call woe(0)
      TS=DT
      PI=3.1415926535
      WCP=SIN(FCUT*PI*TS)/COS(FCUT*PI*TS)
      DO 5 K=1.NFS
      CS=COS(FLOAT(2*(K+NFS)-1)*PI/FLOAT(4*NFS))
      A(K)=1./(1.+WCP*WCP-2.*WCP*CS)
      B(K)=2.*(WCP*WCP-1.)*A(K)
    5 C(K)=(1.+WCP*WCP+2.*WCP*CS)*A(K)
C...PERFORM CONVOLUTION IN TWO DIRECTIONS
      DO 30 IJK=1,2
      DO 6 I=1,NFS1
      DO 6 J=1,2
    6 F(I,J)=0.
      DO 10 N=1,NDATA
      F(1.3)=array(N)
      DO 14 I=1,NFS
      TEMP=A(I)*(F(I,3)-2.*F(I,2)+F(I,1))
   14 F(I+1,3)=TEMP-B(I)*F(I+1,2)-C(I)*F(I+1.1)
      DO 16 I=1,NFS1
      DO 16 J=1,2
   16 F(I,J)=F(I,J+1)
   10 array(N)=F(NFS1,3)
      NBY2=INT(FLOAT(NDATA)/2.)
      DO 20 N=1,NBY2
      NUM=NDATA-N+1
      TEMP=array(N)
      array(N)=array(NUM)
   20 array(NUM)=TEMP
   30 CONTINUE

      Return
      End
```

C-54
subroutine RFFT
library

```
      SUBROUTINE RFFT(X,N)
c
c+
c
c Subroutine RFFT.
c
c  RFFT computes the finite Fourir transform of a real
c array.  The Fourier coefficients are returned as a complex
c array.  RFFT calls FOUR1.
c
c  Call RFFT (x, n)
c
c          x = the array that will be transformed
c          n = the length of x; n must be an integral power
c              of 2.  On return, the length of x is n/2+1.
c
c-
c
      REAL X(2)
      NN=N/2
      IS=1
      CALL FOUR1(X,NN.IS)
      NM=NN/2
      S=X(1)
      X(1)=X(1)+X(2)
      X(N+1)=S-X(2)
      X(2)=0.
      X(N+2)=0.
      X(NN+2)=-X(NN+2)
      FN=N
      EX=6.2831852/FN
      J=NN
      WR=1.
      WI=0.
      WWR=COS(EX)
      WWI=-SIN(EX)
      DO 1 I=2,NM
      WRR=WR*WWR-WI*WWI
      WI=WR*WWI+WI*WWR
      WR=WRR
      K1J=2*J-1
      K1I=2*I-1
      K2J=2*J
      K2I=2*I
      A1=0.5*(X(K1I)+X(K1J))
      A2=0.5*(X(K2I)-X(K2J))
      B1=0.5*(-X(K1I)+X(K1J))
      B2=0.5*(-X(K2I)-X(K2J))
      S=B1
      B1=B1*WR+B2*WI
      B2=B2*WR-S*WI
      X(K1I)=A1-B2
      X(K2I)=-A2-B1
```

```
      X(K1J)=A1+B2
      X(K2J)=A2-B1
1     J=J-1
      RETURN
      END
```

C-56
subroutine RFFTI
library

```
      SUBROUTINE RFFTI(X,N)
c
c+
c
c Subroutine RFFTI.
c
C  RFFTI computes the inverse finite Fourier transform of the
c complex array  x with length n/2+1, where n is an integral power
c of 2.  The transform returns a real array of length n.
c
c  Call RFFTI (x,n)
c
c          x = complex array
c          n = length of x
c
c-
c
      REAL X(2)
      NN= N/2
      S= X(1)
      X(1)= .5*(X(1)+X( N+1))
      X(2)= .5*(S-X( N+1))
      X(NN+2)= -X(NN+2)
      IS= -1
      NM= NN/2
      FN= N
      EX= 6.2831852/FN
      J = NN
      WR= 1.
      WI=0.
      WWR= COS(EX)
      WWI= -SIN(EX)
      DO 1 I = 2,NM
      WRR= WR*WWR-WI*WWI
      WI= WR*WWI+WI*WWR
      WR=WRR
      K1J = 2*J-1
      K1I = 2*I-1
      K2J = 2*J
      K2I = 2*I
      A1= .5*(X(K1I)+X(K1J))
      A2= .5*(X(K2I)-X(K2J))
      B1= .5*(-X(K1I)+X(K1J))
      B2= .5*(-X(K2I)-X(K2J))
      S= B1
      B1= B1*WR+B2*WI
      B2= B2*WR-S*WI
      X(K1I)= A1-B2
      X(K2I)=-A2-B1
      X(K1J)= A1+B2
      X(K2J)=  A2-B1
1     J= J-1
      CALL FOUR1(X,NN,IS)
```

