The *New* SU User's Manual

John W. Stockwell, Jr. & Jack K. Cohen

Version 3.2: August 2002

The Seismic Unix project is supported by these organizations:

The Society of Exploration Geophysicists

Center for Wave Phenomena

Colorado School of Mines
Golden, CO 80401, USA

Past support was received from:

The Gas Research Institute

The CWP/SU Free Software Package: Legal Matters

```
Approved Reference Format:
In publications, please refer to SU as per the following example:
Cohen, J. K. and Stockwell, Jr. J. W., (200_), CWP/SU: Seismic Un*x
Release No. __: a free package for seismic research and processing,
Center for Wave Phenomena, Colorado School of Mines.

Recent articles about SU in peer-reviewed journals:
Saeki, T., (1999), A guide to Seismic Un*x (SU)(2)---examples of data processing (part 1),
Stockwell, Jr. J. W. (1999), The CWP/SU: Seismic Un*x Package, Computers and Geosciences,
Stockwell, Jr. J. W. (1997), Free Software in Education: A case study of CWP/SU: Seismic U
Templeton, M. E., Gough, C.A., (1998), Web Seismic Un*x: Making seismic reflection process
```

```
Acknowledgements:
SU stands for CWP/SU:Seismic Un*x, a processing line developed at Colorado
School of Mines, partially based on Stanford Exploration Project (SEP)
software.
```

Contents

Acknowledgments

The Seismic Unix project is partially funded by the Society of Exploration Geophysicists (SEG), and by the Center for Wave Phenomena (CWP), Department of Geophysical Engineering, Colorado School of Mines. Past support for SU has included these groups, as well as the Gas Research Institute (GRI).

Thank you SEG, and CWP for your continued support!

The sponsors of the CWP Consortium Project have long been partners in the SU project and we are pleased to explicitly acknowledge that relationship here. In addition, we wish to acknowledge extra support supplied in the past by IBM Corporation and by the Center for Geoscience Computing at the Colorado School of Mines during the late 1980's when SU was ported to the modern workstation from its previous incarnation on graphics terminals.

So many faculty and students, both at our Center and elsewhere, have contributed to SU, that it is impossible to list them all here. However, certain people have made such important contributions that they deserve explicit mention.

Einar Kjartansson began writing what is now called SU (the SY package) in the late 1970's while still a graduate student at Jon Claerbout's Stanford Exploration Project (SEP). He continued to expand the package while he was a professor at the University of Utah in the early eighties. In 1984, during an extended visit to SEP Einar introduced SY to Shuki Ronen, then a graduate student at Stanford. Ronen further developed SY from 1984 to 1986. Other students at SEP started to use it and contributed code and ideas. SY was inspired by much other software developed at SEP and benefited from the foundations laid by Clarbout and many of his students; Rob Clayton, Stew Levin, Dave Hale, Jeff Thorson, Chuck Sword, and others who pioneered seismic processing on Unix in the seventies and early eighties.

In 1986, Shuki Ronen brought this work to our Center during a one year postdoctoral appointment at the Colorado School of Mines. During this time, Ronen aided Cohen in turning SU into a supportable and exportable product.

Chris Liner, while a student at the Center, contributed to many of the graphics codes used in the pre-workstation (i.e, graphics terminal) age of SU. Liner's broad knowledge of seismology and seismic processing enabled him to make a positive and continuing influence on the SU coding philosophy.

Craig Artley, now at Golden Geophysical, made major contributions to the graphics codes while a student at CWP and continues to make significant contributions to the general package.

Dave Hale wrote several of the "heavy duty" processing codes as well as most of the core scientific and graphics libraries. His knowledge of good C-language coding practice helped make our package an excellent example for applied computer scientists.

Ken Larner contributed many user interface ideas based on his extensive knowledge of seismic processing in the "real world."

John Scales showed how to use SU effectively in the classroom in his electronic text, Theory of Seismic Imaging, Samizdat Press, 1994. This text is available from the Samizdat press site at: samizdat.mines.edu.

John Stockwell's involvment with SU began in 1989. He is largely responsible for designing the Makefile structure that makes the package easy to install on the majority of Unix platforms. He has been the main contact for the project since the first public release of SU in September of 1992 (Release 17). After Jack Cohen's death in 1996, Stockwell assumed the role of principal investigator of the SU project and has remained in this capacity to the present.

The project has also has had extensive technical help from the worldwide SU user community. Among those who should be singled out for mention are Tony Kocurko at Memorial University in Newfoundland, Toralf Foerster of the Baltic Sea Research Institute in Warnemuende Germany, Stewart A. Levin, John Anderson, and Joe Oravetz at Mobil Oil, Joe Dellinger at Amoco, Matthew Rutty, Jens Hartmann, Alexander Koek, Michelle Miller of the University of Southern California at Chico, Berend Scheffers, and Guy Drijkoningen, Delft, Dominique Rousset, Pau and Marc Schaming, Strasbourg, Matt Lamont, Curtin University, Australia, Wenying Cai at the University of Utah, Brian Macy of Phillips Petroleum, Robert Krueger of Terrasys, Torsten Shoenfelder of the University of Koeln, Germany, Ian Kay of the Geological Survey of Canada, Dr. Toshiki Watanabe of the University of Kyoto. Our apologies in advance for undoubtedly omitting mention of other deserving contributors to SU—we promise to include you in future updates of this manual!

We especially thank Barbara McLenon for her detailed suggestions on the text and also for her excellent design of this document, as well as the SU pamphlets, and other materials, which we distribute at public meetings.

In Memoriam

Dr. Jack K. Cohen passed away in October 1996. The Seismic Unix package exists owing to his knowledge of seismic processing, his creativity, and his desire to make a lasting contribution to the scientific community. He will surely be missed by all who had contact with him in the mathematical, geophysical, and SU-user communities.

Preface

In the 5 years that have elapsed since the first version of this manual was distributed, there have been numerous changes in the SU package. These changes have been a result of both in-house activities here at CWP and, equally important, to the many contributions of code, bug fixes, extensions, and suggestions from SU users outside of CWP. After reviewing these many changes and extensions, and after much discussion with members of the worldwide SU user community, it has become apparent that a new manual was necessary.

This new version began in preparation for a short course on SU at the CREWES project at the University of Calgary. Many of the items that were in the original manual, such as information about obtaining and installing the package, have been have been moved to appendices. Additional sections describing the core collection of programs have been added. The entire manual has been expanded to include more detailed descriptions of the codes.

The intent is that you will be able to find your way around the package, via the help mechanisms, that you will learn to run the programs by running the demos, and finally will be able to see how to begin writing SU code, drawing on the source code of the package.

Chapter 1

About SU

In 1987, Jack K. Cohen and Shuki Ronen of the Center for Wave Phenomena (CWP) at the Colorado School of Mines (CSM) conceived a bold plan. This plan was to create a seismic processing environment for Unix-based systems, written in the C language, that would extend the Unix operating system to seismic processing and research tasks. Furthermore, they intended that the package be freely available as full source code to anyone who would want it.

They began with a package called SY, started by Einar Kjartansson while he was a graduate student at Jon Claerbout's Stanford Exploration Project (SEP) in the late seventies, and improved while he was a professor at the University of Utah in the eighties. In 1984 Einar returned to SEP to baby sit the students (and Jon's house) while Claerbout went for a summer vacation. Einar introduced SY to Shuki Ronen, then a graduate student at SEP (a day before Claerbout returned from vacation and Einar left back to Utah). Ronen further developed SY from 1984 to 1986. Other students at SEP started to use it and contributed code and ideas. SY was inspired by much other software developed at SEP and benefited from the foundations laid by Clarbout and many of his students; Rob Clayton, Stew Levin, Dave Hale, Jeff Thorson, Chuck Sword, and others who pioneered seismic processing on Unix in the seventies and early eighties.

In 1986, when Ronen accepted a one year visiting professorship at CSM he was encouraged by Claerbout to disseminate whatever SEP ideas and software he cared for. In the days before Internet and ftp, SY was on a 9 track tape in the trunk of the car he drove from California to Colorado. Ronen went to CSM to replace Jack Cohen who went on sabbatical. Fortunately for SU, Jack's sabbatical lasted only two months, instead of the whole year initially planned, because of the poor health of Jack's son Dan. Upon Jack's return to CWP, he became interested in SY. This package was a sharp departure from the commercial seismic processing software that was available at the time. The industry standard at the time was to use Fortran programs on VAX-VMS based systems.

By the time Cohen and Ronen created the first version of SU in 1987 the sponsors of CWP had already begun showing interest in the package. The availability of Unix-based workstations combined with the influx of Unix-literate geophysicists from the major academic institutions, shifted the industry to using primarily Unix-based systems for research and for processing, increasing the interest in Unix-based software, including SU.

Until September of 1992, SU was used primarily in-house at CWP. Earlier versions of SU had been ported to CWP sponsor companies, even providing the basis for in-house seismic processing packages developed by some of those companies! Once SU became generally available on the Internet, it began to be used by a much broader community. The package is used by exploration geophysicists, earthquake seismologists, environmental engineers, software developers and others. It is used by scientific staff in both small geotechnical companies and major oil and gas companies, and by academics and government researchers, both as a seismic data processing and software development environment.

1.1 What SU is

The SU package is *free software*, meaning that you may have unrestricted use of the codes for both processing and software development, provided that you honor the license that appears at the beginning of this manual and as the file LEGAL_STATEMENT in the current release of the package. (This latter file takes precedence). The package is maintained and expanded periodically, with each new release appearing at 3 to 6 month intervals, depending on changes that accumulate in the official version here at CWP. The package is distributed with the full source code, so that users can alter and extend its capabilities. The philosophy behind the package is to provide both a free processing and development environment in a proven structure that can be maintained and expanded to suit the needs of a variety of users.

The package is not necessarily restricted to seismic processing tasks, however. A broad suite of wave-related processing can be done with SU, making it a somewhat more general package than the word "seismic" implies. SU is intended as an extension of the Unix operating system, and therefore shares many characteristics of the Unix, including Unix flexibility and expandibility. The fundamental Unix philosophy is that all operating system commands are programs run under that operating system. The idea is that individual tasks be identified, and that small programs be written to do those tasks, and those tasks alone. The commands may have options that permit variations on those tasks, but the fundamental idea is one-program, one-task. Because Unix is a multi-tasking operating system, multiple processes may be strung together in a cascade via "pipes" (|).

This decentralization has the advantage of minimizing overhead by not launching single "monster" applications that try to do everything, as is seen in Microsoft applications, or in some commercial seismic utilities, for example.

Unix has the added feature of supporting a variety of shell languages, making Unix, itself, a meta-language. Seismic Unix benefits from all of these attributes as well. In combination with standard Unix programs, Seismic Unix programs may be used in shell scripts to extend the functionality of the package.

Of course, it may be that no Unix or Seismic Unix program will fulfill a specific task. This means that new code has to be written. The availability of a standard set of source code, designed to be readable by human beings, can expedite the process of extending the package through the addition of new source code.

1.2 What SU is not

The SU package is not a graphical user interface driven utility at this time, though there are several possibilities including Java and TCL/TK scripts, which may be exploited for this purpose in the future. Because most commercial seismic processing packages are GUI-based, it is unavoidable that users will expect SU to be similar to these packages. However, this is not a fair comparison, for several reasons.

As mentioned above, SU is an extension of the Unix operating system. Just as there are no GUI driven Unix operating systems that give full access to all of the capabilities of Unix from menus, it is not reasonable to expect full access to Seismic Unix through such an interface. At most, any SU interface will give limited access to the capabilities of the package.

SU is not a replacement for commercial seismic packages. Commercial seismic software packages are industrial-strength utilities designed for the express purpose of production-level seismic processing. If you do commercial-level processing, and have dedicated, or plan to dedicate funds to purchase one or more license of such commercial software, then it is unlikely that you will be able to substitute SU for the commercial utility.

However, SU can be an important adjunct to any commercial package you use. Where commercial packages are used for production work, SU often has found a place as a prototyping package. Also, if new code needs to be written, SU can provide a starting base for new software applications. Indeed, the availability of seismic processing capability may encourage non-processors to experiment with processing, non-software-developers to experiment with software development, and non-researchers to engage in research.

SU is not confined to seismic applications. It may find use, both in geophysical and more general signal processing applications. It certainly can be useful for teaching students about "wave related" signal processing and, particularly, Fourier transform properties. This can include radar, non-seismic acoustics, and image processing, just to name a few.

Another thing that SU is not, at least in its current version, is a 3D package. However, it is not, expressly a 2D package, either, as there are numerous filtering and trace manipulation tasks that are the same in 2D as in 3D. It is likely, however, that there will be 3D applications in future releases of SU. A single 3D migration code appeared first in Release 32, which will hopefully set the stage for new developments.

1.3 Obtaining and Installing SU

Because the coding standards of SU stress portability, the package will install on any system that is running some form of the Unix operating system, and has a decent version of "make" and an ANSI C compiler. The programs GNU Make and GCC may be substituted for these respective programs on most systems.[1]

New releases of SU are issued at 3 to 6 month intervals, depending upon the accumulation of changes in the home version at CWP. Old releases are not supported. However, if materials appear to break between releases, please contact me (John Stockwell) by email so that we can fix the problem.

Instructions for obtaining and installing SU may be found in Appendix A of this manual.

[1]Remarkably, a software package called **CYGWIN32**, recently released by the GNU project, provides enough Unix-like functionality, so that SU may be ported to Windows NT, with no changes to the SU source code or Makefiles!

Chapter 2

Help facilities

Like the Unix operating system, Seismic Unix may be thought of as a language (or meta-language). As with any language, a certain amount of vocabulary must be mastered before useful operations may be performed. Because the SU contains many programs, there must be a "dictionary" to permit the inevitable questions about vocabulary can be answered. It is intended that this manual be the beginning of such a dictionary.

SU does not have "man pages," in the same way that Unix does, but it does have equivalent internal documentation mechanisms. There is a general level help utilities which give an overview of what is available. For information about specific aspects of a particular code, the majority of the programs contain a **selfdoc**—a self-documentation paragraph, which will appear when the name of the program is typed on the commandline, with no options.

(In all of the examples that follow, the percent sign "%" indicates a Unix commandline prompt, and is not typed as part of the command.)

The following tools provide internal documentation at various levels of detail for the main programs, shell scripts, and library functions in the package:

- SUHELP - list the CWP/SU programs and shells
- SUNAME - get name line from self-docs and location of the source code
- The selfdoc - is an internal documentation utility which exists in the majority of executable mains and shell scripts. The selfdoc is seen by typing the name of the program or shell script on the commandline with no arguments, and without redirection of input or output via pipes | or Unix redirects > <, For non-executables (library routines) and for programs without the selfdoc feature, there is a dummy selfdoc included which provides a database of information about those items, as well,
- SUDOC - get DOC listing for code
- SUFIND - get info from self-docs
- GENDOCS - generate complete list of selfdocs in latex form
- suhelp.html - is an HTML global overview of SU programs by subject matter,
- SUKEYWORD – guide to SU keywords in segy.h

This chapter discusses each of these utilities, with the intent of showing the reader how to get from the most general to the most specific information about SU programs.

2.1 SUHELP - List the Executable Programs and Shell Scripts

For a general listing of the contents of SU, which includes each executable (that is, each main program and shell script) in the package type:

```
% suhelp

CWP PROGRAMS: (no self-documentation)
ctrlstrip       fcat            maxints         t
downfort        isatty          pause           upfort

PAR PROGRAMS: (programs with self-documentation)
a2b             kaperture       resamp          transp          vtlvz
b2a             makevel         smooth2         unif2           wkbj
farith          mkparfile       smoothint2      unisam
ftnstrip        prplot          subset          unisam2
h2b             recast          swapbytes       velconv


press return key to continue
....
```

Please type **suhelp** or see Appendix B for the full text.

Another useful command sequence is:

```
% suhelp | lpr
```

which will the output from **suhelp** to the local print.

2.2 SUNAME - Lists the Name and Short Description of Every Item in SU

A more complete listing of the contents of the CWP/SU package may be obtained by typing

```
% suname

 -----  CWP Free Programs -----
CWPROOT=/usr/local/cwp

Mains:

In CWPROOT/src/cwp/main:
* CTRLSTRIP - Strip non-graphic characters
* DOWNFORT - change Fortran programs to lower case, preserving strings
* FCAT - fast cat with 1 read per file
* ISATTY - pass on return from isatty(2)
* MAXINTS - Compute maximum and minimum sizes for integer types
* PAUSE - prompt and wait for user signal to continue
* T - time and date for non-military types
* UPFORT - change Fortran programs to upper case, preserving strings

In CWPROOT/src/par/main:
```

```
A2B - convert ascii floats to binary
B2A - convert binary floats to ascii
DZDV - determine depth derivative with respect to the velocity ",
FARITH - File ARITHmetic -- perform simple arithmetic with binary files
FTNSTRIP - convert a file of floats plus record delimiters created
H2B - convert 8 bit hexidecimal floats to binary
KAPERTURE - generate the k domain of a line scatterer for a seismic array
MAKEVEL - MAKE a VELocity function v(x,y,z)
MKPARFILE - convert ascii to par file format
PRPLOT - PRinter PLOT of 1-D arrays f(x1) from a 2-D function f(x1,x2)
RAYT2D - traveltime Tables calculated by 2D paraxial RAY tracing
RECAST - RECAST data type (convert from one data type to another)
REGRID3 - REwrite a [ni3][ni2][ni1] GRID to a [no3][no2][no1] 3-D grid
RESAMP - RESAMPle the 1st dimension of a 2-dimensional function f(x1,x2)
...
```

Please type: **suhelp** or see Appendix B for the full text.

2.3 The Selfdoc - Program Self-Documentation

There are no Unix man pages for SU. To some people that seems to be a surprise (even a disappointment) as this would seem to be a standard Unix feature, which Seismic Unix should emulate. The package does contain an equivalent mechanism called a **selfdoc** or self-documentation feature.

This is a paragraph that is written into every program, and arranged so that if the name of the program is typed on the commandline of a Unix terminal window, with no options or redirects to or from files, the paragraph is printed to standard error (the screen).

For example:

```
% sustack

 SUSTACK - stack adjacent traces having the same key header word

     sustack <input >output [Optional parameters]

 Required parameters:
        none

 Optional parameters:
        key=cdp         header key word to stack on
        normpow=1.0     each sample is divided by the
                        normpow'th number of non-zero values
                        stacked (normpow=0 selects no division)
        verbose=0       verbose = 1 echos information

 Note:  The offset field is set to zero on the output traces.
        Sushw can be used afterwards if this is not acceptable.
```

The first line indicates the name of the program **sustack** and a short description of what the program does. This is the same line that appears for the listing of **sustack** in the **suname** listing. The second line

```
    sustack <stdin >stdin [Optional parameters]
```

indicates how the program is to be typed on the commandline, with the words "stdin" and "stdout" indicating that the input is from the standard input and standard output, respectively. What this means in Unix terms is that the user could be inputting and outputting data via diskfiles or the Unix "redirect in" < and "redirect out" > symbols, or via pipes |.

The paragraphs labeled by "Required parameters:" and "Optional parameters" indicate the commandline parameters which are required for the operation of the program, and those which are optional. The default values of the Optional parameters are given via the equality are the values that the program assumes for these parameters when data are supplied, with no additional commandline arguments given. For example: "key=cdp" indicates that **sustack** will stack over the common depth point gather number field, "cdp." (The shell script **sukeyword** tells the choices of keyword that are available.)

2.4 SUDOC - List the Full Online Documentation of any Item in SU

As has been alluded to in previous sections of this manual, there is a database of selfdocumentation items that is available for each main program, shell script, and library function. This database exists in the directory $CWPROOT/src/doc and is composed of all of the selfdocumentation paragraphs of all of the items in SU.

Because not all all items with selfdocs are executable, an additional mechanism is necessary to see the selfdoc for these items. For example, information about the Abel transform routines, located in $CWPROOT/src/cwp/lib/abel.c (on the system at CWP, CWPROOT=/usr/local/cwp) is obtained via

```
% sudoc abel

In /usr/local/cwp/src/cwp/lib:
ABEL - Functions to compute the discrete ABEL transform:

abelalloc allocate and return a pointer to an Abel transformer
abelfree  free an Abel transformer
abel compute the Abel transform

Function prototypes:
void *abelalloc (int n);
void abelfree (void *at);
void abel (void *at, float f[], float g[]);

Input:
ns number of samples in the data to be transformed
f[] array of floats, the function being transformed

Output:
at pointer to Abel transformer returned by abelalloc(int n)
g[] array of floats, the transformed data returned by
abel(*at,f[],g[])
```

```
Notes:
The Abel transform is defined by:

         Infinity
g(y) = 2 Integral dx f(x)/sqrt(1-(y/x)^2)
   |y|

Linear interpolation is used to define the continuous function f(x)
corresponding to the samples in f[].  The first sample f[0] corresponds
to f(x=0) and the sampling interval is assumed to be 1.  Therefore, the
input samples correspond to 0 <= x <= n-1.  Samples of f(x) for x > n-1
are assumed to be zero.  These conventions imply that

g[0] = f[0] + 2*f[1] + 2*f[2] + ... + 2*f[n-1]

References:
Hansen, E. W., 1985, Fast Hankel transform algorithm:  IEEE Trans. on
Acoustics, Speech and Signal Processing, v. ASSP-33, n. 3, p. 666-671.
(Beware of several errors in the equations in this paper!)

Authors:  Dave Hale and Lydia Deng, Colorado School of Mines, 06/01/90
```

Here we see that sudoc shows information about the routines, including their names, usage information (via the function prototype), some theory of how the items are used, published references, and finally the author's names.

As an another example, type:

```
% sugabor

 SUGABOR -  Outputs a time-frequency representation of seismic data via
                the Gabor transform-like multifilter analysis technique
                presented by Dziewonski, Bloch and  Landisman, 1969.

    sugabor <stdin >stdout [optional parameters]

 Required parameters:
        if dt is not set in header, then dt is mandatory

 Optional parameters:
        dt=(from header)        time sampling interval (sec)
        fmin=0                  minimum frequency of filter array (hz)
        fmax=NYQUIST            maximum frequency of filter array (hz)
        beta=3.0                ln[filter peak amp/filter endpoint amp]
        band=.05*NYQUIST        filter bandwidth (hz)
        alpha=beta/band^2       filter width parameter
        verbose=0               =1 supply additional info

 Notes: This program produces a muiltifilter (as opposed to moving window)
 representation of the instantaneous amplitude of seismic data in the
 time-frequency domain. (With Gaussian filters, moving window and multi-
 filter analysis can be shown to be equivalent.)

 An input trace is passed through a collection of Gaussian filters
 to produce a collection of traces, each representing a discrete frequency
 range in the input data. For each of these narrow bandwidth traces, a
 quadrature trace is computed via the Hilbert transform. Treating the narrow
 bandwidth trace and its quadrature trace as the real and imaginary parts
 of a "complex" trace permits the "instantaneous" amplitude of each
 narrow bandwidth trace to be compute. The output is thus a representation
 of instantaneous amplitude as a function of time and frequency.

 Some experimentation with the "band" parameter may necessary to produce
 the desired time-frequency resolution. A good rule of thumb is to run
 sugabor with the default value for band and view the image. If band is
 too big, then the t-f plot will consist of stripes parallel to the frequency
 axis. Conversely, if band is too small, then the stripes will be parallel
 to the time axis.

 Examples:
    suvibro | sugabor | suximage
    suvibro | sugabor | suxmovie n1= n2= n3=
     (because suxmovie scales it's amplitudes off of the first panel,
     may have to experiment with the wclip and bclip parameters
    suvibro | sugabor | supsimage | ... ( your local PostScript utility)
```

If you compare this output to the output from typing:

```
% sudoc sugabor
```

You will see the same output as above, preceeded by a line showing the location of the source code, and followed by the additional paragraphs:

```
 Credits:

CWP: John Stockwell, Oct 1994

 Algorithm:

 This programs takes an input seismic trace and passes it
 through a collection of truncated Gaussian filters in the frequency
 domain.

 The bandwidth of each filter is given by the parameter "band". The
 decay of these filters is given by "alpha", and the number of filters
 is given by nfilt = (fmax - fmin)/band. The result, upon inverse
 Fourier transforming, is that nfilt traces are created, with each
 trace representing a different frequency band in the original data.

 For each of the resulting bandlimited traces, a quadrature (i.e. pi/2
 phase shifted) trace is computed via the Hilbert transform. The
 bandlimited trace constitutes a "complex trace", with the bandlimited
 trace being the "real part" and the quadrature trace being the
 "imaginary part".  The instantaneous amplitude of each bandlimited
 trace is then computed by computing the modulus of each complex trace.
 (See Taner, Koehler, and Sheriff, 1979, for a discussion of complex
 trace analysis.

 The final output for a given input trace is a map of instantaneous
 amplitude as a function of time and frequency.

 This is not a wavelet transform, but rather a redundant frame
 representation.

 References:  Dziewonski, Bloch, and Landisman, 1969, A technique
for the analysis of transient seismic signals,
Bull. Seism. Soc. Am., 1969, vol. 59, no.1, pp.427-444.

Taner, M., T., Koehler, F., and Sheriff, R., E., 1979,
Complex seismic trace analysis, Geophysics, vol. 44,
pp.1041-1063.

  Chui, C., K.,1992, Introduction to Wavelets, Academic
Press, New York.

 Trace header fields accessed: ns, dt, trid, ntr
 Trace header fields modified: tracl, tracr, d1, f2, d2, trid, ntr
```

There is more information in the **sudoc** listing, than in the selfdoc listing. The selfdoc is intended as a quick reference, whereas the sudoc listing can provide additional information that we do not necessarily want to see everytime we are, say, simply wanting to know what a particular parameter means, for example.

2.5 SUFIND - Find SU Items with a Given String

The "doc" database (located in $CWPROOT/src/doc) may also be searched for specific strings, or topics. The shell script **sufind** serves this purpose. If you type **sufind** with no options, you will see

```
% sufind

sufind - get info from self-docs about SU programs
Usage: sufind [-v -n -P<command_pattern>] string
        ("string" can be an "egrep" pattern)
"sufind string" gives brief synopses
"sufind -v string" verbose hunt for relevant items
"sufind -n name_fragment" searches for command name
"sufind -P<pattern> string"
        gives brief synopses by searching for "string" among
        commands/libraries whose names match the "pattern"
```

showing that there is some sophisticated functionality which may help you find items of a particular type in the SU package.

As an example of this, let's say that you are looking for programs that use the fast Fourier transform algorithms in SU, type

```
% sufind fft

 FFTLAB - Motif-X based graphical 1D Fourier Transform

 Usage:  fftlab


HANKEL - Functions to compute discrete Hankel transforms

hankelalloc     allocate and return a pointer to a Hankel transformer
hankelfree      free a Hankel transformer

PFAFFT - Functions to perform Prime Factor (PFA) FFT's, in place

npfa            return valid n for complex-to-complex PFA
npfar           return valid n for real-to-complex/complex-to-real PFA

 SUAMP - output amp, phase, real or imag trace from
        (frequency, x) domain data

 suamp <stdin >stdout mode=amp

 SUFFT - fft real time traces to complex frequency traces

 suftt <stdin >sdout sign=1


 SUFRAC -- take general (fractional) time derivative or integral of
            data, plus a phase shift.  Input is TIME DOMAIN data.

 sufrac power= [optional parameters] <indata >outdata

 SUIFFT - fft complex frequency traces to real time traces

 suiftt <stdin >sdout sign=-1


 SUMIGPS - MIGration by Phase Shift with turning rays

 sumigps <stdin >stdout [optional parms]


 SUMIGTK - MIGration via T-K domain method for common-midpoint stacked data

 sumigtk <stdin >stdout dxcdp= [optional parms]


 SURADON - forward generalized Radon transform from (x,t) -> (p,tau) space.

 suradon <stdin >stdout [Optional Parameters]
```

```
For more information type: "program_name <CR>"
```

The final line of this output ends with a symbol meant to indicate that the user is to type a carriage return.[1] As you can see, there is a mixed collection of programs which are either demos of the CWP/SU prime factor fft routines (pfafft), libraries containing the routines, or programs that use fft routines.

2.5.1 Getting information about SU programs

A more specific example would be to use **sufind** to look for dip moveout (DMO) programs:

```
% sufind dmo

 SUDMOFK - DMO via F-K domain (log-stretch) method for common-offset gathers

 sudmofk <stdin >stdout cdpmin= cdpmax= dxcdp= noffmix= [...]


 SUDMOTX - DMO via T-X domain (Kirchhoff) method for common-offset gathers

 sudmotx <stdin >stdout cdpmin= cdpmax= dxcdp= noffmix= [optional parms]

 SUDMOVZ - DMO for V(Z) media for common-offset gathers

 sudmovz <stdin >stdout cdpmin= cdpmax= dxcdp= noffmix= [...]

 SUFDMOD2 - Finite-Difference MODeling (2nd order) for acoustic wave equation

 sufdmod2 <vfile >wfile nx= nz= tmax= xs= zs= [optional parameters]


 SUSTOLT - Stolt migration for stacked data or common-offset gathers

 sustolt <stdin >stdout cdpmin= cdpmax= dxcdp= noffmix= [...]
```

The last two “hits” are spurious, but we see that three DMO programs have been found.
Now use the self-doc facility to get more information about **sudmofk**:

```
% sudmofk

 SUDMOFK - DMO via F-K domain (log-stretch) method for common-offset gathers

 sudmofk <stdin >stdout cdpmin= cdpmax= dxcdp= noffmix= [...]

 Required Parameters:
 cdpmin                  minimum cdp (integer number) for which to apply DMO
 cdpmax                  maximum cdp (integer number) for which to apply DMO
 dxcdp                   distance between adjacent cdp bins (m)
 noffmix                 number of offsets to mix (see notes)
```

[1]The phrase “carriage return” refers to an older technology, the typewriter. Ask your parents (or grandparents) for further details.

```
Optional Parameters:
tdmo=0.0                times corresponding to rms velocities in vdmo (s)
vdmo=1500.0             rms velocities corresponding to times in tdmo (m/s)
sdmo=1.0                DMO stretch factor; try 0.6 for typical v(z)
fmax=0.5/dt             maximum frequency in input traces (Hz)
verbose=0               =1 for diagnostic print

Notes:
Input traces should be sorted into common-offset gathers.  One common-
offset gather ends and another begins when the offset field of the trace
headers changes.

The cdp field of the input trace headers must be the cdp bin NUMBER, NOT
the cdp location expressed in units of meters or feet.

The number of offsets to mix (noffmix) should typically equal the ratio of
the shotpoint spacing to the cdp spacing.  This choice ensures that every
cdp will be represented in each offset mix.  Traces in each mix will
contribute through DMO to other traces in adjacent cdps within that mix.

The tdmo and vdmo arrays specify a velocity function of time that is
used to implement a first-order correction for depth-variable velocity.
The times in tdmo must be monotonically increasing.

For each offset, the minimum time at which a non-zero sample exists is
used to determine a mute time.  Output samples for times earlier than this
mute time will be zeroed.  Computation time may be significantly reduced
if the input traces are zeroed (muted) for early times at large offsets.

Trace header fields accessed:  ns, dt, delrt, offset, cdp.
```

By using **sufind** in conjunction with the selfdoc feature (or **sudoc**), and choosing smart strings to search on, it is possible to find a great deal of detailed information about SU's facilities.

2.6 GENDOCS - An Instant LaTeX Document Containing All Selfdocs

The ultimate shell script for exploiting the **sudoc** database is **gendocs**. Typing:

```
% gendocs -o
```

will generate the 528+ page document "selfdocs.tex", which is in LaTeX format. You may process this using LaTeX on your system. Obviously, you must *really* be sure that you want to print this document, considering its size. However, it does contain all of the self-documentations for all CWP/SU programs, library routines, and shell scripts, and may be a useful one-to-a-lab type reference.

2.7 Suhelp.html

Long-time SU contributor, Dr. Christopher L. Liner of the University of Tulsa, created the following document which may be accessed from the CWP/SU web site, or from his location of:

`http://www.mcs.utulsa.edu/~cll/suhelp/suhelp.html`

```
                                SeismicUn*x

                          Version 33 (5 April 1999)

                            An HTML Help Browser

   * This is a help browser for the SeismicUn*x free software package
     developed and maintained by the Center for Wave Phenomena at the
     Colorado School of Mines. The SU project is directed at CWP by John
     Stockwell.
   * The author of this help facility is Dr. Christopher Liner (an alumnus
     of CWP) who is a faculty member in the Department of Geosciences at The
     University of Tulsa.
   * Last updated January 16, 1998



          o The arrangement below is by funtionality
          o Clicking on a program name pulls up the selfdoc for that item
          o Your web browser's Find capability is useful if you have a
            fragment in mind (e.g. sort or nmo)
          o While programs may logically apply to more than one catagory
            below, each program appears only once
     -----------------------------------------------------------------------

  1. Functional Listing
       1. Data Compression
       2. Editing, Sorting and Manipulation
       3. Filtering, Transforms and Attributes
       4. Gain, NMO, Stack and Standard Processes
       5. Graphics
       6. Import/Export
       7. Migration and Dip Moveout
       8. Simulation and Model Building
       9. Utilities

    * Alphabetical name list            * 280 items
    * List is for scanning only, it     * For further info on an
      does not pull up the selfdoc of     interesting item use your
      a selected item.                    browser's find command, then
                                          follow the link

  -----------------------------------------------------------------------

Data Compression

 Discrete Cosine Transform
                                   * dctcomp           * dctuncomp
 Packing                          * supack1           * suunpack1
                                  * supack2           * suunpack2
                                  * wpc1comp2         * wpc1uncomp2
 Wavelet Transform                * wpccompress       * wpcuncompress
                                  * wptcomp           * wptuncomp
                                  * wtcomp            * wtuncomp

                                 Go to top
...
```

To see the full listing see Appendix B or point your web browser to the http address above.

2.8 Demos

The SU package contains a suite of demos, which are shell scripts located in the directory $CWPROOT/src/demos.

The instructions for accessing the demos are located in the $CWPROOT/src/demos/README

- The Making_Data demos shows the basics of making synthetic data shot gathers and common offset sections using susynlv. Particular attention is paid to illustrating good display labeling.
- The Filtering/Sufilter demo illustrates some real data processing to eliminate ground roll and first arrivals. The demos in the Filtering subdirectories give instructions for accessing the data from the CWP ftp site.
- The Deconvolution demo uses simple synthetic spike traces to illustrate both dereverberation and spiking decon using supef and other tools. The demos include commands for systematically examining the effect of the filter parameters using loops.
- The Sorting_Traces Tutorial is an interactive script that reinforces some of the basic UNIX and SU lore discussed in this document. The interactivity is limited to allowing you to set the pace. Such tutorials quickly get annoying, but we felt that one such was needed to cover some issues that didn't fit into our standard demo format. There is also a standard, but less complete, demo on this topic.
- The next step is to activate the Selecting_Traces Demo. Then proceed to the NMO Demo.

Beyond that, visit the Demo directories that interest you. The **demos** directory tree is still under active development—please let us know if the demos are helpful and how they can be improved.

2.9 Other Help Mechanisms

- SUKEYWORD - List SU Header Field Key Words Many of the SU programs that draw on header field information have the parameter "key=" listed in their selfdocs, with the reference to "keywords." The SU keywords are based on the SEGY trace header fields. (This will be explained later, in the sections on tape reading and data format conversion.) To find out what these header fields are, and what they stand for in the SU data type, type:

```
% sukeyword -o
```

Please see Appendix B for the full text of the output. See Section 3.5.3 for details on the usage of this command. A number of programs sort data, window data, and display data by making use of the values in the SU header fields, making **sukeyword** an oft-used utility.

- The essence of SU usage is the construction of shell programs to carry out coordinated data processing. The **su/examples** directory contains a number of such programs. By the way, the terms "shell scripts," "shell programs," "shell files," and "shells," are used interchangeably in the UNIX literature.
- The **faq** directory contains a growing collection of answers to frequently asked questions about SU, including detailed information about tape reading, data format questions, and seismic processing tips.
- The text book, *Theory of Seismic Imaging*, by John A. Scales, Samizdat Press, 1994, is available in our anonymous ftp site in both 300 and 400 dots per inch PostScript format: `pub/samizdat/texts/imaging/imaging_300dpi.ps.Z` or `imaging_400dpi.ps.Z`. The exercises in this text make extensive use of SU.

You should not hesitate to look at the source code itself. Section 10.2 explains the key SU coding idioms. Please let us know if you discover any inconsistencies between the source and our documentation of it. We also welcome suggestions for improving the comments and style of our codes. The main strength of the SU package is that it is a source-code product. No commercial package can give you the source code, for obvious reasons.

You may direct email to: john@dix.mines.edu if you have comments, questions, suggestions regarding SU, or if you have your own SU-style programs to contribute to the package.

Chapter 3

Core Seismic Unix Programs

The core of the Seismic Unix program set performs a broad collection of tasks, which may be viewed as being common to a large collection of research and processing disciplines. Many, however, are purely seismic in nature, and are indicated as such in the text.

Some of these tasks include

- input/output issues
- data format conversion,
- setting, viewing, and editing trace header fields
- viewing SU data,
- windowing, sorting, and editing the data.
- general operations,
- transform and filtering operations,
- seismic operations on SU data.

It is the intent of the chapters that follow is to deal with many of these issues. Please note that more detailed information about any of the programs discussed can be obtained by typing the name of the program on the commandline with no arguments,

```
% programname
```

or by typing:

```
% sudoc programname
```

to see the selfdoc information.

This chapter does not cover all SU programs, but just a sufficient number such that a person with some Unix experience can get started with the package. Once you get the idea of how SU is used, then you may draw on the help facilities to discover additional programs in the package.

3.1 Reading and Writing Data to and from Tapes

Reading tapes is more of an art than a science. This is true in general, and is especially true for SU. The variability of hardware formats, as well as the variability of data format types, makes the creation of a "general tape reading utility" a challenging, if not impossible, proposition.

The following programs are useful for the specialized data input and output tasks related to geophysical applications, as well as to the internal SU data format

- BHEDTOPAR - convert a Binary tape HEaDer file to PAR file format
- DT1TOSU - Convert ground-penetrating radar data in the Sensors & Software X.dt1 GPR format to SU format
- SEGDREAD - read an SEG-D tape
- SEGYCLEAN - zero out unassigned portion of header
- SEGYREAD - read an SEG-Y tape
- SEGYHDRS - make SEG-Y ascii and binary headers for segywrite
- SEGYWRITE - write an SEG-Y tape
- SETBHED - SET the fields in a SEGY Binary tape HEaDer file
- SUADDHEAD - put headers on bare traces and set the tracl and ns fields
- SUSTRIP - remove the SEGY headers from the traces
- SUPASTE - paste existing SEGY headers on existing data

The following programs are useful for general data input, output, and data type conversion, which may also find use in tape reading,

- A2B - convert ascii floats to binary
- B2A - convert binary floats to ascii
- FTNSTRIP - convert Fortran floats to C-style floats
- H2B - convert 8 bit hexidecimal floats to binary
- RECAST - RECAST data type (convert from one data type to another)
- TRANSP - TRANSPose an n1 by n2 element matrix

3.1.1 The SEGY format and the SU data format

The data format that is expected by all programs in the CWP/SU package whose names begin with the letters 'su' (with the exception of the program **subset**), consists of "SEGY traces written in the native binary format of the machine you are running the programs on." To understand what this phrase means, we must understand what the SEGY standard is.

In the early 1980's, the most common data storage format was SEG-Y. This is the Society of Exploration Geophysicists Y format which is described in the SEG's publication *Digital Tape Standards*. The format is still widely used, today, though there is no guarantee that the format is used "by the book."

The SEGY data format consists of 3 parts. The first part is a 3200 byte EBCDIC card image header which contains 40 cards (i.e. 40 lines of text with 80 characters per line) worth of text data describing the tape. The second part is a 400 byte binary header containing information about the contents of the tape reel. The third portion of the SEG-Y format consists of the actual seismic traces. Each trace has a 240 byte *trace header*. The data follow, written in one of 4 possible 32 formats in IBM floating point notation as defined in IBM Form GA 22-6821. (Note, this "IBM format" is not the common IEEE format found on modern IBM PC's.)

The SU data format is based on the trace portion of the SEGY format. The primary difference between the SEGY traces and SU traces is that the data portion of the SU format are floats, written in the native binary float format of the machine you are running SU on. SU data consists of the SEGY traces *only*! The ebcdic and binary reel headers are not preserved in the SU format, so simply redirecting in a SEGY file will not work with any SU program.

To convert SEGY data into a form that can be used by SU programs, you will need to use **segyread**.

3.1.2 SEGYREAD - Getting SEG-Y data into SU

The program **segyread** is used to convert data from the SEGY format to the SU format. If you type:

```
% segyread
```

You will see the selfdoc for this program.

When reading a SEGY tape, or datafile, you will need to be aware of the byte-order (endian) of the machine you are running on. The so-called "big-endian" or high-byte IEEE format is found on SGI, SUN, IBM RS6000, and all Motorola chip-based systems. The "little-endian" or low-byte systems are systems that are based on Intel and Dec chips. You will also need to know what Unix device your tape drive is.

A typical execution of **segyread** on a big-endian machine, looks like this:

```
% segyread tape=/dev/rmt0 verbose=1 endian=1 > data.su
```

More often you will have to use the following

```
% segyread tape=/dev/rmt0 verbose=1 endian=1 | segyclean > data.su
```

for reading a tape on a big-endian platform.

There are optional header fields (bytes 181-240) in the SEGY trace headers. There is no standard for what may go in these fields, so many people have items that they place in these fields for their own purposes. SU is no exception. There are several parameters used by SU graphics programs that may be stored in these fields. The program **segyclean** zeros out the values of the optional header fields so that SU graphics programs don't become confused by this information.

There are additional issues, such as whether or not your device is buffered or unbufferd (i.e. 9 track 1/2 reel tape, or 8mm Exabyte) tape which may have to be experimented with when you actually try to read a tape. Also, if you are trying to read a tape on a different system than the one it was made on, you may simply not be able to read the tape.

The most common problem with reading tapes is matching the density that the tape was written in, with the tapedrive that the tape is being read on. Some systems, for example Silicon Graphics (SGI) systems, have many tape devices, which support different hardware configurations and tape densities. Other systems, most notably recent versions of Linux have an improved version of the Unix command "mt" which has a "setdensities" option. In either case, it is common for tapes to be made using the default settings of a tape drive, or its default densities.

As a last resort in all tape reading situations, it is often possible to use the Unix device-to-device copying program **dd** to make an image of the entire tape on disk

```
% dd if=/dev/rmtx of=filename bs=32767 conv=noerror
```

where "/dev/rmtx" is replaced with your tapedrive device and "filename" is some file name you choose. If this works, then the next step is to try using **segyread** as above, with "tape=filename." If **dd** fails, then it is likely that the hardware format of your tapedrive is not compatible with your tape.

Of course, the best way to prevent tape reading problems is to make sure that you talk to the person who is writing the tape before they write it. On SGI systems, in particular, there are so many possible choices for the type of tape format, that the person who is making the tape must have information about the platform that the tape is intended to be read on, before they can make a tape that is guaranteed to be readable.

3.1.3 SEG-Y abuses

Unfortunately, there are formats which are called "SEGY" but which are not true to the SEG's standards for SEGY. One common variation is to honor most of the SEGY convention, but have the traces be in an IEEE format.

Such data would be read via:

```
% segyread tape=/dev/rmt0 verbose=1 endian=1 conv=0 | segyclean > data.su
```

where the "conv=0" tells the program not to attempt the IBM to float conversion.

DOS SEGY

There is also a "DOS SEGY" format which is similar to the previous format, with the exception that the traces and headers are all written in a little-endian format. On a big-endian machine, the command to read such a dataset would be

```
% segyread tape=/dev/rmt0 verbose=1 endian=0 conv=0 | segyclean > data.su
```

will read the data. Note, that endian=0 is set to swap the bytes. (All of the bytes, header and data are in the swapped format.) On a little-endian machine, the procedure is

```
% segyread tape=/dev/rmt0 verbose=1 endian=1 conv=0 | segyclean > data.su
```

with endian=1, in this case preventing byteswapping.

In each case, if we had a diskfile with some 'filename', we would use "tape=filename."

Landmark BCM2D format in SEISWORKS

Commercial seismic software can be sources of non-standard SEGY formats. In addition to non-standard usage of the official header fields, commercial variations on SEGY may employ definitions of parts of the optional SEGY header fields.

A remedy for this problem, supplied by Matthias Imhoff of Virginia Tech, is a remapping feature in segyread, which allows nonstandard fields to be remapped into compatible locations in the SU header. The options remap= allows the destination SU header fields to be specified, while byte= specifies the byte location in the SEGY trace header, and its data type.

For the example of Landmark BCM2D format, header fields 73 and 77 are floats, but these are int's in the standard SEGY format and are hence also int's in the SU format. BCM2D also has to header fields set as longs at bytes 181 and 185. The following usage of segyread

```
% segyread tape=... remap=d1,d2,gelev,selev byte=73f,77f,181l,185l > ...
```

The floats at 73 and 77 are mapped to d1 and d2, while the long integers at 181 and 185 are mapped to gelev and selev, which are integers in the SU format. By selecting compatible destination fields, no precision is lost.

3.1.4 SEGYWRITE - Writing an SEGY Tape or Diskfile

The companion program to **segyread** is **segywrite**. This program permits data to be written either to a tape or a diskfile in a number of variations on the SEGY format. This program is useful for putting data into a form that can be read by commercial seismic packages. Before showing examples of how **segywrite** is used, there are several preprocessing steps that must be discussed in preparation for actually writing a tape.

3.1.5 SEGYHDRS - make SEG-Y ascii and binary headers for segywrite

To write a tape in exactly the SEGY format as specified by the SEG's Digital Tape Standards book, you will need to supply the ASCII and binary tape reel header files which will become the EBCDIC and binary tape reel headers in the SEGY tape or file. These are the files "header" and "binary" created by segywrite.

If you don't have the "binary" and "header" files, then you must create them with the program **segyhdrs** (pronounced SEG Y headers) The command

```
% segyhdrs < data.su
```

will write the files "header" and "binary" in the current working directory. As an example, make some test data with **suplane** and then run **segyhdrs** on it

```
% suplane > data.su
% segyhdrs < data.su
```

You will see the files *binary* and *header* appear in your current working directory.

The program has options that will allow you to set the values of binary header fields. These fields may be seen by typing:

```
% sukeyword jobid

        int jobid;      /* job identification number */

        int lino;       /* line number (only one line per reel) */

        int reno;       /* reel number */

        short ntrpr;    /* number of data traces per record */

        short nart;     /* number of auxiliary traces per record */

        unsigned short hdt; /* sample interval in micro secs for this reel */

        unsigned short dto; /* same for original field recording */
...
```

where "jobid" is the first binary header field.

The file "header" is an ASCII file which may be edited with a normal text editor. You can put anything in there, as long as the format is 40 lines of 80 characters each. **Segywrite** will automatically convert this program The default header file created by **segyhdrs** looks like this

```
C      This tape was made at the
C
C      Center for Wave Phenomena
C      Colorado School of Mines
C      Golden, CO, 80401
C
...
C
C
```

3.1.6 BHEDTOPAR, SETBHED - Editing the binary header file

The binary header file must be converted to an ASCII form, to be edited. The program **bhedtopar** permits the "binary" file to be written in the form of a "parfile"

```
% bhedtopar  < binary outpar=binary.par
```

which for the case of the header file created for the test SU data appears as follows

```
jobid=1
lino=1
reno=1
ntrpr=0
nart=0
hdt=4000
...
```

The values that are assigned to the various header files may be edited, and be reloaded into the header file via **setbhed**

```
% setbhed bfile=binary par=binary.par
```

Individual header field values may also be set. For example

```
% setbhed bfile=binary par=binary.par lino=3
```

which uses the contents of binary.par but with the field lino set to 3.

Finally, the tape may be written via a command sequence like this

```
% segywrite tape=/dev/rmtx verbose=1 < data.su
```

taking care to note that the files "header" and "binary" are in the current working directory. You may use different names for these files, if you wish. **Segywrite** has a "bfile=" and an "hfile=" option to permit you to input the files by the different names you choose.

3.1.7 SEGDREAD - Other SEG formats

There are other SEG formats (SEG-A, SEG-B, SEG-X, SEG-C, SEG-D, SEG-1, and SEG-2). Of these, SEG-D, SEG-B, and SEG-2 are the types that you will most commonly encounter. There is a **segdread** program which supports only 1 of the vast number of variations on SEG-D.

In the directory $CWPROOT/Third_Party/ is a **seg2segy** conversion program which may be used to convert SEG-2 format to SEG-Y. Future plans are to create a **seg2read** program, which will be similar to **segyread** and **segdread**.

3.1.8 DT1TOSU - Non-SEG tape formats

Currently, there is only one non-SEG tape format that is completely supported in the SU package, and two others which are supported through third-party codes which have not yet been integrated into the package. This is the Sensors & Software DT1 format, via **dt1tosu**, which is a GPR (ground penetrating radar) format. In the $CWPROOT/src/Third_Party directory are two additional non-SEG conversion programs, these are **segytoseres** and **bison2su**. Future plans include incorporating each of these codes into the main CWP/SU package.

3.2 Data Format conversion

Often, it is necessary to transfer data from other systems, or to input data which may be in a variety of formats. A number of tools and tricks are available in SU for dealing with these issues.

The following programs may be useful for such conversion problems

- A2B - convert ascii floats to binary
- B2A - convert binary floats to ascii
- FTNSTRIP - convert Fortran floats to C-style floats
- FTNUNSTRIP - convert C-style floats to Fortran-style floats
- H2B - convert 8 bit hexidecimal floats to binary
- RECAST - RECAST data type (convert from one data type to another)
- TRANSP - TRANSPose an n1 by n2 element matrix

- FARITH - File ARITHmetic – perform simple arithmetic with binary files
- SUADDHEAD - put headers on bare traces and set the tracl and ns fields
- SUSTRIP - remove the SEGY headers from the traces
- SUPASTE - paste existing SEGY headers on existing data
- SWAPBYTES - SWAP the BYTES of various data types
- SUSWAPBYTES - SWAP the BYTES in SU data to convert data from big endian to little endian byte order, and vice versa

The purpose of this section is to discuss situations where these programs may be used.

3.2.1 A2B and B2A - ASCII to Binary, Binary to ASCII

Of all of the formats of data, the most transportable (and most space consuming) is ASCII. No matter what system you are working on, it is possible to transport ASCII data to and from that system. Also, because text editors support ASCII, it is usually possible to data entry and data editing in the simplest of text editors.

Such data probably come in a multicolumn format, separated either by spaces or tabs. To convert such a, say 5 column, dataset into binary floats, type:

```
% a2b < data.ascii n1=5 > data.binary
```

The reverse operation is

```
% b2a < data.binary n1=5 > data.ascii
```

3.2.2 FTNSTRIP - Importing Fortran Data to C

Often, because Fortran is a popular language in seismic data processing, data may be obtained that was either created or processed in some way with Fortran. Binary data in Fortran are separated by beginning-of-record and end-of-record delimiters. Binary data created by C programs do not have any such delimiters. To use Fortran data in a C program requires that the Fortran labels be stripped off, via

```
ftnstrip < fortdata > cdata
```

This will produce C-style floats, most of the time. The program assumes that each record of fortran data is preceded and followed by an integer listing the size of the record in bytes. There have been fortran data types which have had only one or the other of these integer markers, but having both a beginning-of-record (BOR) and an end-of-record (EOR) markers seems to be standard today.

3.2.3 Going from C to Fortran

It is fairly easy to transport data made with Fortran code to C, however, it may not be so easy to go the other way. On the SGI Power Challenge, it is possible to read a file of C-floats called “infile” via, open and read statements that look like:

```
OPEN(99,file='infile',form='system')

DO i=1,number
READ(99) tempnumber
Array(i)=tempnumber
END DO

CLOSE(99)
```

The statement ”form=’system’” does not work on all machines, however, as it is likely that this is not standard Fortran. The general format command to read in binary is ”form=’unformatted’”. This may not work on other systems, (for example, SUN). Indeed, it may not be generally guaranteed that you can read binary files in Fortran that have been created with a C-programs (as in SU).

If you have problems with binary and the input files are not too big you could convert to ASCII (using ’b2a’) and use formatted I/0

```
OPEN(99,file='infile')

DO i=1,number
READ(99,*) tempnumber
Array(i)=tempnumber
END DO

CLOSE(99)
```

FTNUNSTRIP - convert C binary floats to Fortran style floats

However, another possibility is to make “fake” Fortran floats with a C-program. Such a program is **ftnunstrip**.

This program assumes that the record length is constant throughout the input and output files. In fortran code reading these floats, the following implied do loop syntax would be used: DO i=1,n2 READ (10) (someARRAY(j), j=1,n1) END DO Here n1 is the number of samples per record, n2 is the number of records, 10 is some default file (fort.10, for example) which is opened as form=’unformatted’. Here “someArray(j)” is an array dimensioned to size n1.