```
      RETURN
      END
```

C-58
subroutine FOUR1
library

```
      SUBROUTINE FOUR1(DATA,N,ISIGN)
C
c+
c  FOUR1 is the Cooley-Tukey fast Fourier transform.
c
c  Call FOUR1 (data, n, isign)
c
c          data = the 1-dimensional complex array FOUR1
c                 will operate on; its length is n=2**k,
c                 where k is >= 0 (if necessary, append
c                 0's to the end of data)
c          n = the number of points in data
c          isign = -1 or +1; if a -1 transform is followed
c                  by a +1 transform or vice versa, then
c                  the original data reappear multiplied by n.
c
c Transform (k) = sum (data(j)*exp(isign*2*pi*sqrtz(-1)*(j-1)*(k-1)/n))
c
c  where the sum is taken over all j and k from 1 to n.  The
c transform is proportional to n*log2(n), rather than n**2.  The
c RMS relative error is bounded by 6*sqrt(2)*log2(n)*2**(-B),
c where B is the number of bits in the floating point fraction.
c
c Written by Norman Brenner of MIT Laboratory, July 1967.
c
c  NOTE: This is the shortest version of the FFT known to
c the author.  Faster programs FOUR2 and FOURT exist that operate
c on arbitrarily sized multidimensional arrays--see IEEE Audio
c Transactions (June 1967), special issue on FFT).
c
c-
c
      DIMENSION DATA(1)
C
      IP0=2
      IP3=IP0*N
      I3REV=1
      DO 50 I3=1,IP3,IP0
      IF(I3-I3REV)10,20,20
10    TEMPR=DATA(I3)
      TEMPI=DATA(I3+1)
      DATA(I3)=DATA(I3REV)
      DATA(I3+1)=DATA(I3REV+1)
      DATA(I3REV)=TEMPR
      DATA(I3REV+1)=TEMPI
20    IP1=IP3/2
30    IF(I3REV-IP1)50,50,40
40    I3REV=I3REV-IP1
      IP1=IP1/2
      IF(IP1-IP0)50,30,30
50    I3REV=I3REV+IP1
      IP1=IP0
60    IF(IP1-IP3)70,100,100
```

```
70    IP2=IP1*2
      THETA=6.283185307/FLOAT(ISIGN*IP2/IP0)
      SINTH=SIN(THETA/2.)
      WSTPR=-2.*SINTH*SINTH
      WSTPI=SIN(THETA)
      WR=1.
      WI=0.
      DO 90 I1=1,IP1.IP0
      DO 80 I3=I1,IP3,IP2
      I2A=I3
      I2B=I2A+IP1
      TEMPR=WR*DATA(I2B)-WI*DATA(I2B+1)
      TEMPI=WR*DATA(I2B+1)+WI*DATA(I2B)
      DATA(I2B)=DATA(I2A)-TEMPR
      DATA(I2B+1)=DATA(I2A+1)-TEMPI
      DATA(I2A)=DATA(I2A)+TEMPR
80    DATA(I2A+1)=DATA(I2A+1)+TEMPI
      TEMPR=WR
      WR=WR*WSTPR-WI*WSTPI+WR
90    WI=WI*WSTPR+TEMPR*WSTPI+WI
      IP1=IP2
      GO TO 60
100   RETURN
      END
```

files.inc
library