Please note that the Fortran style of having BOR and EOR markers is smart, if used properly, but is stupid if used incorrectly. The Fortran READ statement finds out from the BOR marker how many bytes will follow, reads the number of values specified, keeping track of the number of bytes. When it finishes reading, it compares the number of bytes read to the number of bytes listed in the EOR value, and can effectively trap errors in bytes read, or premature EOR.

Because there is an additional “sizeof(int)” (usually 4 bytes) at the beginning and end of every record, it is smart to have as few records as is necessary. The worst case scenario is to have every value of data be a record, meaning that the size of the file could be 2/3 BOR and EOR markers and 1/3 data!

3.2.4 H2B - Importing 8 Bit Hexidecimal

The issue of converting 8 bit hexidecimal may seem to be one that would not come up very often. However 8 bit hex is a common format for bitmapped images (grayscale PostScript) and if you wish to take a scanned image and turn it into floats for further processing, then it will come up.[1]

If you have a scanned image, written as a 256 level grayscale bitmapped PostScript image, then the bitmap portion is in 8 bit hex. By removing all of the PostScript commands, and leaving only the bitmap then the command

```
% h2b < hexdata > floatdata
```

will convert the bitmap into a form that can be viewed and processed by programs in the CWP/SU package.

3.2.5 RECAST - Changing Binary Data Types

Of course, C supports a variety of types, and instead of having a bunch of program to convert each type into every other type, there is a program called “recast” that will do the job for a large collection of these types.

Types supported by recast for input and output:

- float - floating point
- double - double precision
- int - (signed) integer
- char - character
- uchar - unsigned char
- short - short integer
- long - long integer
- ulong - unsigned long integer

For example, to convert integers to floats

```
% recast < data.ints  in=int out=float > data.floats
```

The name of this program derives from the fact that an explicit type conversion in the C-language is called a “cast.”

[1]This issue came up originally, when a student had destroyed an original dataset, by accident, but only had a bit-mapped PostScript image of the data. Using **h2b** it was possible to recover the data from the PostScript file.

3.2.6 TRANSP - Transposing Binary Data

In the course of any of the operations, it is often necessary to transpose datasets. In particular, data which are represented in a multi-column ASCII format, will likely need to be transposed after being converted from ASCII to binary with **a2b**. The reason for this is that it is convenient to have the fast dimension of the data be the time (or depth) dimension for seismic purposes.

Our example above was a 5 column dataset, which you might think of as 5 seismic traces, with the same number of samples in each trace, side by side in the file. The processing sequence

```
% a2b < data.ascii n1=5 > data.binary
```

will result in a file with the fast dimension being in the trace number direction, rather than the number of samples direction The additional processing sequence

```
% transp < data.binary n1=5 > data.transp
```

will put the data in the desirable form of the fast dimension being in the number of samples direction, so that each trace is accessed successively.

3.2.7 FARITH - Performs simple arithmetic on binary data.

It is often necessary to perform arithmetical operations on files, or between two files of binary data. The program **farith** has been provided for many of these tasks.

Some of the tasks for single files supported by **farith** include

- scaling value,
- polarity reversal,
- signum function,
- absolute value,
- exponential,
- logarithm,
- square root,
- square,
- inverse (punctuated),
- inverse of square (punctuated),
- inverse of square root (punctuated).

Binary operations (operations involving two files) include value by value

- addition,
- subtraction,
- multiplication,

- division,
- cartesian product.

Seismic operations (which assume files consist of wavespeeds) include

- slowness perturbation (difference of inverses of files),
- sloth perturbation (difference of inverses of files),

Examples of using **farith** are

```
% farith in=data.binary op=pinv out=data.out.bin
% farith in=data1.binary in2=data2.binary op=add > data.out2.bin
```

3.3 Trace Header Manipulation

The data format of SU inherits the seismic trace headers from SEGY data. If your data are not SEGY, but are created by conversion from some other type, there is a minimum collection of headers that you will need to set, for the data to be compatible with commonly used SU programs.

The issues of interest in this section are:

- adding trace headers,
- removing trace headers,
- pasting trace headers back on,

3.3.1 SUADDHEAD - Adding SU Headers to Binary Data

Once data have been put in the correct form, that is to say, an array of C-style floats organized with the fast dimension in the direction of increasing sample number per trace, then it is necessary to add headers so that these data may be accessible to other SU programs.

If the data are SEGY, SEGD, DT1, or Bison/Geometrics data, read from a tape or diskfile, then the programs **segyread**, **segdread**, **dt1tosu**, or **bison2su** will have set the trace header values so that the output will be "SU data."

For data read from some other means, such as an ASCII dataset by **a2b**, headers have to be added, this is done by using **suaddhead**. If our dataset consists of a file of binary C-style floats with, say 1024 samples per trace, then the command sequence

```
% suaddhead < data.bin ns=1024 > data.su
```

will yield the SU datafile "data.su."

3.3.2 SUSTRIP - Strip SU headers SU data

The inverse operation to **suaddhead** is **sustrip**. Sometimes, you will have data with SU headers on it, but you may want to export the data to another program that does not understand SU headers. The command sequence

```
% sustrip < data.su head=data.headers  >data.bin
```

will remove the SU headers and save them in the file "data.headers."

3.3.3 SUPASTE - Paste SU Headers on to Binary Data

Having performed an operation on the binary data, we may want to paste the headers back on. This is done with **supaste**. The command sequence

```
% supaste < data.bin head=data.headers  >data.su
```

will paste the headers contained in data.headers back on to the data.

3.4 Byte Swapping

Even the best of human intentions can be circumvented. This is the case with the IEEE floating point data format. The implementation of the IEEE standard by chip manufacturers has resulted in two types of commonly encountered data types. These are called "big-endian" (high-byte) or "little-endian" (low-byte) types. Little-endian machines are Intel or Dec-based, whereas big-endian is every other chip manufacturer.

Transporting data (either regular floating point, or SU data) requires that the bytes be swapped, in order for the data to be read. Two issues are necessary to address. These are the issues of swapping the bytes in

- normal floating point data,
- SU data.

Two programs are provided to perform these tasks. These are

- SWAPBYTES - SWAP the BYTES of various data types
- SUSWAPBYTES - SWAP the BYTES in SU data to convert data from big endian

3.4.1 SWAPBYTES - Swap the Bytes of Binary (non-SU) Data

If you transport binary data (data without SU headers) between platforms of a different "endian" (that is to say, the IEEE byte-order), from the one you are working on, then you will have to swap the bytes to make use of that data.

This problem arises because the order of the mantissa and exponent of binary data comes in two different possible order under the IEEE data standard. The so-called "big-endian" or high-byte IEEE format is found on SGI, SUN, IBM RS6000, and all Motorola chip-based systems. The "little-endian" or low-byte systems are systems that are based on Intel and Dec chips.

The program **swapbytes** is provided to do this for a variety of data format types, but not SU data. For example,

```
% swapbytes < data.bin in=float  >data.swap
```

will swap the bytes in a file of containing C-style floating point numbers.

3.4.2 SUSWAPBYTES - Swap the Bytes of SU Data

If you transport SU data (data with SU headers) between systems, you have to swap both the data and the SU headers. The program **suswapbytes** is provided to do this. The command sequence

```
% suswapbytes < data.su format=0 > data.su.swapped
```

will swap data in the SU format from a machine of the reverse endian to the byte order of your system.

The command sequence

```
% suswapbytes < data.su format=1 > data.su.swapped
```

will swap data in the SU format that is in your system's byte order to the opposite byte order. You would use this if you were transporting your SU data from you system to another system of the opposite byte order.

3.5 Setting, Editing, and Viewing Trace Header Fields

Seismic data can have a large number of parameters associated with it. In SU data (following the example of the SEG-Y format) the values of these parameters are stored in the header fields of the traces. There are a number of programs which access the header fields allowing you to set, view, and modify those fields for a variety of purposes.

The tasks of interest are

- adding SU headers,
- stripping SU headers off of data and then pasting them back on,
- identifying SU keywords,
- viewing the range of SU header values,
- setting specified SU header fields,
- computing a third header field from the values of two given fields,
- getting the values of header fields,
- editing specific header fields,

all of which are necessary, and may come under the heading of "geometry setting."

The following list of programs will be discussed in this chapter

- SUADDHEAD - put headers on bare traces and set the tracl and ns fields
- SUSTRIP - remove the SEGY headers from the traces
- SUPASTE - paste existing SEGY headers on existing data
- SUKEYWORD – guide to SU keywords in segy.h
- SURANGE - get max and min values for non-zero header entries
- SUSHW - Set one or more Header Words using trace number, mod and integer divide to compute the header word values or input the header word values from a file
- SUCHW - Change Header Word using one or two header word fields

- SUGETHW - Get the Header Word(s) in SU Data
- SUEDIT - examine segy diskfiles and edit headers
- SUXEDIT - examine segy diskfiles and edit headers

Some of these items were discussed in the previous section, but for completeness, let's recap what these programs do, with a bit more information.

3.5.1 SUADDHEAD - add SU (SEGY-style) Trace Headers

To add headers to a datafile consisting of C-style binary floating point numbers do:

```
% suaddhead < data.bin ns=1024 > data.su
```

or for some other type, such as integers, use **recast**

```
% recast < data.ints in=int out=float | suaddhead ns=1024 > data.su
```

Here we have used a pipe | to cascade the processing flow.
If the data are integers were originally from Fortran, then this is a likely processing flow:

```
% ftnstrip < data.fortran | recast in=int out=float | suaddhead ns=1024 > data.su
```

Other variations are obviously possible.

3.5.2 SUSTRIP and SUPASTE - Strip and Paste SU Headers

```
% sustrip < data.su head=data.head > data.strip
... other processing that doesn't change the number of traces ...
% supaste < datanew.strip head=data.head > datanew.su
```

See the discussion of **sustrip** and **supaste** above.

3.5.3 SUKEYWORD - See SU Keywords

The next 5 programs that are discussed in this section have the option "key=" in their selfdocs and the reference to the trace header field "keywords." As has been discussed in Chapter 2, **sukeyword** is used to determine just what these header keywords are. Typing:

```
% sukeyword -o
```

shows this list. (You may also see this list in Appendix B.) However, of the 80+ fields which are defined in the SU header, only a relatively small subset are used most of the time. These fields are given by the **sukeyword** listing as

```
int tracl; /* trace sequence number within line */
int tracr; /* trace sequence number within reel */
int cdp; /* CDP ensemble number */
int cdpt; /* trace number within CDP ensemble */
 ...
short trid; /* trace identification code:
 ... 1 = seismic data

int offset; /* distance from source point to receiver
```

```
 ...
int  sx; /* X source coordinate */
int  sy; /* Y source coordinate */
int  gx; /* X group coordinate */
int  gy; /* Y group coordinate */
short counit; /* coordinate units code:
 ...
short delrt; /* delay recording time, time in ms between
   initiation time of energy source and time
   when recording of data samples begins
   (for deep water work if recording does not
   start at zero time) */
unsigned short ns; /* number of samples in this trace */
unsigned short dt; /* sample interval; in micro-seconds */

 ...
/* local assignments */
float d1; /* sample spacing for non-seismic data */

float f1; /* first sample location for non-seismic data */

float d2; /* sample spacing between traces */

float f2; /* first trace location */
```

It is a good idea to be aware of this collection when using data derived either from modeling programs or from field datasets.

3.5.4 SURANGE - Get the Range of Header Values

A useful piece of information about trace headers is to see the range of values of the headers in a given dataset. Typing,

```
% surange < data.su
```

will return the ranges of all SU header fields that are nonzero. For example:

```
% suplane | surange
32 traces:
 tracl=(1,32)  tracr=(1,32)  offset=400 ns=64 dt=4000
```

The program **suplane** generates a test pattern whose header values simulate a common-offset dataset. The default parameters for **suplane** are to make 32 traces, with 64 samples per trace at 4 ms (4000 microseconds by SEG convention) sampling, with offset=400. The output consists of three intersecting lines of spikes.

Please note that corrupt data may show really strange values for a large number of the header fields. Detecting such a problem is one of the primary uses of **surange**.

3.5.5 SUGETHW - Get the Values of Header Words in SU Data

Having header fields on data means that there are many additional pieces of information to be kept track of, besides just the seismic data, themselves. If you read data from a SEGY format tape, **segyread** will preserve the trace header information in the main part of the SEGY header.

We saw above that **surange** would permit us to see the min and max header values over an entire dataset. However, we often need to see the values of trace header fields, trace by trace, and in an order that we choose.

The program **sugethw** (pronounced, SU get header word) is just such a utility. For example, the command sequence:

```
% sugethw < data.su key=keyword1,keyword2,... | more
```

Will dump the values of each of the header fields specified by the keywords listed.

A more tangible example, using **suplane** data input via a pipe | is

```
% suplane | sugethw key=tracl,tracr,offset,dt,ns | more
 tracl=1         tracr=1        offset=400          dt=4000         ns=64

 tracl=2         tracr=2        offset=400          dt=4000         ns=64

 tracl=3         tracr=3        offset=400          dt=4000         ns=64

 tracl=4         tracr=4        offset=400          dt=4000         ns=64

 tracl=5         tracr=5        offset=400          dt=4000         ns=64

 tracl=6         tracr=6        offset=400          dt=4000         ns=64

 tracl=7         tracr=7        offset=400          dt=4000         ns=64
...
```

There is no requirement regarding the order of the key words specified, or the number of keywords, as long as at least one is specified.

If, for some reason, you need to dump the values in binary format, an example, again using **suplane** data

```
% suplane | sugethw key=tracl,tracr,offset,dt,ns output=binary >  file.bin
```

outputs the values sequentially in order of keyword given, trace by trace.

For "geometry setting," you may want to use a command sequence (again illustrated by piping suplane data into sugethw)

```
% suplane | sugethw key=tracl,tracr,offset,dt,ns output=geom >  hdrfile
```

The contents of hdrfile may be viewed via

```
% more hdrfile

1 1 400 4000 64
2 2 400 4000 64
3 3 400 4000 64
4 4 400 4000 64
5 5 400 4000 64
...
```

where only the first 5 rows of data have been shown.

3.5.6 SUSHW - Set the Header Words in SU Data

The program **sushw** (pronounced, SU set header word) is an all purpose utility for setting the value of seismic trace headers. This program permits the user to set one or more trace header words. A common use of sushw is to just set a particular field to one value. For example, sometimes data don't have the "dt" field set. Let's say that the data are sampled at 2 ms, By using **sukeyword**, we see that dt is in microseconds

```
% sukeyword dt

...skipping
        unsigned short ns;      /* number of samples in this trace */

        unsigned short dt;      /* sample interval; in micro-seconds */
...
```

This means that the following command sequence will set all dt values to 2000 microseconds

```
% sushw < data.su key=dt a=2000 > data.out.su
```

A more tangible example can be seen by piping **suplane** data into **sushw**

```
% suplane | sushw key=dt a=2000 | sugethw key=dt | more
161 wenzel> suplane | sushw key=dt a=2000 | sugethw key=dt | more

    dt=2000

    dt=2000

    dt=2000

    dt=2000

    dt=2000
...
```

From the selfdoc for **sushw** we see that the following optional parameters are defined

```
 Optional parameters ():
 key=cdp,... header key word(s) to set
 a=0,... value(s) on first trace
 b=0,... increment(s) within group
 c=0,... group increment(s)
 d=0,... trace number shift(s)
 j=ULONG_MAX,ULONG_MAX,... number of elements in group
```

These extra options permit more complicated operations to be performed. This is necessary, because often there is a relationship between header fields and the position of the trace within the dataset. The value of the header field is computed by the following formula

```
  i = itr + d
  val(key) = a + b * (i % j) + c * (i / j)
 where itr is the trace number (first trace has itr=0, NOT 1)
```

where percent % indicates the "modulo" function and / is division.

For example if we want to set the sx field of the first 5 traces to 6400, the second 5 traces to 6300, decrementing by -100 for each 5 trace groups

```
% sushw < data.su key=sx a=6400 c=-100 j=5  > data.new.su
```

Again, piping in **suplane** data into **sushw**

```
% suplane | sushw  key=sx a=6400 c=-100 j=5 | sugethw key=sx | more

    sx=6400

    sx=6400

    sx=6400

    sx=6400

    sx=6400

    sx=6300

    sx=6300

    sx=6300

    sx=6300

    sx=6300

    sx=6200

    sx=6200
...
```

As another example, if we wanted set the "offset" fields of each group of 5 traces to 200,400,...,6400

```
%  sushw  < data.su key=offset a=200 b=200 j=5 > data.out.su
```

As before, piping **suplane** data into **sushw** yields the following

```
% suplane | sushw  key=offset a=200 b=200 j=5 | sugethw key=offset | more

offset=200

offset=400

offset=600

offset=800

offset=1000

offset=200
```

```
offset=400

offset=600

offset=800

offset=1000

offset=200

...
```

We can perform all 3 operations with one call to **sushw**, via:

```
% sushw < data.su key=dt,sx,offset a=2000,6400,200 b=0,0,200 c=0,-100,0 j=0,5,5 > newdata.su
```

Or with **suplane** data piped in

```
% suplane | sushw key=dt,sx,offset a=2000,6400,200 b=0,0,200 c=0,-100,0 j=0,5,5 |
 sugethw key=dt,sx,offset | more

    dt=2000         sx=6400     offset=200

    dt=2000         sx=6400     offset=400

    dt=2000         sx=6400     offset=600

    dt=2000         sx=6400     offset=800

    dt=2000         sx=6400     offset=1000

    dt=2000         sx=6300     offset=200

    dt=2000         sx=6300     offset=400

    dt=2000         sx=6300     offset=600

    dt=2000         sx=6300     offset=800

    dt=2000         sx=6300     offset=1000

    dt=2000         sx=6200     offset=200

    dt=2000         sx=6200     offset=400
...
```

As you can see, it is natural to use pipes and redirects to control job flow, but this becomes ungainly on a single command line. Later in this document, we will see how to construct complicated processing sequences in the controlled environment of shell scripts.

3.5.7 Setting Geometry - Converting Observers' Logs to Trace Headers

There is an often unpleasant task called "setting geometry" which must be performed on field data. Often, a SEGY tape will have only a rudimentary set of header fields set in the data.

The rest of the information (shot location, geophone location, etc...) will be supplied in the form of observers' logs.

For setting geometry, you may wish to dump a specific collection of header fields of interest into a file, read the file into a text editor or spreadsheet program so that you can make changes.

For example, if you have some file "sudata" which has some header fields set incorrectly or incompletely then the following command sequence illustrates a possible way of working with such data. You begin by reading the selected header fields into a file "hdrfile".

```
% sugethw < sudata output=geom key=key1,key2,... > hdrfile
```

Now edit the ASCII file hdrfile with any editor, setting the fields appropriately. Convert hdrfile to a binary format via:

```
% a2b < hdrfile n1=nfields > binary_file
```

were "nfields" is the number of header fields in the "key=.." list above. Then load the new file of header fields via:

```
% sushw < sudata infile=binary_file key=key1,key2,... > sudata.edited
```

Again, "key=key1,key2,..." here is the same list as in the **sugethw** statement above. The finished product is the file sudata.edited.

If you are just beginning to set the header fields, you may build the ASCII header file "hdrfile" any way you want. This could be with your favorite text editor, or with a spreadsheet program. It is not important how the ascii file is created, as long as it is in multi-column ASCII format for the sequence above.

Of course, if you have the header values in a file consisting of C-style floats, (which you can make either from a C-program, or from Fortran data with **ftnstrip**) listed trace-by-trace, then you already have the "binary_file" and need only execute the final sequence.

3.5.8 SUCHW - Change (or Compute) Header Words in SU Data

Some header fields such as "cdp" may be computed from existing header fields. The program **suchw** provides this functionality.

From the selfdoc of **suchw**,

```
...
 key1=cdp,... output key(s)
 key2=cdp,... input key(s)
 key3=cdp,... input key(s)

...
 a=0,... overall shift(s)
 b=1,... scale(s) on first input key(s)
 c=0,... scale on second input key(s)
 d=1,... overall scale(s)
```

we can see that this program uses the values of 2 header fields, key2 and key3, to compute a third, key3, via the equation

```
...
val(key1) = (a + b * val(key2) + c * val(key3)) / d
..
```

For example, to shift the values of the cdp header field by a constant amount, say −1

```
% suchw <data >outdata a=-1
```

or to add a constant amount, say 1000, to a header field, say "tracr,"

```
% suchw key1=tracr key2=tracr a=1000 <infile >outfile
```

Another possible example is that of setting the "gx" field by summing the offset and "sx" (shot point) values using **sushw** and then computing the "cdp" field by averaging the "sx" and "gx." Here, we are using the actual cpp locations as the cdp numbers, instead of the conventional 1, 2, 3, ... enumeration

```
% suchw <indata key1=gx key2=offset key3=sx b=1 c=1 |
% suchw key1=cdp key2=gx key3=sx b=1 c=1 d=2 >outdata
```

It is possible to perform both operations in one call via:

```
%  suchw<indata key1=gx,cdp key2=offset,gx key3=sx,sx b=1,1 c=1,1 d=1,2 >outdata
```

3.5.9 SUEDIT and SUXEDIT - Edit the Header Words in SU Data

Finally, it may be that you wish to examine, and possibly change just few headers. For this purpose, we have **suedit** and **suxedit**. the SU editing programs are executed via:

```
% suedit diskfile  (open for possible header modification if writable)
% suedit <diskfile  (open read only)
```

and permit the interactive viewing and editing of the header fields.

For example, making test data with **suplane**

```
% suplane > data.su
% suedit  data.su
```

yields the following

```
32 traces in input file
 tracl=32 tracr=32 offset=400 ns=64 dt=4000
>                          <------- prompt for interactive use
```

The commands that may be used interactively in **suedit** and **suxedit** may be seen by typing a question mark (?) at the prompt. For example

```
32 traces in input file
 tracl=32 tracr=32 offset=400 ns=64 dt=4000
> ?

 n             read in trace #n
 <CR>          step
 +             next trace;   step -> +1
 -             prev trace;   step -> -1
 dN            adv N traces; step -> N
 %             percentiles
 r             ranks
 p [n1 [n2]]   tabplot
 ! key=val     modify field
 ?             print this file
 q             quit
>
```

This program allows the user to view traces as a tabplot of the data sample values or view or change individual header values

The program **suxedit** is similar to **suedit**, with the addition of X-windows graphics for plotting traces

```
% suxedit diskfile  (open for possible header modification if writable)
% suxedit <diskfile  (open read only)


% suxedit data.su
32 traces in input file
 tracl=32 tracr=32 offset=400 delrt=5 ns=64 dt=4000

> ?

 n              read in trace #n
 <CR>           step
 +              next trace;   step -> +1
 -              prev trace;   step -> -1
 dN             adv N traces; step -> N
 %              percentiles
 r              ranks
 p [n1 [n2]]    tabplot
 g [tr1 tr2] ["opts"]   wiggle plot
 f              wig plot Fourier Transf
 ! key=val      modify field
 ?              print this file
 q              quit
```

Again, the options of this program are largely self-explanatory. Please note that the selfdoc is more informative than the help menu given by typing the question mark "?."

Chapter 4

Viewing SU Data in X-Windows and PostScript

The Seismic Unix package has a small collection of graphics utilities for plotting data, both in general C-style float format, and in SU format, both in the X-windows environment, for screen viewing and in the PostScript form for hard copy.

The types of plotting that are available in SU are

- contour plots,
- gray or colorscale image plots,
- wiggle trace plots,
- line or symbol graphs,
- movies,
- 3D cube plots (PostScript only).

These programs have lengthy selfdocs, reflecting the large number of options for selecting the appearance and labeling of the plots. However, functionality such as windowing data, should be done via the programs **subset** or **suwind** as a preprocessing step before the data are actually sent to the plotting program. These plotting programs are purposely not designed to window data, as raw seismic datasets are often huge.

Also, following the "small is beautiful" philosophy of Unix, we have seprate, but more or less equivalent codes for generating PostScript output for hardcopy plotting.

4.1 X-Windows Plotting Programs

X-windows provides a unified environment for the creation of screen graphics routines, which can be quite portable, provided that the code is written using only the items that can be guaranteed to come with the general distributions of X.

Therefore, all of our X-windows codes are written using straight X calls, or with the X-Toolkit. While coding in such widget sets as Motif is easier in many respects, the code that results is not nearly as portable. There are often great differences between the implementation of commercial widget sets on various platforms.

4.1.1 Plotting General Floating Point Data

The programs that are used for viewing general floating point data (data without SU headers) in the X-window environment are

- XCONTOUR - X CONTOUR Plot of f(x1,x2) via vector plot call,
- XIMAGE - X IMAGE plot of a uniformly-sampled function f(x1,x2),
- XWIGB - X WIGgle-trace plot of f(x1,x2) via Bitmap,
- XGRAPH - X GRAPHer Graphs n[i] pairs of (x,y) coordinates,
- XMOVIE - image one or more frames of a uniformly sampled function f(x1,x2).

Try the following. Make some binary data by stripping the headers off of some SU data. For example

```
% suplane | sustrip > data.bin
n1=64 n2=32 d1=0.004000
nt=64 ntr=32 dt=0.004000
ns=64
```

The items which follow indicate that the dimensions of the data set are “n1=64” by “n2=32.” Now view these data in each of the plotting programs listed above (except **xgraph**) via:

```
% xcontour < data.bin n1=64 n2=32 title="contour"  &
% ximage < data.bin n1=64 n2=32  title="image"  &
% xwigb < data.bin n1=64 n2=32  title="wiggle trace"  &
% xmovie < data.bin n1=64 n2=32  title="movie"  &
```

Please note that the ampersand “&” is a Unix command telling the working shell to run the program in background.

To test **xgraph**, make an ASCII file containing a double column listing of pairs of data to be plotted such as the following

```
1 1
2 1.5
3 3
4 8
10 7
```

Call this file “data.ascii” Then use **a2b** to convert the file to binary and then plot it with **xgraph**

```
% a2b < data.ascii n1=2 > data.bin
n=5
% xgraph < data.bin n=5
```

Note that the “n=5” that is echoed by a2b is the same as the input to xgraph.

When you are finished, and wish to get rid of the windows, you may click on the window and type the letter “q,” for quit. On some systems, you may have to actually select the “destroy” option by clicking and dragging on the square in the upper left hand side of the window frame.

Please note that all of these programs have a huge number of options, reflecting a fairly large collection of functionalities. See the selfdoc of each of these programs by typing

```
% programname
```

or

```
% sudoc programname
```

4.1.2 X-Windows Plotting of SU Data

To plot data that are in the SU format, a number of programs (many of which have parallel functionality to those already discussed) have been created. These are

- SUXCONTOUR - X CONTOUR plot of Seismic UNIX tracefile via vector plot call,
- SUXIMAGE - X-windows IMAGE plot of an SU data set,
- SUXWIGB - X-windows Bit-mapped WIGgle plot of an SU data set,
- SUXGRAPH - X-windows GRAPH plot of an SU data set,
- SUXMOVIE - X MOVIE plot of an SU data set,
- SUXMAX - X-windows graph of the MAX, min, or absolute max value on each trace of an SU data set.

Rather than maintain multiple codes, each of these programs actually calls one or more of the X-windows graphics programs listed in the previous subsection. Please note, that the selfdoc of the non-SU version of a given graphics program also applies to the SU version, meaning that, these programs also have a large functionality.

You may test each of these programs using suplane data via:

```
% suplane | suxcontour title="contour"  &
% suplane | suximage  title="image"  &
% suplane | suxwigb  title="wiggle trace"  &
% suplane | suxgraph  title="graph" &
% suplane | suxmovie  title="movie" &
% suplane | suxmax    title="max"  &
```

Again, note that the ampersand "&" is a Unix command telling the working shell to run the program in background. When you are finished, and wish to get rid of the windows, you may click on the window and type the letter "q," for quit. On some systems, you may have to actually select the "destroy" option by clicking and dragging on the square in the upper left hand side of the window frame.

4.1.3 Special Features of X-Windows Programs

To find out all of the many options of these programs, please see their selfdocs, by typing the names of each of these programs on the commandline:

```
% suxcontour
% suxwigb
% suximage
% suxmovie
% suxmax
```

In addition, the names of the non-SU versions of some of these programs may be typed to yield additional information

```
% xcontour
% xwigb
% ximage
% xmovie
```

However there are some properties that these programs have that can only be illustrated with examples.

Plotting wiggle traces in true offset with SUXWIGB

It is possible to plot wiggle traces in true offset, that is to say, to take the values for the horizontal dimension of the wiggle plot from the values in the header. This is done with the "key=" parameter.

For example, lets make some test data with **suplane** and plot it using the "key=offset" via:

```
% suplane | suchw key1=offset key2=tracl a=0 b=100  | suxwigb key=offset   &
```

The result is a plot with the x2 axis labeled in the values of the offset header field (which count by 100's).

Making a movie with SUXMOVIE

It is possible to make movies of seismic data with **suxmovie**. An example of this is to make several synthetic data panels with **suplane** appending each successive panel with the double redirect sign "¿¿"

```
% suplane > junk1.su
% suplane | suaddnoise sn=20 >> junk1.su
% suplane | suaddnoise sn=15 >> junk1.su
% suplane | suaddnoise sn=10 >> junk1.su
% suplane | suaddnoise sn=5 >> junk1.su
% suplane | suaddnoise sn=3 >> junk1.su
% suplane | suaddnoise sn=2 >> junk1.su
% suplane | suaddnoise sn=1 >> junk1.su

% suxmovie < junk1.su n2=32 title="frame=%g" loop=1  &
```

The final command has "n2=32" set to show that there are 32 traces per frame of data. The usage of "%q" permits the frame number to be listed as part of the title, and "loop=1" runs the movie in a continuous loop.

To make the movie go faster or slower, simply enlarge or shrink the window by clicking and dragging on the lower right corner of the plot. Clicking the far-right mouse button once will freeze the frame, and clicking it a second time will start the movie again.

4.1.4 PostScript Plotting Programs

To complement our collection of X-Windows plotting utilities are a collection of very similar PostScript codes. The idea was to create PostScript plotting codes which would correspond to each of the X-Windows codes listed above.

4.1.5 PostScript Plotting of General Floating Point Data

The programs that are used for PostScript plotting of general floating point data (data without SU headers) are

- PSCONTOUR - PostScript CONTOURing of a two-dimensional function f(x1,x2),
- PSIMAGE - PostScript IMAGE plot of a uniformly-sampled function f(x1,x2),
- PSCUBE - PostScript image plot of a data CUBE,
- PSGRAPH - PostScript GRAPHer Graphs n[i] pairs of (x,y) coordinates,
- PSMOVIE - PostScript MOVIE plot of a uniformly-sampled function f(x1,x2,x3),
- PSWIGB - PostScript WIGgle-trace plot of f(x1,x2) via Bitmap,
- PSWIGP - PSWIGP - PostScript WIGgle-trace plot of f(x1,x2) via Polygons.

Again, you may create binary data to test these programs by stripping off the headers of some suplane data.

```
% suplane | sustrip > data.bin
n1=64 n2=32 d1=0.004000
nt=64 ntr=32 dt=0.004000
ns=64
```

The dimensions of the data are n1=64 samples per trace by n2=32 traces.

```
% pscontour < data.bin n1=64 n2=32 title="contour" > data1.eps
% psimage < data.bin n1=64 n2=32  title="image"  > data2.eps
% pscube < data.bin n1=64 n2=32  title="cube plot"  > data4.eps
% pswigb < data.bin n1=64 n2=32  title="bitmap wiggle trace"  > data3.eps
% pswigp < data.bin n1=64 n2=32  title="wiggle trace"  > data4.eps
% psmovie < data.bin n1=64 n2=32  title="movie"  > data5.eps
```

The output files contain Adobe Level 2 Encapsulated PostScript. You should be able to view these files with any standard X-windows PostScript previewer (such as "ghostview").

Please note, that the output from "psmovie" may not work on your system. This output works under NeXTStep, but is multi-page Encapsulated PostScript, which is not generally supported by PostScript devices.

To test psgraph, make an ascii file containing a double column listing of pairs of data to be plotted such as the following

```
1 1
2 1.5
3 3
4 8
10 7
```

Call this file "data.ascii" Then use "a2b" to convert the file to binary and then plot it with psgraph

```
% a2b < data.ascii n1=2 > data.bin
n=5
% psgraph < data.bin n=5 > data6.eps
```

Note that the "n=5" that is echoed by **a2b** is the same as the input to **psgraph**. This permits the following trick to be used

```
% a2b < data.ascii outpar=junk.par n1=2 > data.bin
% psgraph < data.bin par=junk.par > data6.eps
```

to yield the same output.

4.1.6 PostScript Plotting of SU Data

Just as there are X-Windows codes for plotting SU data, there are also codes for making PostScript plots of these data. The programs for PostScript graphics are

- SUPSCONTOUR - PostScript CONTOUR plot of an SU data set
- SUPSIMAGE - PostScript IMAGE plot of an SU data set
- SUPSCUBE - PostScript CUBE plot of an SU data set
- SUPSGRAPH - PostScript GRAPH plot of an SU data set

- SUPSWIGB - PostScript Bit-mapped WIGgle plot of an SU data set
- SUPSWIGP - PostScript Polygon-filled WIGgle plot of an SU data set
- SUPSMAX - PostScript of the MAX, min, or absolute max value on each trace of a SU data set

We can use **suplane** data to test each of these programs as we did with the X-Windows codes

```
% suplane > junk.su


% supscontour < junk.su title="contour" > data1.eps
% supsimage < junk.su title="image" label1="sec" label2="trace number"  > data2.eps
% supscube < junk.su  title="cube plot"  > data4.eps
% supswigb < junk.su title="bitmap wiggle trace"  > data3.eps
% supswigp < junk.su title="wiggle trace"  > data4.eps
% supsmovie  < junk.su title="movie"  > data5.eps
% supsmax < junk.su title="max"  > data5.eps
```

The output files contain Adobe Level 2 Encapsulated PostScript, and is compatible with TeX, LaTeX, and many draw tools. You will need to use a PostScript previewer, such as GhostScript or Ghostview to view these files on the screen.

Please note again, that programs for which there is a non-SU version, have selfdoc information which applies to these codes, as well.

4.2 Additional PostScript Support

There are several additional tools for supporting PostScript operations in SU. These are

- PSBBOX - change BoundingBOX of existing PostScript file
- PSMERGE - MERGE PostScript files
- MERGE2 - MERGE2 PostScript figures onto one page
- MERGE4 - MERGE4 figures onto one page
- PSLABEL - output PostScript file consisting of a single TEXT string on a specified background. (Use with psmerge to label plots.)
- PSMANAGER - printer MANAGER for HP 4MV and HP 5Si Mx Laserjet PostScript printing
- PSEPSI - add an EPSI formatted preview bitmap to an EPS file

The programs **psbbox**, **pslabel**, **psmerge**, **merge2**, and **merge4** are designed to help in constructing figures made from SU-style graphics programs, and are not guaranteed to work with EPS files generated by other means.

4.2.1 PSBBOX - Changing the BoundingBox

For example, create test PostScript data with **suplane** and **supswigb** via

```
% suplane | supswigb > junk1.eps
```

where the ".eps" extension is chosen as a reminder that this is encapsulated PostScript. Let's say that there is too much white-space surrounding this figure. To fix this problem, we want to change the size of the BoundingBox at the top of the file. For example

```
% more junk1.eps
```

shows the dimensions of the BoundingBox

```
%!PS-Adobe-2.0 EPSF-1.2
%%DocumentFonts:
%%BoundingBox: 13 31 603 746
...
```

We could manually edit this, but with **psbbox** we can type:

```
% psbbox < junk1.eps llx=40 lly=80 urx=590 ury=730 > junk2.eps
Original:  %%BoundingBox: 13 31 603 746
Updated:   %%BoundingBox: 40 80 590 730
```

to yield a smaller BoundingBox, with correspondingly less white space.

4.2.2 PSMERGE, MERGE2, MERGE4 - Merging PostScript Plots

It is often useful to merge several plots to make a compound figure. The program **psmerge** is the general tool in SU provided for this. There are two additions shell scripts, which call **psmerge** called **merge2** and **merge4**. If we make a couple of test datasets

```
% suplane > junk.su
% suplane | sufilter > junk1.su
```

and display these by various means

```
% supswigb < junk.su title="Wiggle trace" label1="sec" label2="trace number" > junk1.eps
% supsimage < junk.su title="Image Plot" label1="sec" label2="trace number" > junk2.eps
% supscontour < junk.su title="Contour Plot" label1="sec" label2="trace number" > junk3.eps
% supswigb < junk1.su title="Filtered" label1="sec" label2="trace number" > junk4.eps
```

we now have 4 PostScript files which can be merged to make new plots.

Merging 2 plots may be done via:

```
% merge2 junk1.eps junk2.eps > junk.m2.eps
```

while merging all 4 plots may be done via

```
% merge2 junk1.eps junk2.eps junk3.eps junk4.eps > junk.m4.eps
```

Of course, neither **merge2** nor **merge4** are robust enough to handle all plot sizes, so you may need to manually merge plots with psmerge. Also, if you want to overlay plots, such as a graph on top of a wiggle trace plot, then you will also need to use **psmerge**.

Here is an example of creating plots and merging them with **psmerge**. Because the command sequence is ungainly for typing on the commandline it is expressed as a shell script

```
#! /bin/sh
# shell script for demonstrating PSMERGE

# make data
suplane > junk.su
suplane | sufilter > junk1.su

# make PostScript Plots of data
supswigb < junk.su wbox=6 hbox=2.5 \
 title="Wiggle trace" label1="sec" label2="traces" > junk1.eps
```

```
supscontour < junk.su wbox=2.5 hbox=2.5  \
 title="Contour Plot" label1="sec" label2="traces" > junk3.eps
supswigp < junk1.su wbox=2.5 hbox=2.5 \
 title="Filtered" label2="traces" > junk4.eps

# merge PostScript plots
psmerge in=junk1.eps translate=0.,0. \
 in=junk3.eps translate=0.0,3.7 \
 in=junk4.eps translate=3.3,3.7 > junk5.eps

echo "You may view the files: junk1.eps, junk3.eps, junk4.eps, junk5.eps"
echo "with your PostScript Previewer"

exit 0
```

In this case, the original files were made small to fit within an 8-1/2" by 11" window. However, **psmerge** has the capability of scaling plots. (This is how the **merge2** and **merge4** shells work. You can examine the texts of these for further information by typing

```
% more $CWPROOT/bin/merge2
or
% more $CWPROOT/bin/merge4
```

An additional example of merging 3 plots of different sizes is given by the following shell script

```
#! /bin/sh
# shell script for demonstrating PSMERGE

# make data
suplane > junk.su
suplane | sufilter > junk1.su

# make PostScript Plots of data
supswigb < junk.su wbox=7 hbox=4 \
 title="Wiggle trace" label1="sec" label2="traces" > junk1.eps
supscontour < junk.su \
 title="Contour Plot" label1="sec" label2="traces" > junk3.eps
supswigp < junk1.su label1="sec" \
 title="Filtered" label2="traces" > junk4.eps

# merge PostScript plots
psmerge in=junk1.eps translate=0.,0. scale=.6,.6 \
 in=junk3.eps scale=.4,.4 translate=0.0,3.7 \
 in=junk4.eps scale=.4,.4 translate=3.3,3.7 > junk5.eps

echo "You may view the files: junk1.eps, junk3.eps, junk4.eps, junk5.eps"
echo "with your PostScript Previewer"

exit 0
```

In this case, the plots are of normal size, and are then scaled to fit within an 8-1/2" by 11" window.

4.3 Trace Picking Utilities

For lack of a better location to discuss this is the subject, we will now list“trace picking” utilities. The X-Windows wiggle trace, image, and contour plotting programs all have the attribute that by placing the cursor on the point to be picked and typing the letter ‘s’ the coordinates of the point are saved in memory. When the letter ‘q’ is typed, the values are saved in a user-specified file “mpicks.”

There are two additional programs which are dedicated to picking issues. These are

- XPICKER - X wiggle-trace plot of f(x1,x2) via Bitmap with PICKing
- SUXPICKER - X-windows WIGgle plot PICKER of an SU data set
- SUPICKAMP - pick amplitudes within user defined and resampled window

with “xpicker” being the non-SU data version which “suxpicker” calls. The xpicker/suxpicker programs are interactive tools for picking. The program “supickamp” is a simple automated picking program, which seeks the largest amplituded within a user-specified window.

See demos in $CWPROOT/src/demos/Picking for more information about “supickamp.”

4.4 Editing SU Data

Once the data are read in, and the headers are set properly, then there will often be manipulation and data editing issues to be dealt with.

It is often necessary to perform tasks of varying so that data may be

- windowed,
- sorted,
- truncated,
- tapered,
- zeroed,
- made uniform in number of samples,
- concatenated,

which are dataset editing issues.

This section, therefore will deal with the following programs

- SUWIND - window traces by key word
- SUSORT - sort on any segy header keywords
- SURAMP - Linearly taper the start and/or end of traces to zero.
- SUTAPER - Taper the edge traces of a data panel to zero.
- SUNULL - create null (all zeroes) traces
- SUZERO – zero-out data within a time window
- SUKILL - zero out traces
- SUMUTE - mute above (or below) a user-defined polygonal curve with the distance along the curve specified by key header word
- SUVLENGTH - Adjust variable length traces to common length
- SUVCAT - append one data set to another (trace by trace)

which provide a variety of trace editing utilities.

4.4.1 SUWIND - window traces by key word

It is very common to view or process only a subset of a seismic dataset. Because seismic data have a number of parameters that we may want to window the data about, “suwind” has been written.

Windowing by trace header field

In its simplist usage, “suwind” permits the user to set min and max values of a specific header field

```
key=tracl       Key header word to window on (see segy.h)
min=LONG_MIN    min value of key header word to pass
max=LONG_MAX    max value of key header word to pass
```

For example, windowing suplane data by trace number yields

```
% suplane  | suwind key=tracl  min=5 max=10 | sugethw key=tracl | more

 tracl=5

 tracl=6

 tracl=7

 tracl=8

 tracl=9

 tracl=10
```

On a large dataset, the “count” parameter should be used, instead of setting the max value. If you set an explicit “max” value, suwind will have to go through the entire dataset to capture all possible traces with values between the min and max value, because the program assumes that multiple occurrences of trace labeling are possible. For example, compare

```
% suplane ntr=100000 | suwind key=tracl min=5 max=10 | sugethw tracl | more
```

(it’s ok to type “control-c” after a few minutes) with

```
%suplane ntr=100000 | suwind key=tracl min=5 count=5 | sugethw tracl | more
```

where suplane has been set to create 100000 traces in each case.

More sophisticated windowing, (decimating data, for example)

```
j=1             Pass every j-th trace ...
s=0             ... based at s  (if ((key - s)%j) == 0)
```

can be illustrated with suplane data by showing every 2nd trace

```
% suplane  | suwind key=tracl j=2 | sugethw key=tracl | more

 tracl=2

 tracl=4

 tracl=6
 tracl=8

 tracl=10
...
```

or by every 2nd trace, based at 1

```
% suplane  | suwind key=tracl j=2 s=1 | sugethw key=tracl | more

 tracl=1

 tracl=3

 tracl=5

 tracl=7

 tracl=9

...
```

Accepting and rejecting traces is also possible with "suwind."

```
...
       reject=none     Skip traces with specified key values
...    accept=none     Pass traces with specified key values(see notes)
```

The reject parameter is a straightforward rejecting of numbered traces. For example

```
suplane | suwind key=tracl reject=3,8,9 | sugethw key=tracl | more
 tracl=1

 tracl=2

 tracl=4

 tracl=5

 tracl=6

 tracl=7

 tracl=10

 tracl=11

 tracl=12
```

traces 3, 8, and 9 are rejected.

The accept option is a bit strange–it does *not* mean accept *only* the traces on the accept list! It means accept these traces, even if they would otherwise be rejected. For example:

```
suplane | suwind key=tracl reject=3,8,9 accept=8 | sugethw key=tracl
| more
```

```
 tracl=1

 tracl=2

 tracl=4

 tracl=5

 tracl=6

 tracl=7

 tracl=8

 tracl=10

 tracl=11
....
```

If you want to *accept only* the traces listed, then you need to set "max=0"

```
% suplane | suwind key=tracl accept=8 max=0 | sugethw key=tracl | more
 tracl=8
```

Only trace 8 is passed in this example.

The count parameter overrides the accept parameter, so you can't specify count if you want true unconditional acceptance.

See the demos in $CWPROOT/src/demos/Selecting_Traces for specific examples.

Time gating

The second issue windowing is time gating. In fact, when people want to window data, they often want to do both trace and time gating.

```
 Options for vertical windowing (time gating):
       tmin = 0.0             min time to pass
       tmax = (from header)   max time to pass
       itmin = 0              min time sample to pass
       itmax = (from header)  max time sample to pass
       nt = itmax-itmin+1     number of time samples to pass
```

The result of setting either "itmin and itmax" or "tmin and tmax" will be to create a time gate which is to the nearest sample. If you want to time gate to an arbitrary (inter-sample) window, then this is a problem in data resampling and "suresamp" is the program of choice.

4.4.2 SUSORT - sort on any SEGY header keywords

One of the advantages of working in the Unix operating system is that when a superior Unix system call for a particular operation exists, it is advantageous to use that call to perform the necessary task. Such a task is sorting, and the Unix "sort" command is just such a utility.

Susort takes advantage of the Unix system sort command to permit the sorting of traces by header field key work.

For example, sorting data (with values in ascending order) for two fields (cdp and offset) would be done via:

```
% susort <indata.su >outdata.su cdp offset
```

In decending order for, offset, and ascending order for cdp

```
% susort <indata.su >outdata.su cdp -offset
```

Note: Only the following types of input/output are supported Disk input to any output, but pipe input to disk output.

Please see the demo in $CWPROOT/src/demos/Sorting_Traces for specific examples of trace sorting.

4.4.3 SURAMP and SUTAPER - tapering data values

Many seismic processing algorithms will show spurious artifacts resulting from sharp edges on a dataset. Tapering the amplitudes on the edges of a dataset is the one of the easiest ways of suppressing these artifacts. For this purpose, we have "sutaper" to taper the edges of the dataset (for example here, linear taper over 5 traces on each end of the dataset)

```
% sutaper <diskfile >stdout ntaper=5
```

and "suramp" to smooth the beginning and/or end of traces (for the example here, ramp up from 0 to tmin=.05 seconds, and ramp down from tmax=1.15 seconds to the end of the traces

```
% suramp <diskfile tmin=.05 tmax=1.15 >stdout
```

4.4.4 SUKILL, SUZERO, SUNULL, SUMUTE - zeroing out data

It is often useful to zero out noisy traces, or traces on the edge of a dataset (abrupt analog to tapering), or to create null traces as separators to be used between successive panels of data in plotting.

SUKILL - zero out traces

To zero out a block of traces type,

```
% sukill <stdin >stdout min=MIN_TRACE count=COUNT
```

where COUNT is the number of traces to be zeroed, and MIN_TRACE is the minimum trace number in the block of traces.

SUNULL - Create a Panel of Empty traces

It is sometimes necessary to create a panel of zero value traces. To create a panel of traces of NTR traces with NT time samples

```
% sunull nt=NT ntr=NTR <stdin >stdout min=MIN_TRACE count=COUNT
```

SUZERO – zero-out data within a time window

To zero out data within a time window, use "suzero"

```
% suzero itmin=MIN_TIME_SAMPLE itmax=MAX_TIME_SAMPLE <indata.su > outdata.su
```

SUMUTE - Surgically Muting Data

The last collection of programs perform a form of crude muting of the data. For a more precise muting operation, there is "sumute" which can perform surgical muting of SU data. The program may be used by specifying trace header field to mute on via the "key=" parameter,

```
% sumute <indata.su >outdata.su key=KEYWORD xmute=x1,x2,x3,... tmute=t1,t2,t3,...
```

An suplane data example of this is to compare original suplane data, made by

```
% suplane | suxwigb &
```

with surgically muted data

```
% suplane | sumute key=tracl xmute=1,10,12 tmute=.06,.1,.11 | suxwigb &
```

This says to mute every arrival above the polygonal curve defined by the curve defined by the xmute= and tmute= values.

The program also permits the x,t values to be input from binary files by setting the options "nmute, xfile, and tfile" as listed in a portion of the selfdoc for this program

```
...
 nmute= number of x,t values defining mute
 xfile= file containing position values as specified by
  the ‘key’ parameter
 tfile= file containing corresponding time values (sec)
...
  =tracl  use trace number instead
 ntaper=0 number of points to taper before hard
mute (sine squared taper)
 below=0 =1 to zero BELOW the polygonal curve
...
```

Please note also that "above" and "below" refer to the appearance on a seismic plot, such as the suxwigb plot and not to the time values.

4.4.5 SUVCAT and CAT - Concatenating Data

There are two possible ways of appending one dataset onto another (concatenating). The first way would be to append one dataset to another, so that the traces from the second file simply follow the traces of the first file. This is done with the Unix "cat" command via:

```
% cat data1.su data2.su > data3.su
```

In addition, it may be necessary to renumber the traces via:

```
% cat data1.su data2.su | sushw key=tracl a=1 > data3.su
```

so that the tracl parameter is monotonically increasing.

The second way of appending one dataset to another is to "vertically" append each trace of the second dataset to the end of the first dataset. This is done via "suvcat."

```
% suvcat data1.su data2.su > data3.su
```

In this case, no modification of the header fields should be necessary.

4.4.6 SUVLENGTH - Adjust Variable Length Traces to a Common Number of Samples

Sometimes the data consists of traces which have different numbers of samples on each trace. The program "suvlength" We can construct an example of this with suplane data

```
% suplane nt=64 > data1.su
% suplane nt=32 > data2.su
% cat data1.su data2.su > data3.su
```

Attempts to use any SU program on the resulting file "data3.su" will probably result in failure, because most SU programs require that the number of samples be constant on a panel of data. Applying "suvlength" fixes this problem

```
% suvlength ns=64 < data3.su > data4.su
% suxwigb < data4.su title="Test of suvlength"  &
```

by making all of the traces the same length.

Chapter 5

General Operations on SU Data

Beyond the task of editing SU data are operations level codes which perform

- gaining,
- resampling,
- unary operations (arithmetic operations involving a single data file),
- binary operations (arithmetic operations involving two data files),

The purpose of this section, therefore will be to discuss the programs

- SUADDNOISE - add noise to traces,
- SUGAIN - apply various types of gain to display traces,
- SUOP - do unary arithmetic operation on segys,
- SUOP2 - do a binary operation on two data sets,

that perform these operations

5.0.7 SUADDNOISE - Add noise to SU data

While it may seem that having a program that *adds* noise to seismic data is counterproductive, it is often useful for testing purposes to have the capability of simulating noise.

It is also useful for demonstration purposes, and many of the demos below use suaddnoise to "fill in the blanks" in the suplane testpattern. A couple of examples of the output of this program, using suplane data

```
% suplane | suxwigb title="no noise" &
% suplane | suaddnoise | suxwigb title="noise added" &
% suplane | suaddnoise sn=2 | suxwigb title="noise added" &
```

5.0.8 SUGAIN - Gaining to SU data

There are numerous operations which come under the heading of gaining, which "sugain" performs. These operations include

- scaling the data,
- multiplying the data by a power of time,
- taking the power of the data,

- automatic gain control,
- trapping noise spiked traces,
- clipping specified amplitudes or quantiles,
- balancing traces by quantile clip, rms value, or mean,
- scaling the data,
- biasing or debiasing the data.

The heirarchy of the operations is stated by the following equation

```
out(t) = scale * BAL{CLIP[AGC{[t^tpow * exp(epow * t) * ( in(t)-bias )]^gpow}]}
```

You may see what sugain does by running the following examples using suplane data. Noise has been added with "suaddnoise" to make the affects of AGC apparent. Type only the items following the percent %. Create some SU data (with noise added, via)

```
% suplane | suaddnoise > data.su


% suxwigb < data.su title="Ungained Data"  &
% sugain < data.su scale=5.0 | suxwigb title="Scaled data"  &
% sugain < data.su agc=1 wagc=.01 | suxwigb title="AGC=1 WAGC=.01 sec &
% sugain < data.su agc=1 wagc=.2 | suxwigb title="AGC=1 WAGC=.1 sec &
% sugain < data.su pbal=1 | suxwigb title="traces balanced by rms" &
% sugain < data.su qbal=1 | suxwigb title="traces balanced by quantile" &
% sugain < data.su mbal=1 | suxwigb title="traces balanced by mean" &
% sugain < data.su tpow=2 | suxwigb title="t squared factor applied" &
% sugain < data.su tpow=.5 | suxwigb title="square root t factor applied" &
```

Please note, on your terminal window, there will be a message with "clip=" some number, for example:

```
xwigb: clip=1
```

This indicates the amplitude value above which traces are clipped. You may think of this as the value of the maximum on the trace.