```
c ------------begin files.inc --------
c   Changes:
c     15dec81    Took out the smash file table definitions. Moved them
c                to SFT.INC.
c     15jun81   -ZBIO was renamed to ZBBF
c               -YCBLK.YRBLK are new
c                (get rid of ybblk and yfbllk someday?)
c     30mar81    See smashlib.aaa
c   End of change list.
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c   FILES.INC defines constants and allocates variables (in /FILES/)
c             that are used in calls to FILPKG subroutines.
c
c   /FILES/ contains
c     KDIAG      = lun for FILPKG diagnostic messages.  Set by AINIT.
c     NAMFIL     = Space used for constructing file names.
c                  (+++ NAMFIL should become a character variable since
c                  open statements work faster with character names
c                  rather than Holerith names.  Need to change the
c                  open statements in bio and blkopn first, though.
c                  Maybe add NAMFIL as a character variable and
c                  keep FILNAM as a Hollerith variable until all
c                  references to it have been rewritten. +++)
cc
      parameter ynamf=40
      common  /files/ kdiag,  namfil(ynamf)
      byte namfil
cc
c   Constants used  in calls to FILPKG subroutines.
cc
c   a) FILPKG symbols used with all files --
cc
      parameter zsff=1, zbbf=2, zsuf=3
      parameter znew=0, zold=1, zife=2, zscr=3
      parameter zrw=0,  zro=1
      parameter zsave=1,zdelet=2
      parameter znocc=0,zcc=1
cc
c   b) FILPKG symbols used only with the block io files --
cc
      parameter zpos=0, zread=1, zwrite=2
      parameter zf=3, zi=1, zc=2
      parameter zwaitn=0, zwaita=1, zwaitb=2, zwaitd=3
      parameter znextb=-1, ZLASTB=-7
      parameter zdata1=-5, ZEOD=-6
      parameter zihead=-2, zrhead=-3, zthead =-4
cc
      parameter ybblk=512, yfblk=128, yiblk=256
      parameter ycblk=512, yrblk=128
cc
c   ZNEW,ZOLD,ZIFE, and ZSCR indicate whether a file that is about to
c             be opened shall be a new file, an old file, an old file
```

```
c              if one already exists - a new file if not, or a scratch
c              file.
c              Note that block io files can't be opened with ZSCR.
c   ZRW,ZRO    indicate whether a file shall be opened for read-write or
c              read-only.
c   ZSAVE,ZDELET indicate whether a file shall be saved or deleted
c              when it is closed.
c   ZWAIT-     indicate, in calls to BIO, whether and when to wait
c              for the io to finish.
c               N => no wait,
c               B => wait before the new i/o request,
c               A =>      after
c               D =>      B + A                    .
c   ZNEXTB     is used in calls to bio inplace of the block number when
c              a file is to be accessed sequentially.
c   ZDATA1, ZIHEAD, ZRHEAD, ZTHEAD
c              are used in calls to BIO in place of the block number
c              when a file is to be positioned to the first data block,
c              the first integer header block, the first real header
c              block, or the first text header block.
c   ZF,ZI and ZC indicate, in calls to BIO, whether the buffer size and
c              number of items transfered are counted as floating point
c              words (4 bytes), integer words (2 bytes), or characters
c              (1 byte).
c
c -------- end of files.inc ------
```

C-62
headerloc.inc
library

```
c ----- begin headerloc.inc ---
c   Changes:
c     08dec82   -Replaced HVH (=41) and HVERT (=42) with HUP (=41)
c                according to Ed Cranswick's new convention.
c     10jan82    Added HVH,HORIEN,HVERT,HDTYPE, and all the H-FF-
c                filter items.  Dropped HLSQFC.
c     09oct81   =24sep81 version , but left hand of parameter
c                assingment must be a number with the new compiler.
c   End of change list.
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c   standard locations in an integer header --
c      header-array (hinone) = integer value representing "undefined"
c        "          (hnih  ) = number of integer headers -1 in the file
c        "          (hnth  ) = number of text headers in the file
c        "          (hndb  ) = number of data blocks.
c        "          (hilst ) = index of the last data point within
c                              the last block.
c        "          (hiorr ) = flag to indicate whether the data blocks
c                              contain integer (2byte) or real (4byte)
c                              data.
c        "          (havd  ) = 'bu' -> it's BUTTER data,
c                            = 'ir' -> it's instrument response from
c                                      SCALE,
c                            = 'ac' -> acceleration from DR100, HIFRIC
c                                      or CORAVD,
c                            = 'vl' -> velocity,
c                            = 'dp' -> displacement.
c        "          (horien) = orientation of horizontal components,
c                              between 0 and 360 degrees from north.
c        "          (hup   ) = vertical orientation, degrees from "up".
cc
      parameter hinone=3  hnih=1, hnth=2, hndb=31. hilst=32
      parameter hiorr =4
      parameter hup  =41, horien=42
      parameter havd =43
cc
c   SCRATCH integer header block locations used  in BUTTER output
c     files --
c      header-array (hntrac) = number of digitized traces in the file
c      then, for each trace,
c      header-array (next  ) = type of trace. 1->time-tick,
c                              2-> reference line, and 3-> data.
c      header-array (next  ) = number of the first data block for
c                              the trace (this number doesn t count
c                              the header blocks.)
c      header-array (next  ) = number of data blocks for the trace.
c
      parameter hntrac = 200
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
c   standard locations in a real header block --
c      header-array (hrnone) = real value representing "undefined"
```

```
c         "          (hnrh  ) = number of real headers -1
c         "          (hsps  ) = sampling rate in sanples per second.
c                               undefined => it's butter or phasel data.
c         "          (hdunit) = digitizer output in digitization units/cm.
c         "          (hper  ) = instrument period in seconds
c         "          (hdamp ) = instrument damping as a fraction of
c                               critical damping.
c         "          (hcoilc) = coil constant in DR100 data, or
c                     (=hrsens)   recorder sensitivity, cm/g, in AGRAM
c                               data.
c         "          (hgain ) = gain in DR100 data,
c                               undefined for digitized data.
c         "          (hclock) = clock correction to be added to all
c                               the time values in the data.
c         "          (haafc ) = corner frequency used in anti alias
c                               filter.
c         "          (haafr ) = rolloff bandwith used in  "
cc
      parameter hrnone=2,  hnrh = 1
      parameter hsps  =5,  hdunit=46,  hper =49,  hdamp=50
      parameter haafc =47, haafr =48
      parameter hcoilc=51, hgain =52,  hclock=60
      parameter hrsens=51
cc
c   SCRATCH real header block locations used in SCALE, HIFRIC, CORAVD
c   data files --
c      header-array (htbeg ) = time of the first data point.
c        "          (hvbeg ) = value of the first data point.
c        "          (htend ) = time of the last data point.
c        "          (hvend ) = value of the last data point.
c        "          (htmax ) = time of the max. value
c        "          (hvmax ) = max value
c        "          (htmin ) = time of the min. value
c        "          (hvmin ) = min vlaue
c        "          (hvoffs) = data offset for SCALE data.
c        "          (hbfit ) = Beginning time for the least squares fit
c                                baseline correction done in CORAVD
c        "          (hefit ) = Ending     "
c        "          (hlsqfs) = slope of the linear baseline correction
c                                done in CORAVD.
c        "          (hlsqfi) = intercept            "
c        "          (hdtype) = data type indicator, 1.0,2.1,2.1,3.0,3.1,
c                               or 3.2.
c        "          (hlffc ) = corner frequency for low frequency
c                               filter.
c        "          (hlffr ) = rolloff band of order for low ff.
c        "          (hlffk ) = kind of filter used for the low ff.
c                               (=foat of zuni, zbi,zorm or zfft)
c        "          (hlfff ) = fill option used with the low ff.
c                               (=0., 1., 2., ...)
c        "          (hlffo ) = extra option used with the low ff.
c                               (used as the variable ORK in the ormsby
c                                filter now.)
```