5.0.9 SUOP - Unary Arithmetic Operations on SU Data

Occassionally we want to apply mathematical functions or other operations which go beyond gaining to data. Such operations might include

- absolute value,
- signed square root,
- square,
- signed square,
- signum function,
- exponential,
- natural logarithm,
- signed common logarithm,
- cosine,

- sine,
- tangent,
- hyperbolic cosine,
- hyperbolic sine,
- hyperbolic tangent,
- divide trace by Max. Value,
- express trace values in decibels: 20 * slog10 (data)
- negate values,
- pass only positive values,
- pass only negative values.

Operations involving logarithms are "punctuated" meaning that if the contains 0 values, 0 values are returned.

Examples of suop using SU plane data may be easily run

```
% suplane | suaddnoise > data.su
% suop < data.su op=abs | suxwigb title="absolute value" &
% suop < data.su op=ssqrt | suxwigb title="signed square root" &
% suop < data.su op=sqr | suxwigb title="signed square" &
...
```

Please type:

```
% suop
```

to see the selfdoc and the other options.

5.0.10 SUOP2 - Binary Operations with SU data

To perform operations two SU datasets, the progam "suop2" has been provided. Some of the opperations supported are to compute the

- difference,
- sum,
- product,
- quotient,
- difference of a panel and a single trace,
- sum of a panel and a single trace,
- product of a panel and a single trace,
- quotient of a panel and a single trace.

The first 4 options assume that there are the same number of traces in each SU datafile. In the last 4, it is assumed that there is only a single trace in the second file.

From the selfdoc of "suop2" please note that there are 8 equivalent shell scripts commands which perform these operations

```
...
  susum file1 file2 == suop2 file1 file2 op=sum
  sudiff file1 file2 == suop2 file1 file2 op=diff
  suprod file1 file2 == suop2 file1 file2 op=prod
  suquo  file1 file2 == suop2 file1 file2 op=quo

 For:  panel "op" trace  operations:
  suptsum  file1 file2 == suop2 file1 file2 op=ptsum
  suptdiff file1 file2 == suop2 file1 file2 op=ptdiff
  suptprod file1 file2 == suop2 file1 file2 op=ptprod
  suptquo  file1 file2 == suop2 file1 file2 op=ptquo
...
```

All of these call "suop2" to perform the computation.

Try the following. Make two files of SU data with "suplane"

```
% suplane > junk1.su
% suxwigb < junk1.su | suxwigb title="Data without noise" &
% suplane | suaddnoise > junk2.su
% suxwigb < junk2.su  | suxwigb title="Data with noise added" &
% suop2 junk2.su junk1.su op=diff | suxwigb title="difference" &
```

Note, that the filenames must appear before the "op=."

5.1 Transform and Filtering Operations

A major aspect of seismic research and seismic processing is related to operations which are based on mathematical *transform* methods. In particular, much of seismic processing would not exist without numerical Fourier transforms. Filtering is a related subject, because the majority of filters are applied in the frequency domain, or are at least representable mathematically as frequency domain operations.

In addition to standard Fourier transforms, there are a couple of other transforms, such as the Hilbert transform, and the Gabor transform which are also included in the SU package. These items may find more use as educational tools, rather than processing tools.

5.1.1 Fourier Transform Operations

There are Fourier transform operations for both 1D and 2D applications in the SU package. The 1D transforms provide spectral (either amplitude or phase) information about each trace in a panel of seismic data.

The 2D transforms include the seismic F-K variety, assuming that the fast dimension of the input data is temporal, and the second dimension is spatial, and the non-seismic K1-K2 variety, in which the input is assumed to be purely spatial (x1,x2) data.

5.1.2 1D Fourier Transforms

- SUFFT - fft real time traces to complex frequency traces
- SUIFFT - fft complex frequency traces to real time traces
- SUAMP - output amp, phase, real or imag trace from (frequency, x)
- SUSPECFX - Fourier SPECtrum (T to F) of traces domain data

The program “sufft” produces the output of the Fourier transform operation as a complex data type. The program “suifft” is designed to accept complex input as would be generated by sufft to perform the inverse Fourier transform. The cascade of these operations is not quite an non-operation, because there is zeropadding which is automatically implemented for the transforms. Also, the header fields will not quite be right for seismic data. For example, try:

```
% suplane | suxwigb title="Original Data"  &
% suplane | sufft | suifft | sushw key=d1,dt a=0,4000 | suxwigb  &
```

The result is the same as the input, except there are more samples on the traces due to zero-padding required for the transform.

To view the amplitude and phase spectra, and the real and imaginary parts of the of the output of sufft, do the following

```
% suplane | sufft | suamp mode=amp | suxwigb title="amplitude" &
% suplane | sufft | suamp mode=phase | suxwigb title="phases" &
% suplane | sufft | suamp mode=real | suxwigb title="real" &
% suplane | sufft | suamp mode=imag | suxwigb title="imaginary" &
```

SU data has a format which allows for the storage of the real and imaginary parts of data in a complex datatype. To see the header field settings for that format, type:

```
% suplane | sufft | surange
sufft: d1=3.571428
32 traces:
 tracl=(1,32)  tracr=(1,32)  trid=11 offset=400 ns=72
 dt=4000 d1=3.571428
```

You will notice that the setting for the trace id (trid) is 11. Typing:

```
% sukeyword trid
...
                        11 = Fourier transformed - unpacked Nyquist
                             xr[0],xi[0],...,xr[N/2],xi[N/2]
...
```

shows that trid=11 how the data are arranged in the output of the fft.

Of course, most of the time, we only want to have a quick look at the amplitude spectrum of a seismic trace, or a panel of seismic traces. For these purposes use “suspecfx”

```
% suplane | suspecfx | suximage title="F-X Amplitude Spectrum"  &
```

which directly displays the amplitude spectra of each trace of the input SU data.

5.1.3 2D Fourier Transforms

Seismic data are generally at least 2D datasets. If the data really are data in (time, space) coordinates, then the output should be in (frequency, wavenumber), the F-K domain. However, there are applications where we consider the data to be in two spatial dimensions (x1,x2) meaning that the output is in (k1,k2)

We have the programs

- SUSPECFK - F-K Fourier SPECtrum of data set
- SUSPECK1K2 - 2D (K1,K2) Fourier SPECtrum of (x1,x2) data set

For each of these cases. Examples using suplane data are easily run using

```
% suplane | suspecfk | suximage title="F-K Amplitude Spectrum"  &
% suplane | suspeck1k2 | suximage title="K1-K2 Amplitude Spectrum"  &
```

Please note, as these are *display* programs, so the intent is that plots of the output should appear correct when compared with a plot of the original data. The effect of the 2D Fourier transform on a line of spikes is to produce a line of spikes normal to the original line, if the (k1,k2) data are plotted on the same plot as the (x1,x2) data,

5.2 Hilbert Transform, Trace Attributes, and Time-Frequency Domain

A number of classical techniques for representing instantaneous trace attributes have been created over the years. Many of these techniques involve the construction of a quadrature trace to be used as the imaginary part of a "complex trace" (with the real part, being the real data). The quadrature trace is created with the Hilbert transform following the construction of the so-called "allied function." This representation permits "instantaneous amplitude, phase, and frequency" information to be generated for a dataset, by taking the modulus, phase, and time derivative of the phase, respectively. An alternate approach is to perform multi-filter analysis on data, to represent it as a function of both time and frequency.

Tools in SU which perform these operations are

- SUHILB - Hilbert transform
- SUATTRIBUTES - trace ATTRIBUTES instantanteous amplitude, phase, or frequency
- SUGABOR - Outputs a time-frequency representation of seismic data via the Gabor transform-like multifilter analysis,

To generate the Hilbert transform of a test dataset, try

```
% suplane | suhilb | suxwigb title="Hilbert Transform"  &
```

This program is useful for instructional and testing purposes.

To see an example of "trace attributes" with "suattributes" it is necessary to make data which has a strong time-frequency variability. This is done here with "suvibro" which makes a synthetic vibroseis sweep. Compare the following:

```
% suvibro | suxgraph title="Vibroseis sweep" &
% suvibro | suattributes mode=amp | suxgraph title="Inst. amplitude"  &
% suvibro | suattributes mode=phase unwrap=1.0 | suxgraph title="Inst. phase"  &
% suvibro | suattributes mode=freq | suxgraph title="Inst. frequency"  &
```

which show, respectively, instantaneous amplitude, phase, and frequency.

To see the synthetic vibroseis trace in the time-frequency domain, try the following:

```
% suvibro | sugabor | suximage title="time frequency plot"  &
```

The result is an image which shows the instantaneous or apparent frequency increasing from 10hz to 60hz, with time, exactly as stated by the default parameters of "suvibro."

See the demos in $CWPROOT/src/demos/Time_Freq_Analysis and $CWPROOT/src/demos/Filtering/Sugabor for further information.

5.3 Radon Transform - Tau_P Filtering

The Radon or "tau-p" transform, as it is sometimes called in geophysical literature is useful for a variety of multiple suppression, and other "surgical" data manipulation tasks. The programs

- SUTAUP - forwared and inverse T-X and F-K global slant stacks
- SUHARLAN - signal-noise separation by the invertible linear transformation method of Harlan, 1984
- SURADON - compute forward or reverse Radon transform or remove multiples by using the parabolic Radon transform to estimate multiples and subtract.
- SUINTERP - interpolate traces using automatic event picking

are provided to make use of this transform. You may tests using suplane data as we have with other programs to see the output from each of these codes. Both "suinterp" and "suradon" have some sophisticated options which my require some experimentation.

See also the demos in $CWPROOT/src/demos/Tau_P for examples.

5.4 1D Filtering Operations

A large part of what is called seismic processing, may be thought of as being "filtering." There are filtering operations for 1D applications in the SU package that span simple filtering tasks, to more sophisticated tasks including deconvolution and wavelet shaping operations. These operations are 1D, in that they are applied trace by trace.

Several types of filtering operations that arise in seismic processing are

- zero bandpass, bandreject, lowpass, and highpass , and notch filtering,
- minimum or zero phase Butterworth filtering,
- Wiener prediction error (deconvolution),
- Wiener wavelets shaping,
- convolution,
- crosscorrelation,
- autocorrelation,
- data resampling with sinc interpolation,
- fractional derivatives/integrals,
- median filtering,
- time varying filtering.

The programs that are provided to meet these needs are:

- SUFILTER - applies a zero-phase, sine-squared tapered filter
- SUBFILT - apply Butterworth bandpass filter
- SUACOR - auto-correlation
- SUCONV, SUXCOR - convolution, correlation with a user-supplied filter
- SUPEF - Wiener predictive error filtering
- SUSHAPE - Wiener shaping filter
- SURESAMP - Resample in time,

- SUFRAC – take general (fractional) time derivative or integral of data, plus a phase shift. Input is TIME DOMAIN data.
- SUMEDIAN - MEDIAN filter about a user-defined polygonal curve with the distance along the curve specified by key header word
- SUTVBAND - time-variant bandpass filter (sine-squared taper)

5.4.1 SUFILTER - applies a zero-phase, sine-squared tapered filter

The program **sufilter** provides a general purpose zero-phase filtering capability for the usual tasks of bandpass, bandreject, lowpass, highpass, and notch filtering. Examples of each of these using **sufilter** data are provided

```
% suplane | sufilter f=10,20,30,60 amps=0,1,1,0 | suxwigb title="10,20,30,60 hz bandpass"  &


% suplane | sufilter f=10,20,30,60 amps=1,0,0,1 | suxwigb title="10,20,30,60 hz bandreject"  &


% suplane | sufilter f=10,20,30,60 amps=1,1,0,0 | suxwigb title="10,20 hz lowpass"  &


% suplane | sufilter f=50,60,70 amps=1,0,1 | suxwigb title="60 hz notch"  &
```

The filter is polygonal, with the corners of the polygon defined by the vector of frequency values defined by array of **f=** values, and a collection of amplitude values, defined by the array of **amps=** values. The amplitudes may be any floating point numbers greater than, or equal to zero. The only rule is that there must be the same number of **amps** values, as **f** values. The segments are sine-squared tapered between **f** values of different amplitude. It is best to select bf f values such that tapering to zero is done over an octave to prevent ringing.

With **suplane**, **sufilter** provides a way of easily generating bandlimited testpattern data in SU format.

5.4.2 SUBFILT - apply Butterworth bandpass filter

An alternative to **sufilter** is **subfilt**, which applies a Butterworth filter to data.

```
% suplane | subfilt fstoplo=10 fpasslo=20 fpasshi=30 fstophi=60 | suxwigb title="10,20,30,60 hz  ba



\subsection{SUACOR - auto-correlation}
This program is used to compute the autocorrelation of a trace.
This process is useful to see the size of a wavelet, or the repetitions
of the wavelet, as an aid in choosing the {\bf maxlag\/} parameter
of {\bf supef}. {\bf Suacor\/} is also useful for determining the
frequency range of data in terms of the power spectrum. For example
try
{\small \begin{verbatim}
% suplane | sufilter | suacor | suspecfx | suxwigb &
```

5.4.3 SUCONV, SUXCOR - convolution, correlation with a user-supplied filter

The standard operations of convolution and cross correlation may be performed with **suconv** and **suxcor**, respectively. The filter may be supplied as a vector input on the commandline, or as a file containing a single trace in SU format. In addition to taking the input as a single trace, it is possible to supply a panel of filters, each to be used trace by trace on a panel of SU data.

An example of correlating a vibroseis sweep may be seen by creating vibroseis-like data, with **suvibro**, **suplane** and **suconv**. To make "vibroseis" **suplane** data

```
% suvibro > junk.vib.su
% suplane | suconv sufile=junk.vib.su > plane.vib.su
```

Because **surange** tells us that

```
% surange < junk.vib.su
1 traces:
 tracl=1 ns=2500 dt=4000 sfs=10 sfe=60
 slen=10000 styp=1
```

there are 2500 samples on the vibroseis sweep, we can do the following correlation

```
%  suxcor < plane.vib.su sufile=junk.vib.su |
     suwind itmin=2500 itmax=2563 | sushw key=delrt a=0.0 > data.su
(this line is broken to make it fit on the page here, the real
command is typed on a single line)
```

The value of **itmin=sweeplength** and **itmax=sweeplength+nsout** where **nsout** is the number of samples expected in the output. The final step using **sushw** is to set the trace delay to 0. Choosing **itmin=sweeplength** will ensure that the data start at the correct value. Choosing **nsout=sweeplength-nsin**, where **nsin** is the number of samples in the input, will yield the correct number of samples to keep.

5.4.4 SUPEF - Wiener predictive error filtering

The prediction error filtering method, also known as Wiener filtering, is the principle process of traditional Wiener-Levinson deconvolution. The reason that this subsection is not headed "deconvolution" is because there are two additional issues that have to be addressed before prediction error filtering can be used for effective deconvolution. These issues are preprocessing of the data, and postprocess filtering. Indeed, much feedback has come back claiming that **supef** doesn't work properly, when in fact, it simply being used improperly. (This is our fault, for not supplying sufficient documentation.)

Using **supef**, itself, generally requires that the value of **maxlag** be set. This may be determined by first establishing the size of the wavelet being spiked, through application **suacor**.

As the preprocessing step, you will probably need to use **sugain** to remove any decay in amplitudes with time that may result from geometric spreading.

As a postprocessing step, you need to remove any increase in frequency content which has occurred due to the whitening effect of the prediction error filter. The demos in the demos/Deconvolution directory demonstrate the functioning of the program. However, these examples are a bit unrealistic for field data. For example, in the demos, the data are spiked, and then reverberations are removed via a cascaded of **supef** calls with **maxlag=.04** and **minlag=.05 maxlag=.16**, respectively. However on field data, we would probably only use a single pass of the filter to remove reverberations, or to spike arrivals.

The prediction error filter has a spectral whitening effect, which is likely to put high frequencies in the data that were not there originally. These must be filtered out. It is a good idea to assume that there is a *loss* of frequency information, and design the parameters for the postprocessing filtering, using **sufilter** to slightly reduce the frequency content from its original values.

Also, an added feature of the Release 32 version of **supef** is the mixing parameter, which permits the user to apply a weighted moving average to the autocorrelations that are computed as part of the prediction error filter computation. This can provide additional stability to the operation.

5.4.5 SUSHAPE - Wiener shaping filter

The demos in demos/Deconvolution also contain a demonstration of the Wiener shaping filter **sushape**.

2D Filtering Operations

Filtering in the (k1,k2) domain, and the (F,K) domain is often useful for changing dip information in data. The programs

- SUKFILTER - radially symmetric K-domain, sin$\hat{2}$-tapered, polygonal filter
- SUK1K2FILTER - symmetric box-like K-domain filter defined by the cartesian product of two sin$\hat{2}$-tapered polygonal filters defined in k1 and k2
- SUKFRAC - apply FRACtional powers of i—k— to data, with phase shift
- SUDIPFILT - DIP–or better–SLOPE Filter in f-k domain

provide the beginnings of a set of K-domain and F-K-domain filtering operations.

5.4.6 SURESAMP - Resample Data in Time

Often data need to be resampled to either reduce or increase the number of samples for processing or data storage. For seismic data, the smart way of doing this is by the method of sinc interpolation. The program “suresamp”

- SURESAMP - Resample in time

performs this operation.

See the demos in $CWPROOT/src/demos/Filtering for further information about filtering in SU.

Chapter 6

Seismic Modeling Utilities

An important aspect of seismic exploration and research are programs for creating synthetic data. Such programs find their use, both in the practical problem of modeling real data, as well as in the testing of new processing programs. A processing program that will not work on idealized model data will likely not work on real seismic data.

Another important aspect of modeling programs is the fact that many seismic processing algorithms (such as migration) may be viewed as *inverse processes*. The first step in such an inverse problem may be to create a method to solve the forward problem, and then formulate the solution to the inverse problem as a "backpropagation" of the recorded data to its position in the subsurface.

There are two parts to the seismic modeling task. The first part is the construction of background wavespeed profiles, which may consist of uniformly sampled arrays of floating point numbers. The second part is the construction of the synthetic wave information which propagates in that wavespeed profile.

Because of the intimate relationship between seismic modeling and seismic processing, background wavespeed profiles created for modelling tasks, may also be useful for processing tasks.

Of course, if some simple assumptions are made, it may be possible for background wavespeed information to be built into the modeling program.

6.1 Background Wavespeed Profiles

There are many approaches to creating background wavespeed profiles. For many processes, it may be that a simple array of floating point numbers, each representing the wavespeed, slowness (1/wavespeed), or sloth (slowness squared) on a uniformly sampled grid will be sufficient.

However, more advanced techniques may involve wavespeed profile generation in triangulated or tetrahedrized media.

6.2 Uniformly Sampled Models

In Seismic Unix there are several programs which may be used to generate background wavespeed profile data. Often, such data need to be smoothed.

These programs are:

- UNISAM - UNIformly SAMple a function y(x) specified as x,y pair
- UNISAM2 - UNIformly SAMple a 2-D function f(x1,x2)
- MAKEVEL - MAKE a VELocity function v(x,y,z)
- UNIF2 - generate a 2-D UNIFormly sampled velocity profile from a layered model. In each layer, velocity is a linear function of position.

- SMOOTHINT2 - SMOOTH non-uniformly sampled INTerfaces, via the damped least-squares technique
- SMOOTH2 - SMOOTH a uniformly sampled 2d array of data, within a user- defined window, via a damped least squares technique
- SMOOTH3D - 3D grid velocity SMOOTHing by the damped least squares

Please see the selfdoc of each of these programs for further information. Also see the demos in $CWPROOT/src/Velocity_Profiles

6.3 Synthetic Data Generators

There are a number of programs for generating synthetic seismic and seismic-like data in the SU package. These are

- SUPLANE - create common offset data file with up to 3 planes
- SUSPIKE - make a small spike data set
- SUIMP2D - generate shot records for a line scatterer embedded in three dimensions using the Born integral equation
- SUIMP3D - generate inplane shot records for a point scatterer embedded in three dimensions using the Born integral equation
- SUFDMOD2 - Finite-Difference MODeling (2nd order) for acoustic wave equation
- SUSYNCZ - SYNthetic seismograms for piecewise constant V(Z) function True amplitude (primaries only) modeling for 2.5D
- SUSYNLV - SYNthetic seismograms for Linear Velocity function
- SUSYNVXZ - SYNthetic seismograms of common offset V(X,Z) media via Kirchhoff-style modeling
- SUSYNLVCW - SYNthetic seismograms for Linear Velocity function for mode Converted Waves
- SUSYNVXZCS - SYNthetic seismograms of common shot in V(X,Z) media via Kirchhoff-style modeling

Of these, only sufdmod2, susynvxz, and sysnvxzcs require an file of input wavespeed. The other programs use commandline arguments for wavespeed model input.

Please see the demos in $CWPROOT/src/demos/Synthetic for further information. Also, a number of the other demos use these programs for synthetic data generation.

6.4 Delaunay Triangulation

More sophisticated methods of synthetic data generation use assumptions regarding the nature of the medium (as represented by the input wavespeed profile data format) to expedite computations of the synthetic data. One such method is triangulation via the Delaunay method.

6.4.1 Triangulated Model Building

There are two way of making triangulated models. The first is to explicitly input boundary coordinates and wavespeed (actually sloth values) with "trimodel." The second is to make a uniformly sampled model, perhaps with one of the model building utilities above, and then use "uni2tri" to convert the uniformly sampled model to a triangulated model. (Please note, that this will work better if the wavespeed model is smoothed, prior to conversion.) These programs are

- TRIMODEL - make a triangulated sloth (1/velocity squared) model
- UNI2TRI - convert UNIformly sampled model to a TRIangulated model
- TRI2UNI - convert a TRIangulated model to UNIformly sampled model

6.4.2 Synthetic Seismic Data in Triangulated Media

Several programs make use of triangulated models to create ray tracing, or ray-trace based synthetic seismograms. There is also a code to create Gaussian beam synthetic seismograms in a triangulated medium. These programs are:

- NORMRAY - dynamic ray tracing for normal incidence rays in a sloth model
- TRIRAY - dynamic RAY tracing for a TRIangulated sloth model
- GBBEAM - Gaussian beam synthetic seismograms for a sloth model
- TRISEIS - Synthetic seismograms for a sloth model

There is a comprehensive set of demos located in the directory $CWPROOT/src/Delaunay_Triangulation

6.5 Tetrahedral Methods

Some new functionality will be entering SU in future releases for tetrahedral model building and ray tracing in tetrahedral models. One code that is in the current release is

- TETRAMOD - TETRAhedron MODel builder. In each layer, velocity gradient is constant or a 2-D grid; horizons could be a uniform grid and/or added by a 2-D grid specified.

Chapter 7

Seismic Processing Utilities

There are a collection of operations which are uniquely seismic in nature, representing operations which are designed to perform some aspect of the involved process which takes seismic data and converts it into images of the earth.

- stacking data,
- picking data,
- velocity analysis,
- normal moveout correction,
- dip moveout correction,
- seismic migration and related operations.

7.1 SUSTACK, SURECIP, SUDIVSTACK - Stacking Data

- SUSTACK - stack adjacent traces having the same key header word,
- SURECIP - sum opposing offsets in prepared data,
- SUDIVSTACK - Diversity Stacking using either average power or peak power within windows

7.2 SUVELAN, SUNMO - Velocity Analysis and Normal Moveout Correction

- SUVELAN - compute stacking velocity semblance for cdp gathers
- SUNMO - NMO for an arbitrary velocity function of time and CDP

Please see the demos in $CWPROOT/src/demos/Velocity_Analysis and $CWPROOT/src/demos/NMO for further information.

7.3 SUDMOFK, SUDMOTX, SUDMOVZ - Dip Moveout Correction

Dip-moveout is a data transformation which converts data recorded with offset to zero offset data. The following programs

- SUDMOFK - DMO via F-K domain (log-stretch) method for common-offset gathers
- SUDMOTX - DMO via T-X domain (Kirchhoff) method for common-offset gathers
- SUDMOVZ - DMO for V(Z) media for common-offset gathers

perform this operation.

7.4 Seismic Migration

The subject of seismic migration is one of the most varied in seismic data processing. Many algorthms have been developed to perform this task. Methods include Kirchhoff, Stolt, Finite-Difference, Fourier Finite-Difference, and several types of Phase-Shift or Gazdag Migration.

Please see the demos in $CWPROOT/src/demos/Migration for further information.

7.4.1 SUGAZMIG, SUMIGPS, SUMIGPSPI, SUMIGSPLIT - Phase Shift Migration

- SUGAZMIG - SU version of Jeno GAZDAG's phase-shift migration for zero-offset data.
- SUMIGPS - MIGration by Phase Shift with turning rays
- SUMIGPSPI - Gazdag's phase-shift plus interpolation migration for zero-offset data, which can handle the lateral velocity variation.
- SUMIGSPLIT - Split-step depth migration for zero-offset data.

7.4.2 SUKDMIG2D, SUMIGTOPO2D, SUDATUMK2DR, SUDATUMK2DS - 2D Kirchhoff Migration, and Datuming

- SUKDMIG2D - Kirchhoff Depth Migration of 2D poststack/prestack data
- SUDATUMK2DR - Kirchhoff datuming of receivers for 2D prestack data (shot gathers are the input)
- SUDATUMK2DS - Kirchhoff datuming of sources for 2D prestack data (input data are receiver gathers)
- SUMIGTOPO2D - Kirchhoff Depth Migration of 2D postack/prestack data from the (variable topography) recording surface

7.4.3 SUMIGFD, SUMIGFFD - Finite-Difference Migration

- SUMIGFD - 45 and 60 degree Finite difference migration for zero-offset data.
- SUMIGFFD - Fourier finite difference migration for zero-offset data. This method is a hybrid migration which combines the advantages of phase shift and finite difference migrations.

7.4.4 SUMIGTK - Time-Wavenumber Domain Migration

This algorthm was created by Dave Hale "on the fly" and, as far as we know, exists nowhere in else geophysical literature.

- SUMIGTK - MIGration via T-K domain method for common-midpoint stacked data

7.4.5 SUSTOLT - Stolt Migration

This is the classic F-K migration method of Clayton Stolt.

- SUSTOLT - Stolt migration for stacked data or common-offset gathers

Chapter 8

Processing Flows with SU

8.1 SU and UNIX

You need not learn a special seismic language to use SU. If you know how to use UNIX shell-redirecting and pipes, you are ready to start using SU—the seismic commands and options can be used just as you would use the built-in UNIX commands. In particular, you can write ordinary UNIX shell scripts to combine frequent command combinations into meta-commands (i.e., processing flows). These scripts can be thought of as "job files."