```
headerloc.inc
library

c        "          (hhffc ) = corner frequency for high freq. filter.
c        "          (hhffr )
c        "          (    k )
c        "          (    f )
c        "          (    o )
c        "          (    c )
cc
      parameter htbeg=90,  hvbeg=91,  htend=92, hvend=93
      parameter htmax=94,  hvmax=95,  htmin=96, hvmin=97
      parameter hvoffs=98, hbfit=99,  hefit=100
      parameter hlsqfs=101,hlsqfi=102
      parameter hdtype=103
      parameter hlffc=104, hlffr=106, hlffk=108
      parameter hlfff=110, hlffo=112
      parameter hhffc=105, hhffr=107, hhffk=109
      parameter hhfff=111, hhffo=113
cc
      parameter zuni=1, zbi=2. zorm=3, zfft=4
c ----- end headerloc.inc ---
```

References and Bibliography

General

[1] Hudson, D.E. (1979). Reading And Interpreting Strong Motion Accelerograms: Earthquake Engineering Research Institute, 2620 Telegraph Avenue, Berkeley, CA 94704. 112 pages.

(This monograph provides an introduction to strong-motion accelerograms. It describes the computer processing used in the original CalTech data project.)

[2] Hudson, D.E., (1976). "Strong-Motion Earthquake Accelerograms; Index Volume": Earthquake Engineering Research Laboratory report number EERL 76-02, California Institute of Technology, Pasadena, CA.

(This is the final, summarizing report from the CalTech data project.)

[3] "Strong-Motion Earthquake Accelerograms, Digitization and Analysis, 1971 records": USGS Open-File Report Number 76-609, USGS, Menlo Park, CA.

(This report is the first of the series of strong-motion data reports published by the USGS. It summarizes the processing steps used at that time. It also contains a longer bibliography than is given here.)

[4] Brady, A.G.; Perez, V.; and Mork, P.N. (1982). "Digitization and Processing of Main-shock Ground-motion Data from the USGS Accelerograph Network" in "The Imperial Valley, California, Earthquake of October 15, 1979": Geological Survey Professional Paper 1254, pp. 385-406.

(This report gives a detailed description of the decisions made while processing the set of records recovered from the Imperial Valley earthquake of october 15, 1979.)

[5] Brady, A.G.; Converse, A.M.; Joyner, W.B. and Mork, P.N. (1982). "Processed Accelerograms from the 2319hr 29.62 sec, Aftershock of the 15 October 1979 Imperial Valley, California Earthquake": in preparation.

(This report is the most recent in the same series as

reference [3] above. It is the first report for which the AGRAM programs were used.)

[6] Trifunac, M.D., and V.W. Lee (1979). "Automatic Digitization and Processing of Strong Motion Accelerograms": Department of Civil Engineering, Report number 79-15 I and II, University of Southern California, Los Angeles, CA.

(These reports describe a system similar to the AGRAM programs but which is used by USC and by CDMG.)

[7] Digital Signal Processing Committee of the IEEE (1979). Programs for Digital Signal Processing: IEEE press, The Institute of Electrical and Electronics Engineers, Inc., New York.

(This book presents discussions and software for Discrete Fast Fourier Transform methods, among others.)

HIFRIC
======

[8] Claerbout, J.F. (1976). Fundamentals of Geophysical Data Processing with Applications to Petroleum Prospecting: McGraw Hill.

(The FORK subroutine used for FFT with the FDIC option in HIFRIC is presented in this textbook.)

[9] Oppenheim, A.V., and Schafer, R.W. (1975). Digital Signal Processing: Prentice-Hall.

(Pages 110 through 113 of this textbook describe the overlap-add method used with the FDIC option in HIFRIC.)

CORAVD
======

[10] Basili,M. and Brady, A.G., (1978). "Low frequency Filtering and the Selection of Limits for Accelerogram Corrections": Proc. 6th European Conf. on Earthquake Engineering, Dubrovnik, Yugoslavia.

[11] Fletcher, J.B., Brady, A.G. and Hanks, T.C. (1980). "Strong-Motion Accelerograms of the Oroville, California, Aftershocks: Data Processing and the Aftershock of 0350 August 6, 1975": Bulletin of the Seismological Society of America, vol.70, No.1, pp.243-267.

[12] Hageman, S. (1982). "Program Quickly Figures Complex Filter Parameters": Electronic Design, March 31.

[13] Ormsby, J.F.A., (1961). "Design of Numerical Filters with Application to Missile Data Processing": Journal of the Association for Computing Machinery, v.8, pp.440-446.

PHASE5
======

[14] Perez, V., (1973). "Velocity Response Envelope Spectrum as a Function of Time for the Pacoima Dam, San Fernando Earthquake, February 9,1971": Bulletin of the Seismological Society of America, vol.63, pp.299-313.

PHASE6
======

[15] Perez, V. (1980). "Spectra of Amplitudes Sustained for a Given Number of Cycles: An Interpretation of Response Duration for Strong-Motion Earthquake Records": Bulletin of the Seismological Society of America, vol.70, pp.1943-1954.

USGS LIBRARY-RESTON
3 1818 00068968 5