Table 8.1: UNIX Symbols

process1 < file1	process1 takes input from file1
process2 > file2	process2 writes on (new) file2
process3 >> file3	process3 appends to file3
process4 | process5	output of process4 is input to process5
process6 << text	take input from following lines

So let's begin with a capsule review of the basic UNIX operators as summarized in Table 8.1. The symbols <, >, and >> are known as "redirection operators," since they redirect input and output into or out of the command (i.e., process). The symbol | is called a "pipe," since we can picture data flowing from one process to another through the "pipe." Here is a simple SU "pipeline" with input "indata" and output "outdata":

```
sufilter f=4,8,42,54 <indata |
sugain tpow=2.0 >outdata
```

This example shows a band-limiting operation being "piped" into a gaining operation. The input data set `indata` is directed into the program **sufilter** with the `<` operator, and similarly, the output data set `outdata` receives the data because of the `>` operator. The output of **sufilter** is connected to the input of **sugain** by use of the `|` operator.

The strings with the `=` signs illustrate how parameters are passed to SU programs. The program **sugain** receives the assigned value 2.0 to its parameter `tpow`, while the program **sufilter** receives the assigned four component *vector* to its parameter `f`. To find out what the valid parameters are for a given program, we use the self-doc facility.

By the way, space around the UNIX redirection and pipe symbols is optional—the example shows one popular style. On the other hand, spaces around the `=` operator are *not* permitted.

The first four symbols in Table 8.1 are the basic grammar of UNIX; the final << entry is the symbol for the less commonly used "here document" redirection. Despite its rarity in interactive use, SU shell programs are significantly enhanced by appropriate use of the << operator—we will illustrate this below.

Many built-in UNIX commands do not have a self-documentation facility like SU's—instead, most do have "man" pages. For example,

```
% man cat

CAT(1)              UNIX Programmer's Manual              CAT(1)



NAME
     cat - catenate and print

SYNOPSIS
     cat [ -u ] [ -n ] [ -s ] [ -v ] file ...

DESCRIPTION
     Cat reads each file in sequence and displays it on the stan-
     dard output.  Thus

                    cat file

     displays the file on the standard output, and

                    cat file1 file2 >file3
--More--
```

You need to know a bit more UNIX lore to use SU efficiently—we'll introduce these tricks of the trade in the context of the examples discussed later in this chapter.

8.2 Understanding and using SU shell programs

The essence of good SU usage is constructing (or cloning!) UNIX shell programs to create and record processing flows. In this section, we give some annotated examples to get you started.

8.2.1 A simple SU processing flow example

Most SU programs read from standard input and write to standard output. Therefore, one can build complex processing flows by simply connecting SU programs with UNIX pipes. Most flows will end with one of the SU plotting programs. Because typical processing flows are lengthy and involve many parameter settings, it is convenient to put the SU commands in a shell file.

Remark: All the UNIX shells, Bourne (sh), Cshell (csh), Korn (ksh), ..., include a programming language. In this document, we exclusively use the Bourne shell programming language.

Our first example is a simple shell program called **Plot**. The numbers in square brackets at the end of the lines in the following listing are not part of the shell program—we added them as keys to the discussion that follows the listing.

```
#! /bin/sh                                          [1]
# Plot:   Plot a range of cmp gathers
# Author: Jane Doe
# Usage:  Plot cdpmin cdpmax

data=$HOME/data/cmgs                                [2]
```

```
# Plot the cmp gather.
suwind <$data key=cdp min=$1 max=$2 |                   [3]
sugain tpow=2 gpow=.5 |
suximage f2=0 d2=1 \                                    [4]
        label1="Time (sec)" label2="Trace number" \
        title="CMP Gathers $1 to $2" \
        perc=99 grid1=solid &                           [5]
```

Discussion of numbered lines:

1. The symbol `#` is the comment symbol—anything on the remainder of the line is not executed by the UNIX shell. The combination `#!` is an exception to this rule: the shell uses the file name following this symbol as a path to the program that is to execute the remainder of the shell program.

2. The author apparently intends that the shell be edited if it is necessary to change the data set—she made this easier to do by introducing the shell variable `data` and assigning to it the full pathname of the data file. The assigned value of this parameter is accessed as `$data` within the shell program. The parameter `$HOME` appearing as the first component of the file path name is a UNIX maintained environment variable containing the path of the user's home directory. In general, there is no need for the data to be located in the user's home directory, but the user would need "read permission" on the data file for the shell program to succeed.

   WARNING! Spaces are significant to the UNIX shell—it uses them to parse command lines. So despite all we've learned about making code easy to read, do *not* put spaces next to the `=` symbol. (Somewhere around 1977, one author's (Jack) first attempt to learn UNIX was derailed for several weeks by making this mistake.)

3. The main pipeline of this shell code selects a certain set of cmp gathers with **suwind**, gains this subset with **sugain** and pipes the result into the plotting program **suximage**. As indicated in the Usage comment, the cmp range is specified by command line arguments. Within the shell program, these arguments are referenced as `$1`, `$2` (i.e., first argument, second argument).

4. The lines within the **suximage** command are continued by the backslash escape character.

   WARNING! The line continuation backslash must be the *final* character on the line—an invisible space or tab following the backslash is one of the most common and frustrating bugs in UNIX shell programming.

5. The final `&` in the shell program puts the plot window into "background" so we can continue working in our main window. This is the X-Windows usage—the `&` should *not* be used with the analogous PostScript plotting programs (e.g., supsimage). For example, with **supsimage** in place of **suximage**, the `&` might be replaced by `| lpr`.

   The SU plotting programs are special—their self-doc doesn't show all the parameters accepted. For example, most of the parameters accepted by **suximage** are actually specified in the self-documentation for the generic CWP plotting program **ximage**. This apparent flaw in the self-documentation is actually a side effect of a key SU design decision. The SU graphics programs call on the generic plotting programs to do the actual plotting. The alternative design was to have tuned graphics programs for various seismic applications. Our design choice keeps things simple, but it implies a basic limitation in SU's graphical capabilities.

   The plotting programs are the vehicle for presenting your results. Therefore you should take the time to carefully look through the self-documentation for *both* the "SU jacket" programs (**suximage**, **suxwigb**, . . .) and the generic plotting programs (**ximage**, **xwigb**, . . .).

8.2.2 Executing shell programs

The simplest way to execute a UNIX shell program is to give it "execute permission." For example, to make our above **Plot** shell program executable:

```
chmod +x Plot
```

Then to execute the shell program:

```
Plot 601 610
```

Here we assume that the parameters `cdpmin=601`, `cdpmax=610` are appropriate values for the `cmgs` data set. Figure 8.1 shows an output generated by the `Plot` shell program.

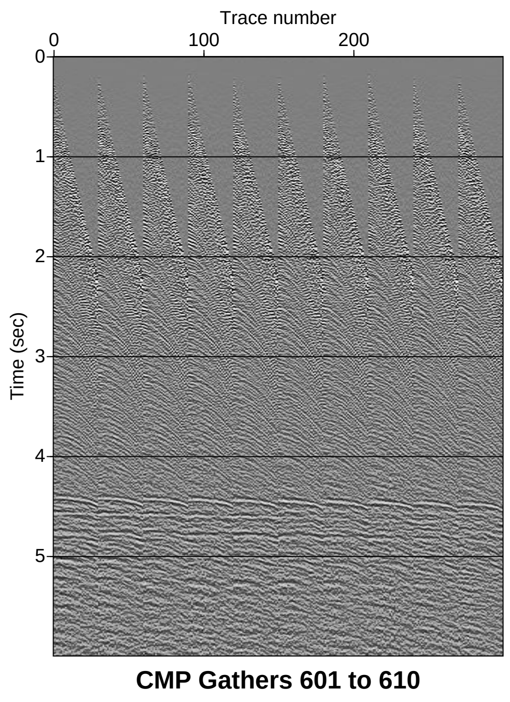

Figure 8.1: Output of the `Plot` shell program.

8.2.3 A typical SU processing flow

Suppose you want to use **sudmofk**. You've read the self-doc, but a detailed example is always welcome isn't it? The place to look is the directory **su/examples**. In this case, we are lucky and find the shell program, **Dmo**. Again, the numbers in square brackets at the end of the lines shown below are *not* part of the listing.

```
#! /bin/sh
# dmo
set -x                                                      [1]

# set parameters
input=cdp201to800                                           [2]
```

```
temp=dmocogs
output=dmocmgs
smute=1.7
vnmo=1500,1550,1700,2000,2300,2600,3000                          [3]
tnmo=0.00,0.40,1.00,2.00,3.00,4.00,6.00


# sort to common-offset, nmo, dmo, inverse-nmo, sort back to cmp
susort <$input offset cdp |                                      [4]
sunmo smute=$smute vnmo=$vnmo tnmo=$tnmo |                       [5]
sudmofk cdpmin=201 cdpmax=800 dxcdp=13.335 noffmix=4 verbose=1 | [6]
sunmo invert=1 smute=$smute vnmo=$vnmo tnmo=$tnmo >$temp         [7]
susort <$temp cdp offset >$output                                [8]
```

Discussion of numbered lines:

The core of the shell program (lines 5-7) is recognized as the typical dmo process: crude nmo, dmo, and then "inverse" nmo. The dmo processing is surrounded by sorting operations (lines 4 and 8). Here is a detailed discussion of the shell program keyed to the numbers appended to the listing (see also the discussion above for the `Plot` shell):

1. Set a debugging mode that asks UNIX to echo the lines that are executed. You can comment this line off when its output is no longer of interest. An alternate debugging flag is `set -v` which echos lines as they are read by the shell interpreter. You can use both modes at once if you like.
2. This line and the next two lines set filenames that, in this case, are in the same directory as the shell program itself. Again, the reason for using parameters here is to make it easy to "clone" the shell for use with other data sets. Those of us who work with only a few data sets at any given time, find it convenient to devote a directory to a given data set and keep the shells used to process the data in that directory as documentation of the processing parameters used. (SU does not have a built-in "history" mechanism.)
3. The dmo process requires a set of velocity-time picks for the subsidiary nmo processes. Because these picks must be consistent between the nmo and the inverse nmo, it is a good idea to make them parameters to avoid editing mistakes. Again, note the format of SU parameter vectors: comma-separated strings with no spaces. The nmo program (**sunmo**) will give an error message and abort if the `vnmo` and `tnmo` vectors have different lengths.
4. Note that **susort** allows the use of *secondary* sort keys. Do not assume that a secondary field that is initially in the "right" order will remain in that order after the sort—if you care about the order of some secondary field, specify it (as this shell program does). In this line, we sort the data according to increasing offsets and then, within each offset, we sort according to increasing cdp number.
5. The forward nmo step.
6. The dmo step.
7. The inverse nmo step.
8. Sort back to cdp and have increasing offset within each cdp.

If you want to thoroughly understand this shell program, your next step is to study the self-docs of the programs involved:

```
% sunmo

SUNMO - NMO for an arbitrary velocity function of time and CDP
```

```
sunmo <stdin >stdout [optional parameters]

Optional Parameters:
vnmo=2000         NMO velocities corresponding to times in tnmo
tnmo=0            NMO times corresponding to velocities in vnmo

...
```

Related shell programs are **su/examples/Nmostack** and **su/examples/Mig**.

8.3 Extending SU by shell programming

Shell programming can be used to greatly extend the reach of SU without writing C code. See, for example, **CvStack**, **FilterTest**, **FirstBreak**, and **Velan** in **su/examples**.

It is a sad fact that the UNIX shell is not a high level programming language—consequently, effective shell coding often involves arcane tricks. In this section, we'll provide some useful templates for some of the common UNIX shell programming idioms.

We use **CvStack** as an illustration. The core of this shell is a double loop over velocities and cdps that produces *velocity panels*—a concept not contained in any single SU program.

Remark: For most of us, writing a shell like **CvStack** from scratch is a time-consuming affair. To cut down the development time, your authors excerpt from existing shells to make new ones even when we don't quite remember what every detail means. We suggest that you do the same!

We won't comment on the lines already explained in our previous two shell code examples (see Sections 8.2.1 and 8.2.3), but instead focus on the new features used in **CvStack**.

```
#! /bin/sh
# Constant-velocity stack of a range of cmp gathers
# Authors: Jack, Ken
# NOTE: Comment lines preceding user input start with  #!#
set -x

#!# Set input/output file names and data parameters
input=cdp601to610
stackdata=cvstack
cdpmin=601 cdpmax=610
fold=30
space=1         # 1 null trace between panels

#!# Determine velocity sampling.
vmin=1500   vmax=3000   dv=150

### Determine ns and dt from data (for sunull)
nt=`sugethw ns <$input | sed 1q | sed 's/.*ns=//'`               [1]
dt=`sugethw dt <$input | sed 1q | sed 's/.*dt=//'`

### Convert dt to seconds from header value in microseconds
dt=`bc -l <<END                                                  [2]
        scale=4
        $dt / 1000000
END`
```

```
### Do the velocity analyses.
>$stackdata  # zero output file                                    [3]
v=$vmin
while [ $v -le $vmax ]                                             [4]
do
        cdp=$cdpmin
        while [ $cdp -le $cdpmax ]                                 [5]
        do
                suwind <$input \                                   [6]
                        key=cdp min=$cdp max=$cdp count=$fold |
                sunmo cdp=$cdp vnmo=$v tnmo=0.0 |
                sustack >>$stackdata
                cdp=`bc -l <<END                                   [7]
                        $cdp + 1
END`
        done
        sunull ntr=$space nt=$nt dt=$dt >>$stackdata               [8]
        v=`bc -l <<END
                $v + $dv
END`
done


### Plot the common velocity stacked data
ncdp=`bc -l <<END
        $cdpmax-$cdpmin+1
END`
f2=$vmin
d2=`bc -l <<END
        $dv/($ncdp + $space)                                       [9]
END`

sugain <$stackdata tpow=2.0 |

suximage perc=99 f2=$f2 d2=$d2 \
        title="File: $input  Constant-Velocity Stack " \
        label1="Time (s)"  label2="Velocity (m/s)" &

exit                                                               [10]
```

Discussion of numbered lines:

1. This elaborate construction gets some information from the first trace header of the data set. The program **sugethw** lists the values of the specified keys in the successive traces. For example,

```
% suplane | sugethw tracl ns
 tracl=1           ns=64

 tracl=2           ns=64

 tracl=3           ns=64
```

```
tracl=4          ns=64

tracl=5          ns=64

tracl=6          ns=64

...
```

Although **sugethw** is eager to give the values for every trace in the data set, we only need it once. The solution is to use the UNIX stream editor (**sed**). In fact, we use it twice. By default, **sed** passes along its input to its output. Our first use is merely to tell **sed** to quit after it puts the first line in the pipe. The second pass through **sed** strips off the unwanted material before the integer. In detail, the second **sed** command reads: replace (or substitute) everything up to the characters `ns=` with nothing, i.e., delete those characters.

2. We are proud of this trick. The Bourne shell does not provide floating point arithmetic. Where this is needed, we use the UNIX built-in **bc** calculator program with the "here document" facility. Here, we make the commonly needed conversion of sampling interval which is given in microseconds in the SEG-Y header, but as seconds in SU codes. Note carefully the *back*quotes around the entire calculation—we assign the result of this calculation to the shell variable on the left of the equal sign, here `dt`. The calculation may take several lines. We first set the number of decimal places with `scale=4` and then do the conversion to seconds. The characters `END` that follow the here document redirection symbol `<<` are arbitrary, the shell takes its input from the text in the shell file until it comes to a line that contains the same characters again. For more information about **bc**:

   ```
   % man bc
   ```

3. As the comment indicates, this is a special use of the output redirection symbol that has the effect of destroying any pre-existing file of the same name or opening a new file with that name. In fact, this is what `>` always does as its first action—it's a dangerous operator! If you intend to *append*, then, as mentioned earlier, use `>>`.

4. This is the outer loop over velocities. Another warning about spaces—the spaces around the bracket symbols are essential.

   Caveat: The bracket notation is a nice alternative to the older clunky `test` notation:

   ```
   while test $v -le $vmax
   ```

   Because the bracket notation is not documented on the typical **sh** manual page, we have some qualms about using it. But, as far as we know, all modern **sh** commands support it—please let us know if you find one that doesn't.

   WARNING! OK, now you know that there is a UNIX command called `test`. So don't use the name "test" for one of your shell (or C) programs—depending on your `$PATH` setting, you could be faced with seemingly inexplicable output.

5. This is the inner loop over cdps.

6. Reminder: No spaces or tabs after the line continuation symbol!

7. Notice that we broke the nice indentation structure by putting the final `END` against the left margin. That's because the **sh** manual page says that the termination should contain only the `END` (or whatever you use). In fact, most versions support indentation. We didn't think the added beautification was worth the risk in a shell meant for export. Also note that we used **bc** for an integer arithmetic calculation even though integer arithmetic is built into the Bourne shell—why learn two arcane rituals, when one will do? See `man expr`, if you are curious.

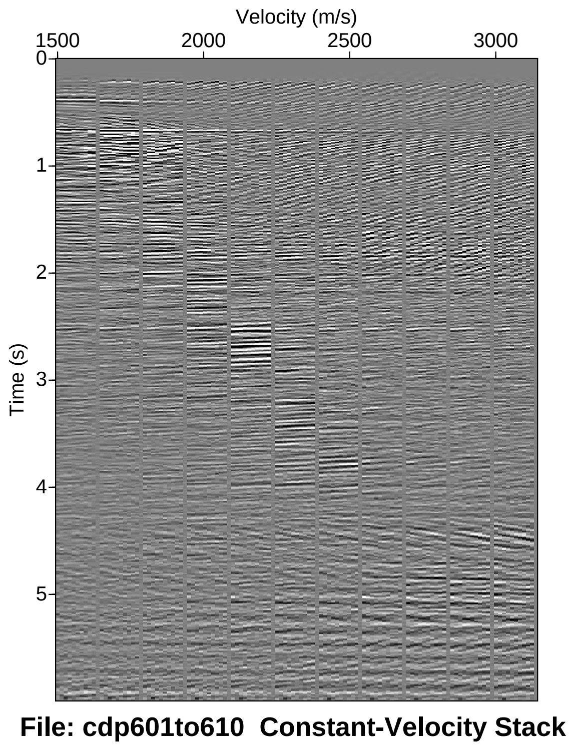

Figure 8.2: Output of the `CvStack` shell program.

8. **sunull** is a program I (Jack) wrote to create all-zero traces to enhance displays of the sort produced by `CvStack`. Actually, I had written this program many times, but this was the first time I did it on purpose. (Yes, that was an attempt at humor.)
9. An arcane calculation to get velocity labeling on the trace axis. Very impressive! I wonder what it means? (See last item.)
10. The `exit` statement is useful because you might want to save some "spare parts" for future use. If so, just put them after the `exit` statement and they won't be executed.

Figure 8.2 shows an output generated by `CvStack`.

Chapter 9

Answers to Frequently Asked Questions

This chapter addresses questions often asked by new SU users. Some answers refer to the directory `CWPROOT`. We use this symbolic name for the directory that contains the CWP/SU source code, include files, libraries, and executables. You are asked to specify this directory name during the SU installation procedure.

9.1 Installation questions

Complete information about the installation process is found in the README files supplied with the distribution. Here we discuss only some commonly found installation problems.

Question 1 *I get error messages about missing* `fgetpos` *and* `fsetpos` *routines, even though I am using the* GCC *compiler. How do I get around this problem?*

Answer 1 We've seen this problem most often with older SUN OS 4.xx (pre-SOLARIS). These SUN systems may not have the `fgetpos` and `fsetpos` subroutines defined. Because these two routines are not currently used in the SU package, we have modified the installation process to permit the user to define a compile-time flag to circumvent this problem. Please uncomment the OPTC line in the paragraph in Makefile.config that looks like this:

```
# For SUN installing with GCC compiler but without GLIBC libraries
#OPTC = -O -DSUN_A -DSUN
```

and do a "make remake".

Question 2 *I get error messages regarding missing* `strtoul`*, and/or* `strerror` *routines, even though I am using the* GCC *compiler. How do I get around this problem?*

Answer 2 Again, this is most often seen with the older SUN OS. The fix is the same as for the previous question.

Question 3 *Why do I get missing subroutine messages about* ANSI C *routines? Isn't the* GCC *compiler supposed to be an* ANSI *compiler?*

Answer 3 The GCC compiler is just that, a compiler. It draws on the libraries that are present on the machine. If the GNU libraries (this is the "glibc" package) have not been installed, then the GCC compiler will use the libraries that are native to the machine you are running on. Because the four routines listed above are not available in the SUN 4. OS, GCC does not recognize them. However, installing the GNU libraries will make the GCC compiler behave as a full ANSI C compiler.

Question 4 *Why do I get missing subroutine messages about* ANSI C *routines? I can't get the code to compile because my compiler can't find "bzero" or "bcopy", how can I fix this?*

Answer 4 You really shouldn't be having this problem, because we try to keep to the ANSI standard, but sometimes old style function calls creep in. The problem of rooting these things out is exacerbated because many systems still support the old style calls.

If you have trouble installing because your compiler can't find "bcopy" or "bzero" make the following replacements.

Replace all statements of the form

```
bzero( a, b);
```

with statements of the form:

```
memset( (void *) a , (int) '\0', b );
```

Please replace all instances of statements of the form of:

```
bcopy ( a , b, c);
```

with a statements of the form:

```
memcpy( (void *) b, (const void *) a, c );
```

9.2 Data format questions

In this section, we address questions about converting data that are in various formats into SU format.

Question 5 *What is the data format that* SU *programs expect?*

Answer 5 The SU data format is based on, (but is not exactly the same as) the SEG-Y format. The SU format consists of data traces each of which has a header. The SU trace header is identical to SEG-Y trace header. Both the header and the trace data are written in the native binary format of your machine. You will need to use **segyread** to convert SEGY data to SU data.
Caution: The optional fields in the SEG-Y trace header are used for different purposes at different sites. SU itself makes use of certain of these fields. Thus, you may need to use `segyclean`—see the answer to Question 7. SU format does not have the binary and ebcdic tape headers that are part of the SEG-Y format.

After installing the package, you can get more information on the SEG-Y/SU header by typing:

```
% sukeyword -o
```

This lists the include file `segy.h` that defines the SU trace header.

Question 6 *Is there any easy way of adding necessary* SEG-Y *information to our own modeled data to prepare our data for processing using the* SU *package?*

Answer 6 It depends on the details of how your data was written to the file:

1. If you have a 'datafile' that is in the form of binary floating point numbers of the type that would be created by a C program, then use `suaddhead` to put SU (SEG-Y) trace headers on the data. Example:

   ```
   % suaddhead < datafile  ns=N_SAMP > data.su
   ```

Here, `N_SAMP` is the (integer) number of samples per trace in the data.

2. If your data are Fortran-style floats, then you would use:

   ```
   % suaddhead < datafile ftn=1 ns=NS > data.su
   ```

   See also, Question 10.

3. If your data are ASCII, then use:

   ```
   % a2b n1=N1 < data.ascii | suaddhead ns=NS > data.su
   ```

   Here `N1` is the number of floats per line in the file `data.ascii`.

4. If you have some other data type, then you may use:

   ```
   % recast < data.other in=IN out=float | suaddhead ns=NS > data.su
   ```

   where `IN` is the type (int, double, char, etc...)

For further information, consult the self-docs of the programs `suaddhead`, `a2b`, and `recast`.

Question 7 *I used* `segyread` *to read a* SEG-Y *tape. Everything seems to work fine, but when I plot my data with suximage, the window is black. What did I do wrong?*

Answer 7 When you read an SEG-Y tape, you need to pipe the data through `segyclean` to zero the optional SEG-Y trace header field. If the SU programs see nonzero values in certain parts of the optional field, they try to display the data as "nonseismic data," using those values to set the plot parameters.

Another possibility is that there are a few data values that are so large that they are overwhelming the 256 gray scale levels in the graphics. The way to get around this problem is to set **perc=99** in the graphics program. For example:

```
% suximage < sudata  perc=99 &
```

This will clip data values with size in the top 1 percentile of the total data.

Question 8 *I am trying to plot data with the* `pswigb` *(or* `pswigp`*, or* `xwigb`*, or ...) program. I know that I have data with* `n1=NSAMP` *and* `n2=NTRACES`*, but when I plot, I find that I have to set* `n1=NSAMP+60` *for the plot to look even remotely correct. Why is this?*

Answer 8 It is likely that you are trying to plot with the wrong tool. The input data format of the programs, `pswigb`, `pswigp`, `pscontour`, `pscube`, `psmovie`, `xwigb`, `xgraph`, and `xmovie`, expect data to consist of simple floating point numbers. If your data are SU data (SEG-Y) traces, then there is an additional header at the beginning of each trace, which, on most computer architectures, is the same number (240) of bytes as the storage for 60 floats.

To plot these data, use respectively: `supswigb`, `supswigp`, `supscontour`, `supscube`, `supsmovie`, `suxwigb`, `suxgraph`, or `suxmovie`.

Also, it is not necessary to specify the dimensions of the data for these latter programs. The `su`-versions of the codes determine the necessary information from the appropriate header values. (In fact, that is *all* they do—the actual graphics is handled by the version without the `su` prefix.)

Question 9 *I want to check the size of a file to see if it has the right number of values, but I am not sure how to take the header into account. How is this done?*

Answer 9 If the file consists of simple floating point numbers, then the size in bytes equals the size of a float times the number of samples (`SIZE = 4 * N_SAMP`). The SU data (SEG-Y traces) also have a header (240 bytes per trace) giving the total number of bytes as:
`(240 + 4 N_SAMP ) N_TRACES`.
The byte count computed in this way is the number that the UNIX command `ls -l` shows.

Caveats: The above calculations assume that you have the conventional architecture and that the header definition in `segy.h` has not been altered. Watch out as machines with 64 bit word size become common!

Question 10 *I have some data in Fortran form and tried to convert it to* SU *data via the following:*

```
% suaddhead < data.fortran ns=N_SAMP ftn=1 > data.su
```

but this did not work properly. I am sure that my fortran data are in unformatted binary floats. What should I do?

Answer 10 There are different ways of interpreting the term "unformatted" with regard to fortran data. Try:

```
% ftnstrip < data.fortran | suaddhead ns=N_SAMP > data.su
```

The program `ftnstrip` can often succeed in converting your fortran data into C-like binary data, even when the `ftn=1` option in `suaddhead` fails. (Note: the program **ftnunstrip** may be used to take C-style binary data and convert it to Fortran form.)

Question 11 *I just successfully installed the* CWP/SU *package, but when I try to run the demo scripts, I get many error messages describing programs that the shell script cannot find. How do I fix this?*

Answer 11 You need to put `CWPROOT/bin` (where `CWPROOT` is `/your/root/path` that contains the CWP/SU source code, include files, libraries, and executables) in your shell `PATH`. This is done in your `.cshrc` file if you run under `csh` or `tcsh`. In Bourne shell (`sh`), Born Again shell (`bash`), or Korn shell (`ksh`) the `PATH` variable is in your `.profile` file. You also need to type

```
% rehash
```

if you are running C-shell `/bin/csh` or TC-shell `/bin/tcsh` as your working shell environment, if you have not relogged since you compiled the codes.

Question 12 *How do I transfer data between* SU *and a commercial package, such as Promax.*

Answer 12 The short answer is that you make a SEGY tape on disk file. To do convert a file called, say, "data.su" to a segy file do the following:

```
% segyhdrs < data.su
% segywrite tape=data.segy < data.su
```

Now use Promax to read data.segy. This file is a "Promax tape-on-disk file in IBM Real format." Choose Promax menus accordingly.
For other commercial packages, use the appropriate commands to read a SEGY tape on disk file.
To go from the commercial package to SU follow the reverse steps. Create a file that is a SEGY tape image and then use

```
% segyread tape=data.segy | segyclean > data.su
```

Question 13 *I would like to strip the trace headers off of some SU data, perform an operation of some type on the bare traces and put the headers back on without losing any of the header information. How do I do this?*

Answer 13 Do the following:

```
% sustrip < data.su head=headers > data.binary
```

(Do whatever was desired to data.binary to make data1.binary)

```
% supaste < data1.binary head=headers > data1.su
```

Question 14 *I have made some data on an IBM RS6000 and have transferred it to my Linux-based PC system. The data looks ok on the RS6000, but when I try to work with it on the PC, none of the SU programs seem to work. What is wrong?*

Answer 14 The problem you have encountered is that there are two IEEE binary formats called respectively 'big endian' and 'little endian' or, alternately 'high byte' and 'low byte'. These terms refer to the order of the bytes that represent the data. IBM RS6000, Silicon Graphics, NeXT (black hardware), SUN, HP, PowerPC, any Motorola chip-based platforms are 'big endian' machines, whereas, Intel-based PCs and Dec and Dec Alpha products are 'little endian' platforms.

Two programs are supplied in the CWP/SU package for swapping the bytes for data transfer. These are **swapbytes** and **suswapbytes**.

The program **swapbytes** is designed to permit the user to swap the bytes on binary data that are all one type of data (floats, doubles, shorts, unsigned shorts, longs, unsigned longs, and ints).

For data that are in the SU format, the program **suswapbytes** is provided.

Furthermore, within the programs **segyread** and **segywrite** there are "swap=" flags that permit the user to specify whether the platform they are working on are "big endian" or "little endian" platforms.

In older releases of SU there were problems with the bitwise operations that would be encountered in the wiggle-trace drawing routines. However, these problems have been fixed via the ENDIANFLAG that appears in Makefile.config.

Question 15 *How do I convert data that are in the SEG-2 format to SEGY?*

Answer 15 In $CWPROOT/src/Third_Party/seg2segy there are two programs that have been made available to us by the University of Pau in France, for this purpose. These should be easy to install on any system where SU has been installed.

Once you have converted data.seg2 to data.segy, you may read it into the SU format via:

```
% segyread tape=data.segy > data.su
```

9.3 Tape reading and writing

This section contains frequently asked questions about reading and writing SEG-Y tapes with SU.
Tape reading/writing is more of an art than a science. Here are a few tips.

1. Make sure your tape drive is set to be variable block length. If you are on an IBM RS6000, this means you will need to use `smit` to set `blocksize=0` on your tape device. Having the tape drive set to some default constant blocksize (say blocksize=1024 or 512) will foil all attempts to read an SEG-Y tape.

2. To read multiple tape files on a tape, use the non rewinding device. On an `RS6000` this would be something like `/dev/rmtx.1`, see `man mt` for details.
3. If this still doesn't work, then try:

   ```
   % dd if=/dev/rmtx of=temps bs=32767 conv=noerror
   ```

   Here, `/dev/rmtx` (not the real name of the device, it varies from system to system) is your regular (rewinding) tape device. In the option, `bs=32767`, we gave the right blocksize ($2^{16}+1$) for an `IBM/RS6000`. Try `bs=32765` ($2^{16}-1$) on a SUN. This will dump the entire contents of the tape onto a single file.

Question 16 *How do I write multiple SEG-Y files onto a tape?*

Answer 16 Here is a shell script for writing multiple files on a tape:

```
#! /bin/sh

DEV=/dev/nrxt0  # non rewinding tape device

mt -f $DEV rewind

j=0
jmax=40

while test "$j" -ne "$jmax"
do
        j=`expr $j + 1`
        echo "writing tape file  $j"
        segywrite tape=$DEV bfile=b.$j hfile=h.$j verbose=1 buff=0 < ozdata.$j
done

exit 0
```

9.4 Geometry Setting

Question 17 *How do I do "geometry setting" in SU?*

Answer 17 There is a common seismic data manipulation task that often is called "geometry setting" in commercial packages in which the user converts information in the survey observers' logs into values in the trace headers.
The CWP/SU package does indeed, have provisions for getting and setting header fields, as well as computing a third header field from one or two other header fields. The programs that you need to use for this are:

sugethw ("SU get header word")
sushw ("SU set header word")
suchw ("SU change or compute header word")

Type the name of each program to see the self documentation of that code.

In addition, to find out what the header field "keywords" mentioned in these programs are: type: sukeyword -o

You may have the information in a variety of forms. The most common and least complicated assumptions of that form will be made here.

The task requires the following basic steps.

1. Get your data into SU format. The SU format is not exactly SEGY, but it does preserve the SEGY header information. If you are starting with SEGY data (either on tape, or on in the form of a diskfile) then you use "segyread" to read the data into an su file format.

   For tape:

   ```
   % segyread tape=/dev/rmt0 bfile=data.1 header=h.1 | segyclean > data.su
   ```

   For diskfile

   ```
   %  segyread tape=data.segy bfile=data.1 header=h.1 | segyclean > data.su
   ```

   The file data.segy is assumed here to be a "tape image" of segy data. You have to be careful because some commercial software will write SEGY-like data, by mimicking the layout of the SEGY format, but this format will not be in the true IBM tape format that SEGY is defined to be. In Promax, if you write a SEGY file in IBM Real format, then this will be true SEGY tape image. working on.

2. If you have your data in the SU format, then you may view the ranges of the SEGY headers (headers that are not set will not be shown) via:

   ```
   % surange < data.su
   ```

3. Data often comes with some fields already set. To dump these fields in a format that is convenient for geometry setting, you would use sugethw in the following way:

   ```
   % sugethw < data.su  output=geom  key=key1,key2,... > hfile.ascii
   ```

   The strings "key1,key2,..." are the keywords representing the desired SEGY trace header fields. These keywords may be listed via:

   ```
   % sukeyword -o
   ```

4. Once you have dumped the desired header fields into hfile.ascii then you may edit them with the editor of your choice. The point is that you may create a multi-column ascii file that lists the values of specific header fields (trace by trace, as they appear in data.su) by *any* method you wish. Each column will contain the value of a specific header field to be set.

5. Now that you have created the ascii file containing your header values, you may load these values into data.su via:

   ```
   % a2b < hfile.ascii n1=N_columns > hfile.bin
   ```

   Here, N_columns is the number of columns in hfile.ascii. This is to convert hfile.ascii to a binary file.

   Now use:

   ```
   % sushw < data.su key=key1,key2,...  infile=hfile.bin > data1.su
   ```

   Here key1,key2,... are the appropriate keywords representing the fields being set, listed in the exact order the values appear, column by column in hfile.ascii.

6. If you want to compute a third header field from two given header field values, then you may use: **suchw** for this. Also, if the header fields that you want to set are systematic in some way (are constant for each trace or vary linearly across a gather), then you don't have to use the "infile=" option. You may simply give the necessary values to sushw. See the selfdocs for sushw and suchw for examples of these.

9.5 Technical Questions

Question 18 *I want to resample my data so that I have half as many traces, and half as many samples. How do I do that?*

Answer 18 To resample data, you must do the following:

1. Check that you won't have aliasing. Do this by viewing the amplitude spectra of your data. Do this with **suspecfx**

   ```
   suspecfx < data.su | suxwigb
   ```

2. If the bandwidth of your data extends beyond the new nyquist frequency of your data (which, in this example, will be half of the original nyquist frequency) then you will have to filter your data to fit within its new nyquist bandwidth. Do this with **sufilter**

   ```
    sufilter < data.su f=f1,f2,f3,f4  amps=0,1,1,0 > data.filtered.su
   ```

   Here, the **f1 f2 f3 f4** are the filter corner frequencies and **amps=0,1,1,0** indicate that the filter is a bandpass filter.

3. Now you may resample your data with suresamp:

   ```
   suresamp < data.filtered.su  nt=NTOUT dt=DTOUT > data.resampled.su
   ```

For your case, NTOUT is 1/2 of the original number of samples, and DTOUT is twice the time sampling interval (in seconds) of that in the original data. Your output data should look quite similar to your input data, with the exception that the bandwidth will change.

9.6 General

This section addresses general questions about the SU package.

Question 19 *What are these funny words gelev, selev, fldr, etc. that I see in various places?*

Answer 19 These are the "keywords" that are required for many of the codes. They refer to SU (Segy) header fields.

```
Type:   sukeyword -o                to see the whole list
Type:   sukeyword keyword           to see the listing for an individual
                                    keyword
```

Question 20 *What do the terms "little endian" and "big endian" and mean?*

Answer 20 There are two IEEE binary formats, called respectively 'little endian' and 'big endian'. These are also called 'high byte' and 'low byte', respectively. These refer to the byte order in the bitwise representation of binary data. The following platforms are 'little endian': DEC and Intel-based PC's. The other common platforms are "big endian": IBM RS6000, Silicon Graphics, NeXT (black hardware), SUN, HP, PowerPC, any Motorola chip-based platform.

Question 21 *Why are* CWP/SU *releases given by integers (22, 23, 24, etc...) instead of the more familiar decimal release numbers (1.1, 1.3, etc...)?*

Answer 21 The CWP/SU release numbers are chosen to correspond to the `SU NEWS` email messages. The individual codes in the package have traditional decimal release numbers (assigned by RCS), but these are all different. The package changes in incremental, but non-uniform ways, so the standard notation seems inappropriate. However, the user may view 24 to be 2.4. We may adopt this convention in the future.

Remark: In the early days, we *did* use RCS to simultaneously update all the codes to 2.1, 3.1, ... This practice died a natural death somewhere along the way.

Question 22 *How often are the codes updated?*

Answer 22 The CWP/SU package is updated at roughly 3-6 month intervals. We mail announcements of these releases to all known users. Since we do not provide support for outdated versions, we urge you to remain current.

Question 23 *I have a complicated collection of input parameters for a* CWP/SU *program. I want to run the command from the command line of a terminal window, but I don't want to retype the entire string of input parameters. What do I do?*

Answer 23 CWP/SU programs that take their input parameters from the command line also have the feature of being able to read from a "parameter file." This is invoked by setting the parameter `par=parfile`, where `parfile` is a file containing the desired commandline string.

For example:

```
suplane ntr=20 nt=40 dt=.001 | ...
```

is completely equivalent to the command:

```
suplane par=parfile | ...
```

if the string

```
ntr=20 nt=40 dt=.001
```

is contained in 'parfile.'

Question 24 *I can't find an* **sudoc** *entry for the function "ints8r," yet the SU manual says that all library functions have online documentation? What am I doing wrong?*

Answer 24 The proper search procedure for a library function (such as ints8r) is:

```
% sufind ints8r
```

Which yields:

```
INTSINC8 - Functions to interpolate uniformly-sampled data via 8-coeff. sinc
                approximations:

ints8c  interpolation of a uniformly-sampled complex function y(x) via an


For more information type: "sudoc program_name <CR>"
```

The name INTSINC8 is the name of the file that contains the library function ins8c. You may now use **sudoc** to find out more information via:

```
% sudoc intsinc8
```

Which yields:

```
In /usr/local/cwp/src/cwp/lib:
INTSINC8 - Functions to interpolate uniformly-sampled data via 8-coeff. sinc
                approximations:

ints8c  interpolation of a uniformly-sampled complex function y(x) via an
         8-coefficient sinc approximation.
ints8r  Interpolation of a uniformly-sampled real function y(x) via a
                table of 8-coefficient sinc approximations

Function Prototypes:
void ints8c (int nxin, float dxin, float fxin, complex yin[],
        complex yinl, complex yinr, int nxout, float xout[], complex yout[]);
void ints8r (int nxin, float dxin, float fxin, float yin[],
        float yinl, float yinr, int nxout, float xout[], float yout[]);

Input:
nxin            number of x values at which y(x) is input
dxin            x sampling interval for input y(x)
fxin            x value of first sample input
yin             array[nxin] of input y(x) values:  yin[0] = y(fxin), etc.
yinl            value used to extrapolate yin values to left of yin[0]
yinr            value used to extrapolate yin values to right of yin[nxin-1]
nxout           number of x values a which y(x) is output
xout            array[nxout] of x values at which y(x) is output

Output:
yout            array[nxout] of output y(x):  yout[0] = y(xout[0]), etc.

Notes:
Because extrapolation of the input function y(x) is defined by the
left and right values yinl and yinr, the xout values are not restricted
to lie within the range of sample locations defined by nxin, dxin, and
fxin.

The maximum error for frequiencies less than 0.6 nyquist is less than
one percent.

Author:  Dave Hale, Colorado School of Mines, 06/02/89
```

Question 25 *I have written my own SU programs and would like them to appear in the* **suname** *and* **sudoc** *listings. How do I do this?*

Answer 25 Run **updatedocall** (source code located in CWPROOT/par/shell). If you have put this code under a new path, then you must add this path to the list of paths in the updatedoc script. For the selfdoc information to be captured by the updatedoc script, you will need to have the following marker lines at the beginning and end of the selfdoc and additional information portion of the source code of your program.

```
/********************** self documentation *********************/
/**************** end self doc ********************************/
```

Be sure to clone these directly out of an existing SU program, rather than typing them yourself, so that the pattern is the exact one expected by the updatedoc script.

Question 26 *I have a gray scale (not color) PostScript file made with psimage and would like to convert it to a color PostScript format, but do not have the original binary data that I made the file from. How do I do this?*

Answer 26 You have to restore the binary file to make the new color PostScript file. Here is how you do it. (Here, we are assuming a bit-mapped graphic as would be produced by psimage or supsimage).

1. Make a backup of your PostScript file.
2. edit the PostScript file removing everything but the hexidecimal binary image that makes up the majority of the file. (Note, in the line preceeding the hexidecimal data portion of the file will be a pair of numbers that represents the dimensions of the data. You will need these numbers for later steps.)
3. use h2b to convert the hexidecimal file to binary
4. You will find that the file is flipped from the original input file. Use transp to flip the data. Note that the n1 and n2 values that are used by transp are the dimensions of the input data, which are the reverse of the output data. (The n1 value, is *not* the total number of samples, that is returned by h2b, instead total no. values = n1 $\times$ n2.)
5. You now have a 0-255 representation of your binary data which you should be able to plot again any way you desire.

This method may be used to convert scanned images to SU format, as well, with the next step in the procedure to be putting SU headers on the data with **suaddhead**.

Chapter 10

How to Write an SU Program

10.1 Setting up the Makefile

The CWP/SU package uses a sophisticated Makefile structure, that you may also use when you develop new code. You should begin any new code writing project by creating a local directory in your working area. You should then copy the Makefile from $CWPROOT/src/su/main into that directory and make the following changes

```
Change:

D = $L/libcwp.a $L/libpar.a $L/libsu.a

LFLAGS= $(PRELFLAGS) -L$L -lsu -lpar -lcwp -lm $(POSTLFLAGS)

to:

D = $L/libcwp.a $L/libpar.a $L/libsu.a

B = .

OPTC = -g

LFLAGS= $(PRELFLAGS) -L$L -lsu -lpar -lcwp -lm $(POSTLFLAGS)

Change:


PROGS =                 \
         $B/bhedtopar    \
         $B/dt1tosu      \
         $B/segyclean    \
         $B/segyhdrs     \
         $B/segyread     \
        ...
         ...

to:

PROGS =                 \
        $B/yourprogram
```

where the source code of your program is called “yourprogram.c” and resides in this directory.

You should then be able to simply type “make” and “yourprogram” will be compiled.

As a test you can try copying one of the existing SU programs, from $CWPROOT/src/su/main into your local working directory, modifying the Makefile accordingly and typing: make.

Indeed, because all new SU programs may be viewed as beginning as clones of existing SU programs of a similar structure, this is perhaps the best way to begin any new coding venture.

10.2 A template SU program

Although variations are usually needed, a template for a typical SU program looks like the program listing below (we excerpted lines from the program `sumute` to build this template). The numbers in square brackets at the end of the lines in the listing are not part of the listing—we added them to facilitate discussion of the template. The secret to efficient SU coding is finding an existing program similar to the one you want to write. If you have trouble locating the right code or codes to “clone,” ask us—this can be the toughest part of the job!

```
/* SUMUTE: $Revision: 1.20 $ ; $Date: 2002/08/22 20:19:54 $      */  [1]

#include "su.h"                                                     [2]
#include "segy.h"

/*********************** self documentation **********************/ [3]
char *sdoc[] = {
"                                                                ",
" SUMUTE - ......                                                ",
"                                                                ",
" sumute <stdin >stdout                                          ",
"                                                                ",
" Required parameters:                                           ",
"        none                                                    ",
"                                                                ",
" Optional parameters:                                           ",
"        ...                                                     ",
"                                                                ",
" Trace header fields accessed: ns                               ",
" Trace header fields modified: none                             ",
"                                                                ",
NULL};
/**************** end self doc ***********************************/

/* Credits:
 *
 *       CWP: Jack Cohen, John Stockwell
 */


segy tr;                                                            [4]

main(int argc, char **argv)
{
        int ns;                /* number of samples          */      [5]
        ...


        /* Initialize */
        initargs(argc, argv);                                        [6]
        requestdoc(1);                                               [7]

        /* Get parameters */
        if (!getparint("ntaper", &ntaper))       ntaper = 0;         [8]


        /* Get info from first trace */
        if (!gettr(&tr)) err("can't read first trace");              [9]
        if (!tr.dt) err("dt header field must be set");              [10]

        /* Loop over traces */
        do {                                                         [11]
                int nt     = (int) tr.ns;                            [12]

                if (below == 0) {                                    [13]
                        nmute = NINT((t - tmin)/dt);
                        memset((void *) tr.data, (int) '\0', nmute*FSIZE);
                        for (i = 0; i < ntaper; ++i)
                                tr.data[i+nmute] *= taper[i];
                } else {
                        nmute = NINT((nt*dt - t)/dt);
                        memset((void *) (tr.data+nt-nmute),
                                        (int) '\0', nmute*FSIZE);
                        for (i = 0; i < ntaper; ++i)
                                tr.data[nt-nmute-1-i] *= taper[i];
                }
                puttr(&tr);                                           [14]
        } while (gettr(&tr));                                         [15]

        return EXIT_SUCCESS;                                          [16]
}
```

Discussion of numbered lines:

1. We maintain the internal versions of the codes with the UNIX utility RCS. This item shows the string template for RCS.
2. The file `su.h` includes (directly or indirectly) all our locally defined macros and prototypes. The file `segy.h` has the definitions for the trace header fields.
3. The starred lines delimit the "self-doc" information—include them exactly as you find them in the codes since they are used by the automatic documentation shells. The style of the self-doc shown is typical except that often additional usage information is shown at the bottom and, of course, often there are more options. Look at some existing codes for ideas.
4. This is an external declaration of an SU (SEG-Y) trace buffer. It is external to avoid wasting stack space.
5. We usually describe the global variables at the time of declaration. Examine codes related to yours to increase consistency of nomenclature (there is no official SU naming standard).

6. The `initargs` subroutine sets SU's command line passing facility (see page 75).
7. The `requestdoc` subroutine call specifies the circumstances under which self-doc will be echoed to the user. The argument '1' applies to the typical program that uses only standard input (i.e. <) to read an SU trace file. Use '0' for codes that create synthetic data (like `suplane`) and '2' for codes that require two input files (we could say "et cetera," but there are no existing SU mains that require *three* or more input files).
8. This is typical code for reading 'parameters from the command line. Interpret it like this: "If the user did not specify a value, then use the default value." The subroutine must be type-specific, here we are getting an *integer* parameter.
9. Read the first trace, exit if empty. The subroutine `fgettr` "knows about" the SU trace format. Usually the trace file is read from standard input and then we use `gettr` which is a macro based on `fgettr` defined in `su.h`. Note that this code implies that the first trace is read into the trace buffer (here called `tr`), therefore we will have to process this trace before the next call to `fgettr`.
10. We've read that first trace because, we need to get some trace parameters from the first trace header. Usually these are items like the number of samples (`tr.ns`) and/or the sampling interval (`tr.dt`) that, by the SEGY-Y standard, are the same for all traces.
11. Since the first trace has been (typically) read before the main processing loop starts, we use a "do-while" that reads a new trace at the *bottom* of the loop.
12. We favor using *local* variables where permitted.
13. This is the seismic algorithm–here incomplete. We've left in some of the actual `sumute` code because it happens to contains lines that will be useful in the new code, we'll be writing below. You may want to call a subroutine here to do the real work.
14. `fputtr` and `puttr` are the output analogs of `fgettr` and `gettr`.
15. The loop end. `gettr` returns a 0 when the trace file is exhausted and the processing then stops.
16. This is an ANSI-C macro conventionally used to indicate successful program termination.

10.3 Writing a new program: `suvlength`

A user asked about SU processing for variable length traces. At his institute, data are collected from time of excitation to a variable termination time. The difficulty is that SU processing is based on the SEG-Y standard which mandates that all traces in the data set be of the same length. Rather than contemplating changing all of SU, it seems to us that the solution is to provide a program that converts the variable length data to fixed length data by padding with zeroes where necessary at the end of the traces—let's name this new program `suvlength`. We can make the length of the output traces a user parameter. If there is a reasonable choice, it makes sense to provide a default value for parameters. Here, using the length of the first trace seems the best choice since that value can be ascertained before the main processing loop starts.

So far, so good. But now our plan runs into a serious snag: the fundamental trace getting facility, `gettr`, itself assumes fixed length traces (or perhaps we should say that `gettr` deliberately enforces the fixed length trace standard). But, if you think about it, you'll realize that `gettr` itself has to take special measures with the *first* trace to figure out its length. All we have to do is make a new trace getting routine that employs that first trace logic for *every* trace. Here, we'll suppress the details of writing the "fvgettr" subroutine and turn to converting the template above into the new `suvlength` code:

```
/* SUVLENGTH: $Revision: 1.20 $ ; $Date: 2002/08/22 20:19:54 $   */

#include "su.h"
```

```
#include "segy.h"

/*********************** self documentation **********************/
char *sdoc[] = {
"                                                                  ",
" SUVLENGTH - Adjust variable length traces to common length     ",
"                                                                  ",
" suvlength <variable_length_traces >fixed_length_traces         ",
"                                                                  ",
" Required parameters:                                            ",
"         none                                                     ",
"                                                                  ",
" Optional parameters:                                            ",
"        ns      output number of samples (default: 1st trace ns)",
NULL};
/**************** end self doc ***********************************/

/* Credits:
 *        CWP: Jack Cohen, John Stockwell
 */

/* prototype */
int fvgettr(FILE *fp, segy *tp);

segy tr;

main(int argc, char **argv)
{
        int ns;        /* number of samples on output traces  */


        /* Initialize */
        initargs(argc, argv);
        requestdoc(1);

        /* Get parameters */
        ...

        /* Get info from first trace */
        ...

        ...

        return EXIT_SUCCESS;                                          [16]
}

/* fvgettr code goes here */
        ...
```

Now we run into a small difficulty. Our only parameter has a default value that is obtained only after we read in the first trace. The obvious solution is to reverse the parameter getting and the trace getting in the template. Thus we resume:

```
        /* Get info from first trace and set ns */
        if (!fvgettr(stdin, &tr))  err("can't get first trace");
        if (!getparint("ns", &ns))    ns = tr.ns;

        /* Loop over the traces */
        do {
                int nt = tr.ns;
```

Now comes the actual seismic algorithm—which is rather trivial in the present case: add zeroes to the end of the input trace if the output length is specified greater than the input length. We could write a simple loop to do the job, but the task is done most succinctly by using the ANSI-C routine `memset`. However, we confess that unless we've used it recently, we usually forget how to use this routine. One solution is to `cd` to the `su/main` directory and use `grep` to find other uses of `memset`. When we did this, we found that `sumute` had usage closest to what we needed and that is why we started from a copy of that code. Here is the complete main for `suvlength`:

```
/* SUVLENGTH: $Revision: 1.20 $ ; $Date: 2002/08/22 20:19:54 $        */

#include "su.h"
#include "segy.h"

/*********************** self documentation **********************/
char *sdoc[] = {
"                                                                   ",
" SUVLENGTH - Adjust variable length traces to common length      ",
"                                                                   ",
" suvlength <vdata >stdout                                         ",
"                                                                   ",
" Required parameters:                                              ",
"         none                                                      ",
"                                                                   ",
" Optional parameters:                                              ",
"          ns     output number of samples (default: 1st trace ns)",
NULL};
/**************** end self doc ***********************************/

/* Credits:
 *        CWP: Jack Cohen, John Stockwell
 *
 * Trace header fields accessed:  ns
 * Trace header fields modified:  ns
 */

/* prototype */
int fvgettr(FILE *fp, segy *tp);

segy tr;

main(int argc, char **argv)
{
        int ns;                /* samples on output traces        */
        /* Initialize */
        initargs(argc, argv);
        requestdoc(1);


        /* Get info from first trace */
        if (!fvgettr(stdin, &tr))  err("can't get first trace");
        if (!getparint("ns", &ns))    ns = tr.ns;


        /* Loop over the traces */
        do {
                int nt = tr.ns;

                if (nt < ns) /* pad with zeros */
                        memset((void *)(tr.data + nt), '\0', (ns-nt)*FSIZE);
                tr.ns = ns;
                puttr(&tr);
        } while (fvgettr(stdin, &tr));

        return EXIT_SUCCESS;
}


#include "header.h"

/* fvgettr - get a segy trace from a file by file pointer (nt can vary)
 *
 * Returns:
 *      int: number of bytes read on current trace (0 after last trace)
 *
 * Synopsis:
 *      int fvgettr(FILE *fp, segy *tp)
 *
 * Credits:
 *      Cloned from .../su/lib/fgettr.c
 */

int fvgettr(FILE *fp, segy *tp)
   ...
```

Remark: In the actual SU, the subroutine `fvgettr` has been extracted as a library function and we also made a convenience macro `vgettr` for the case of standard input. But these are secondary considerations that don't arise for most applications.

For any new SU code, one should provide an example shell program to show how the new code is to be used. Here is such a program for X Windows graphics:

```
#! /bin/sh
# Trivial test of suvlength with X Windows graphics

WIDTH=700
HEIGHT=900
WIDTHOFF=50
```

```
HEIGHTOFF=20

>tempdata
>vdata
suplane >tempdata  # default is 32 traces with 64 samples per trace
suplane nt=72 >>tempdata
suvlength <tempdata ns=84 |
sushw key=tracl a=1 b=1 >vdata

# Plot the data
suxwigb <vdata \
        perc=99 title="suvlength test"\
        label1="Time (sec)" label2="Traces" \
        wbox=$WIDTH hbox=$HEIGHT xbox=$WIDTHOFF ybox=$HEIGHTOFF &

# Remove #comment sign on next line to test the header
#sugethw <vdata tracl ns | more
```

Appendix A

Obtaining and Installing SU

The SU package contains seismic processing programs along with libraries of scientific routines, graphics routines and routines supporting the SU coding conventions. The package is available by anonymous ftp at the site ftp.cwp.mines.edu (138.67.12.4). The directory path is pub/cwpcodes. The package may also be obtained on the World Wide Web at http://www.cwp.mines.edu/cwpcodes. Take the files:

1. README_BEFORE_UNTARRING
2. untar_me_first.xx.tar.Z
3. cwp.su.all.xx.tar.Z

Here the `xx` denotes the number of the current release. An incremental update is also available for updating the previous release `yy` to the current release `xx`. Take the files:

1. README_BEFORE_UNTARRING
2. README_UPDATE
3. untar_me_first.xx.tar.Z
4. update.yy.to.xx.tar.Z
5. update.list

If you find that `ftp` times out during the transmission of the files, the package is available in smaller pieces in the subdirectory `outside_usa`.
For readers who are not familiar with anonymous ftp, an annotated transaction listing follows in section A.1.

A.1 Obtaining files by anonymous ftp

```
Type:
% ftp 138.67.12.4                      —   138.67.12.4 is our ftp site
username: anonymous                    —   your username is "anonymous"
password: yourname@your.machine.name   —   type anything here

ftp>                                   —   this is the prompt you see
                                           when you are in ftp
```

You are now logged in via ftp to the CWP anonymous ftp site. You may type:

ftp> ls	—	to see the contents of the directories
ftp> cd dirname	—	to change directories to "dirname"
ftp> binary	—	to set "binary mode" for transferring files You must do this before you try to transfer any binary file. This includes all files with the form some_name.tar.Z extension.
ftp> get filename	—	to transfer "filename" from our site to your machine
ftp> mget pattern*	—	to transfer all files with names of the "pattern*"
For example:		
ftp> mget *.tar.Z	—	will transfer all files with the form of name.tar.Z to your machine. You will be asked whether you really want each file of this name pattern transferred, before ftp actually does it.
ftp> bye	—	to exit from ftp

A.2 Requirements for installing the package

The only requirements for installing the package are:

1. A machine running the UNIX operating system.
2. An ANSI C compiler.
3. A version of make which supports include files
4. 16-60 megabytes (depending on system) of disk space for the source and compiled binary. If space is an issue, then the compiled binaries may be "stripped" by cd'ing to $CWPROOT/bin and typing "strip *".

The package has been successfully installed on:

- IBM RS6000
- SUN SPARC STATIONS
- HP 9000 series machines
- HP Apollo
- NeXT
- Convex
- DEC
- Silicon Graphics
- PC's running LINUX, PRIME TIME, SCO, FREE BSD, ESIX, and NeXTSTEP 486

There are README files in the distribution with special notes about some of these platforms. We depend on the SU user community to alert us to installation problems, so if you run into difficulties, please let us know.

The distribution contains a series of files that detail the installation process. Read them in the following order:

```
LEGAL_STATEMENT --- license,  legal statement
README_BEFORE_UNTARRING --- initial information
README_FIRST --- general information
README_TO_INSTALL --- installation instructions
Portability/README_*    --- portability information for various platforms
README_GETTING_STARTED --- how to begin using the codes
```

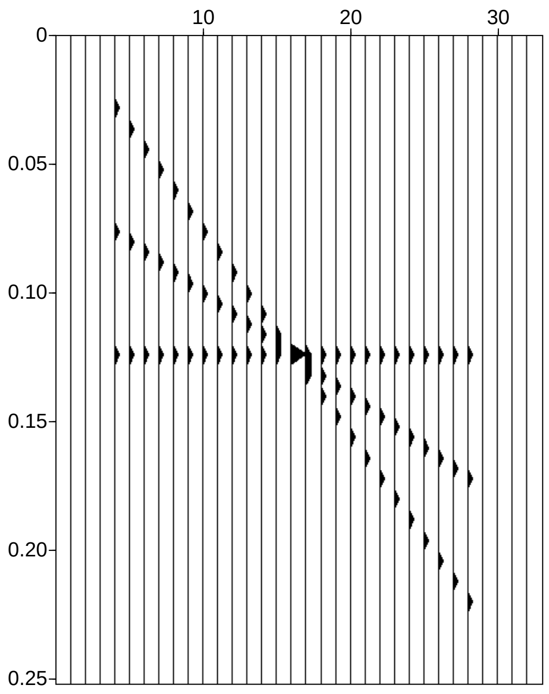

Figure A.1: Output of the `suplane` pipeline.

Many of these files are contained within untar_me_first.xx.tar.Z.

A.3 A quick test

Once you have completed the installation, here is a quick test you can make to see if you have a functioning seismic system. For an X-windows machine, the "pipeline"

```
suplane | suxwigb &
```

should produce the graphic shown in Figure A.1. If you have a PostScript printer, then you should get a hard copy version with the pipeline

```
suplane | supswigb | lpr
```

If you have Display PostScript, or a PostScript previewer, then to get a screen display, replace the `lpr` command in the pipeline by the command that opens a PostScript file, for example:

```
suplane | supswigb | ghostview -
```

Another set of test pipelines are

```
susynlv | supsimage | lpr
```

```
susynlv | supswigb | ghostview -
```

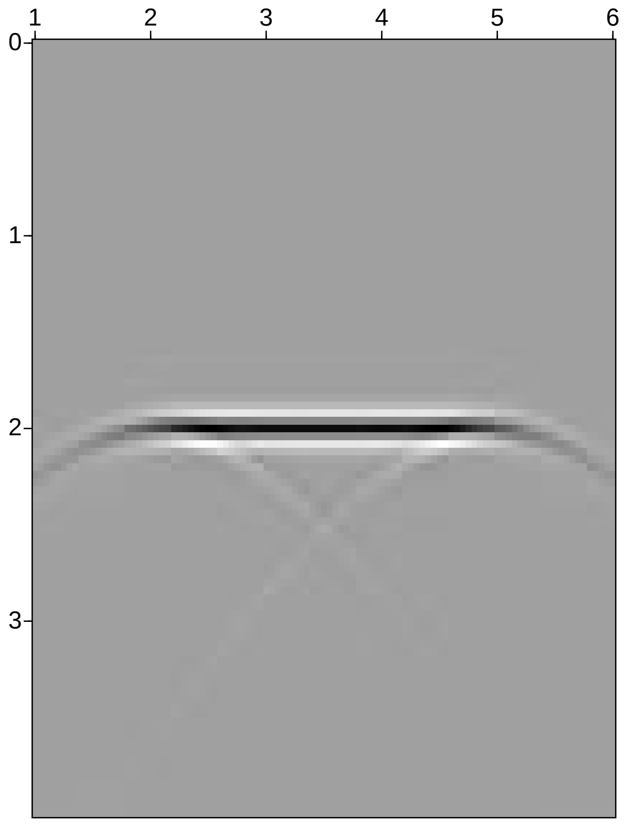

Figure A.2: Output of the `susynlv` pipeline.

Appendix B

Help Facililties

B.1 Suhelp

The full text of the output from:

```
% suhelp
```

```
CWP PROGRAMS: (no self-documentation)
ctrlstrip    downfort    fcat    isatty    maxints    pause    t    upfort

PAR PROGRAMS: (programs with self-documentation)
a2b         grm      rayt2d      smoothint2    unisam2     xy2z
b2a         h2b      recast      subset        vel2stiff    z2xyz
dzdv         kaperture    regrid3     swapbytes     velconv
farith        makevel      resamp      transp        velpert
ftnstrip      mkparfile    smooth2     unif2         vtlvz
ftnunstrip    prplot      smooth3d    unisam        wkbj


press return key to continue
SU PROGRAMS: (self-documented programs for SU data )
  SU data is "SEGY" data run through "segyclean"
bhedtopar   sudmofkcw   sukdmig3d   suop         sustkvel
dt1tosu   sudmotivz   sukdsyn2d   suop2         sustolt
segdread   sudmotx   sukfilter   supack1       sustrip
segyclean   sudmovz   sukfrac   supack2       suswapbytes
segyhdrs   suea2df   sukill   supaste       susyncz
segyread   suedit   sulog   supef         susynlv
segywrite   sufdmod2   sumax   supgc         susynlvcw
setbhed   sufft   sumean   supickamp    susynlvfti
su3dchart   sufilter   sumedian   suplane       susynvxz
suabshw   suflip   sumigfd   suput         susynvxzcs
suacor   sufrac   sumigffd   suquantile   sutab
suaddhead   sufxdecon   sumiggbzo   suradon       sutaper
suaddnoise   sugabor   sumigprefd   suramp        sutaup
suamp   sugain   sumigpreffd   surange       sutsq
suascii   sugazmig   sumigprepspi   surecip       suttoz
suattributes   suget   sumigpresp   sureduce      sutvband
```

```
suazimuth   sugethw   sumigps   surelan       suunpack1
subfilt   suharlan   sumigpspi   suresamp      suunpack2
suchart   suhilb   sumigpsti   suresstat     suvcat
suchw   suifft   sumigsplit   sushape       suvelan
sucommand   suilog   sumigtk   sushift       suvibro
suconv   suimp2d   sumigtopo2d   sushw         suvlength
sudatumfd   suimp3d   sumix   susort        suweight
sudatumk2dr   suinterp   sumute   suspecfk      suwind
sudatumk2ds   suintvel   sunmo   suspecfx      suxcor
sudipfilt   suinvvxzco   sunormalize   suspeck1k2    suxedit
sudivcor   suinvzco3d   sunull   suspike       suzero
sudivstack   suk1k2filter   suocext   sustack
sudmofk   sukdmig2d   suoldtonew   sustatic


press return key to continue

Delaunay Triangulation Materials:
gbbeam    normray    tri2uni   trimodel    triray    triseis    uni2tri

Tetrahedra Materials:
sutetraray    tetramod

X-windows GRAPHICS for Delaunay Triangulation:
sxplot

press return key to continue

Straight X-windows GRAPHICS:
lcmap    scmap      xepsb    ximage    xpsp
lprop    xcontour    xepsp    xpicker    xwigb

X-Toolkit based X-windows GRAPHICS:
xgraph    xmovie    xrects

Motif-based X-windows GRAPHICS:
fftlab

X-windows GRAPHICS: for SU ("segyclean"-ed SEGY)  data sets
suxcontour    suximage    suxmovie    suxwigb
suxgraph      suxmax     suxpicker

PostScript GRAPHICS:
psbbox       pscube    psgraph    pslabel      psmerge    pswigb
pscontour    psepsi    psimage    psmanager    psmovie    pswigp


PostScript GRAPHICS: for SU ("segyclean"-ed SEGY) data sets
supscontour    supsgraph    supsmax    supswigb
supscube       supsimage    supsmovie    supswigp

press return key to continue
```

```
Wavelet Packet Compression:
wpc1comp2    wpc1uncomp2

Wavelet Packet Compression:
wpccompress    wpcuncompress

2D Discrete Cosine Transform Compression:
dctcomp    dctuncomp   entropy    wptcomp    wptuncomp    wtcomp    wtuncomp

press return key to continue

CWP SHELLS SCRIPTS:
Grep          cwpfind      overwrite      time_now      weekday
argv          dirtree      precedence     todays_date     why
copyright     filetype     replace      usernames      zap
cpall         newcase      this_year      varlist

PAR SHELLS SCRIPTS:
gendocs     updatedoc     updatedocall     updatehead

POSTSCRIPT RELATED SHELLS SCRIPTS:
merge2     merge4

SU SHELLS SCRIPTS:
lookpar     rmaxdiff     sudiff     sufind      suhelp    unglitch
maxdiff     suagc     sudoc      sufind2      sukeyword
recip       suband     suenv      sugendocs     suname

press return key to continue

Use: "suname" to list the name and a brief description of all
of the CWP codes.

Use "sufind" to find programs by keyword/name fragment.

Use: "gendocs" to compile a LaTeX document listing all self-docs.

Use: "sukeyword" to find the SEGY header field keyword definitions.

Type: "program_name <CR>" to view its self documentation

Note: not all programs listed here have the self-documentation feature
Type:  "sudoc  program_name" to list information for these programs.

For answers to Frequently Asked Questions, see the contents of:
 /usr/local/cwp/src/faq
```

B.2 Suname

The full text of the output from:

```
% suname


 -----  CWP Free Programs -----
CWPROOT=/usr/local/cwp

Mains:

In CWPROOT/src/cwp/main:
* CTRLSTRIP - Strip non-graphic characters
* DOWNFORT - change Fortran programs to lower case, preserving strings
* FCAT - fast cat with 1 read per file
* ISATTY - pass on return from isatty(2)
* MAXINTS - Compute maximum and minimum sizes for integer types
* PAUSE - prompt and wait for user signal to continue
* T - time and date for non-military types
* UPFORT - change Fortran programs to upper case, preserving strings

In CWPROOT/src/par/main:
A2B - convert ascii floats to binary
B2A - convert binary floats to ascii
DZDV - determine depth derivative with respect to the velocity ",
FARITH - File ARITHmetic -- perform simple arithmetic with binary files
FTNSTRIP - convert a file of binary data plus record delimiters created
FTNUNSTRIP - convert C binary floats to Fortran style floats
GRM - Generalized Reciprocal refraction analysis for a single layer
H2B - convert 8 bit hexidecimal floats to binary
KAPERTURE - generate the k domain of a line scatterer for a seismic array
MAKEVEL - MAKE a VELocity function v(x,y,z)
MKPARFILE - convert ascii to par file format
PRPLOT - PRinter PLOT of 1-D arrays f(x1) from a 2-D function f(x1,x2)
RAYT2D - traveltime Tables calculated by 2D paraxial RAY tracing
RECAST - RECAST data type (convert from one data type to another)
REGRID3 - REwrite a [ni3][ni2][ni1] GRID to a [no3][no2][no1] 3-D grid
RESAMP - RESAMPle the 1st dimension of a 2-dimensional function f(x1,x2)
SMOOTH2 --- SMOOTH a uniformly sampled 2d array of data, within a user-
SMOOTH3D - 3D grid velocity SMOOTHing by the damped least squares
SMOOTHINT2 --- SMOOTH non-uniformly sampled INTerfaces, via the damped
SUBSET - select a SUBSET of the samples from a 3-dimensional file
SWAPBYTES - SWAP the BYTES of various  data types
TRANSP - TRANSPose an n1 by n2 element matrix
UNIF2 - generate a 2-D UNIFormly sampled velocity profile from a layered
UNISAM - UNIformly SAMple a function y(x) specified as x,y pairs
UNISAM2 - UNIformly SAMple a 2-D function f(x1,x2)
VEL2STIFF - Transforms VELocities, densities, and Thomsen parameters
VELCONV - VELocity CONVersion
VELPERT - estimate velocity parameter perturbation from covariance
VTLVZ -- Velocity as function of Time for Linear V(Z);
WKBJ - Compute WKBJ ray theoretic parameters, via finite differencing
XY2Z - converts (X,Y)-pairs to spike Z values on a uniform grid
Z2XYZ - convert binary floats representing Z-values to ascii

In CWPROOT/src/psplot/main:
```

```
PSBBOX - change BoundingBOX of existing PostScript file
PSCONTOUR - PostScript CONTOURing of a two-dimensional function f(x1,x2)
PSCUBE - PostScript image plot of a data CUBE
PSEPSI - add an EPSI formatted preview bitmap to an EPS file
PSGRAPH - PostScript GRAPHer
PSIMAGE - PostScript IMAGE plot of a uniformly-sampled function f(x1,x2)
PSLABEL - output PostScript file consisting of a single TEXT string
PSMANAGER - printer MANAGER for HP 4MV and HP 5Si Mx Laserjet
PSMERGE - MERGE PostScript files
PSMOVIE - PostScript MOVIE plot of a uniformly-sampled function f(x1,x2,x3)
PSWIGB - PostScript WIGgle-trace plot of f(x1,x2) via Bitmap
PSWIGP - PostScript WIGgle-trace plot of f(x1,x2) via Polygons

In CWPROOT/src/xplot/main:
* LCMAP - List Color Map of root window of default screen
* LPROP - List PROPerties of root window of default screen of display
* SCMAP - set default standard color map (RGB_DEFAULT_MAP)
XCONTOUR - X CONTOUR plot of f(x1,x2) via vector plot call
* XESPB - X windows display of Encapsulated PostScript as a single Bitmap
* XEPSP - X windows display of Encapsulated PostScript
XIMAGE - X IMAGE plot of a uniformly-sampled function f(x1,x2)
XIMAGE - X IMAGE plot of a uniformly-sampled function f(x1,x2)
XPICKER - X wiggle-trace plot of f(x1,x2) via Bitmap with PICKing
* XPSP - Display conforming PostScript in an X-window
XWIGB - X WIGgle-trace plot of f(x1,x2) via Bitmap

In CWPROOT/src/Xtcwp/main:
XGRAPH - X GRAPHer
XMOVIE - image one or more frames of a uniformly sampled function f(x1,x2)
XRECTS - plot rectangles on a two-dimensional grid

In CWPROOT/src/Xmcwp/main:
* FFTLAB - Motif-X based graphical 1D Fourier Transform

In CWPROOT/src/su/graphics/psplot:
SUPSCONTOUR - PostScript CONTOUR plot of a segy data set
SUPSCUBE - PostScript CUBE plot of a segy data set
SUPSGRAPH - PostScript GRAPH plot of a segy data set
SUPSIMAGE - PostScript IMAGE plot of a segy data set
SUPSMAX - PostScript of the MAX, min, or absolute max value on each trace
SUPSMOVIE - PostScript MOVIE plot of a segy data set
SUPSWIGB - PostScript Bit-mapped WIGgle plot of a segy data set
SUPSWIGP - PostScript Polygon-filled WIGgle plot of a segy data set

In CWPROOT/src/su/main:
BHEDTOPAR - convert a Binary tape HEaDer file to PAR file format
DT1TOSU - Convert ground-penetrating radar data in the Sensors & Software
SEGDREAD - read an SEG-D tape
SEGYCLEAN - zero out unassigned portion of header
SEGYHDRS - make SEG-Y ascii and binary headers for segywrite
SEGYREAD - read an SEG-Y tape
SEGYWRITE - write an SEG-Y tape
```

```
SETBHED - SET the fields in a SEGY Binary tape HEaDer file, as would be
SU3DCHART - plot x-midpoints vs. y-midpoints for 3-D data
SUABSHW - replace header key word by its absolute value
SUACOR - auto-correlation
SUADDHEAD - put headers on bare traces and set the tracl and ns fields
SUADDNOISE - add noise to traces
SUAMP - output amp, phase, real or imag trace from
SUASCII - print non zero header values and data
SUATTRIBUTES - trace ATTRIBUTES instantanteous amplitude, phase,
SUAZIMUTH - compute trace AZIMUTH given the sx,sy,gx,gy header fields
SUBFILT - apply Butterworth bandpass filter
SUCHART - prepare data for x vs. y plot
SUCHW - Change Header Word using one or two header word fields
SUCOMMAND - pipe traces having the same key header word to command
SUCONV - convolution with user-supplied filter
SUDATUMFD - 2D zero-offset Finite Difference acoustic wave-equation
SUDATUMK2DR - Kirchhoff datuming of receivers for 2D prestack data
SUDATUMK2DS - Kirchhoff datuming of sources for 2D prestack data
SUDIPFILT - DIP--or better--SLOPE Filter in f-k domain
SUDIVCOR - Divergence (spreading) correction
SUDIVSTACK -  Diversity Stacking using either average power or peak
SUDMOFK - DMO via F-K domain (log-stretch) method for common-offset gathers
SUDMOFKCW - converted-wave DMO via F-K domain (log-stretch) method for
SUDMOTIVZ - DMO for Transeversely Isotropic V(Z) media for common-offset
SUDMOTX - DMO via T-X domain (Kirchhoff) method for common-offset gathers
SUDMOVZ - DMO for V(Z) media for common-offset gathers
SUEA2DF - SU version of (an)elastic anisotropic 2D finite difference
SUEDIT - examine segy diskfiles and edit headers
SUFDMOD2 - Finite-Difference MODeling (2nd order) for acoustic wave equation
SUFFT - fft real time traces to complex frequency traces
SUFILTER - applies a zero-phase, sine-squared tapered filter
SUFLIP - flip a data set in various ways
SUFRAC -- take general (fractional) time derivative or integral of
SUFXDECON - random noise attenuation by FX-DECONvolution
SUGABOR -  Outputs a time-frequency representation of seismic data via
SUGAIN - apply various types of gain to display traces
SUGAZMIG - SU version of Jeno GAZDAG's phase-shift migration
SUGET  - Connect SU program to file descriptor for input stream.
SUGETHW - sugethw writes the values of the selected key words
SUHARLAN - signal-noise separation by the invertible linear
SUHILB - Hilbert transform
SUIFFT - fft complex frequency traces to real time traces
SUILOG -- time axis inverse log-stretch of seismic traces
SUIMP2D - generate shot records for a line scatterer
SUIMP3D - generate inplane shot records for a point
SUINTERP - interpolate traces using automatic event picking
SUINTVEL - convert stacking velocity model to interval velocity model
SUINVXZCO - Seismic INVersion of Common Offset data for a smooth
SUINVZCO3D - Seismic INVersion of Common Offset data with V(Z) velocity
SUK1K2FILTER - symmetric box-like K-domain filter defined by the
SUKDMIG2D - Kirchhoff Depth Migration of 2D poststack/prestack data
SUKDMIG3D - Kirchhoff Depth Migration of 3D poststack/prestack data
```

```
SUKDSYN2D - Kirchhoff Depth SYNthesis of 2D seismic data from a
SUKFILTER - radially symmetric K-domain, sin^2-tapered, polygonal
SUKFRAC - apply FRACtional powers of i|k| to data, with phase shift
SUKILL - zero out traces
SULOG -- time axis log-stretch of seismic traces
SUMAX - get trace by trace local/global maxima, minima, or absolute maximum
SUMEAN - get the mean values of data traces ",
SUMEDIAN - MEDIAN filter about a user-defined polygonal curve with
SUMIGFD - 45-90 degree Finite difference migration for zero-offset data.
SUMIGFFD - Fourier finite difference migration for
SUMIGGBZO - MIGration via Gaussian Beams of Zero-Offset SU data
SUMIGPREFD --- The 2-D prestack common-shot 45-90 degree
SUMIGPREFFD - The 2-D prestack common-shot Fourier finite-difference
SUMIGPREPSPI --- The 2-D PREstack commom-shot Phase-Shift-Plus
SUMIGPRESP - The 2-D prestack common-shot split-step Fourier ",
SUMIGPS - MIGration by Phase Shift with turning rays
SUMIGPSPI - Gazdag's phase-shift plus interpolation migration
SUMIGPSTI - MIGration by Phase Shift for TI media with turning rays
SUMIGSPLIT - Split-step depth migration for zero-offset data.
SUMIGTK - MIGration via T-K domain method for common-midpoint stacked data
SUMIGTOPO2D - Kirchhoff Depth Migration of 2D postack/prestack data
SUMIX - compute weighted moving average (trace MIX) on a panel
SUMUTE - mute above (or below) a user-defined polygonal curve with ",
NMO - NMO for an arbitrary velocity function of time and CDP
SUNORMALIZE - Trace balancing by rms, max, or median ",
SUNULL - create null (all zeroes) traces
SUOCEXT - smaller offset extrapolation via Offset Continuation
SUOLDTONEW - convert existing su data to xdr format
SUOP - do unary arithmetic operation on segys
SUOP2 - do a binary operation on two data sets
SUPACK1 - pack segy trace data into chars
SUPACK2 - pack segy trace data into 2 byte shorts
SUPASTE - paste existing SEGY headers on existing data
SUPEF - Wiener predictive error filtering
SUPGC   -   Programmed Gain Control--apply agc like function
SUPICKAMP - pick amplitudes within user defined and resampled window
SUPLANE - create common offset data file with up to 3 planes
SUPUT - Connect SU program to file descriptor for output stream.
SUQUANTILE - display some quantiles or ranks of a data set
SURADON - compute forward or reverse Radon transform or remove multiples
SURAMP - Linearly taper the start and/or end of traces to zero.
SURANGE - get max and min values for non-zero header entries
SURECIP - sum opposing offsets in prepared data (see below)
SUREDUCE - convert traces to display in reduced time ",
SURELAN - compute residual-moveout semblance for cdp gathers based
SURESAMP - Resample in time
SURESSTAT - Surface consistent source and receiver statics calculation
SUSHAPE - Wiener shaping filter
SUSHIFT - shifted/windowed traces in time
SUSHW - Set one or more Header Words using trace number, mod and
SUSORT - sort on any segy header keywords
SUSPECFK - F-K Fourier SPECtrum of data set
```

```
SUSPECFX - Fourier SPECtrum (T -> F) of traces
SUSPECK1K2 - 2D (K1,K2) Fourier SPECtrum of (x1,x2) data set
SUSPIKE - make a small spike data set
SUSTACK - stack adjacent traces having the same key header word
SUSTATIC - Elevation static corrections, apply corrections from
SUSTKVEL - convert constant dip layer interval velocity model to the
SUSTOLT - Stolt migration for stacked data or common-offset gathers
SUSTRIP - remove the SEGY headers from the traces
SUSWAPBYTES - SWAP the BYTES in SU data to convert data from big endian
SUSYNCZ - SYNthetic seismograms for piecewise constant V(Z) function
SUSYNLV - SYNthetic seismograms for Linear Velocity function
SUSYNLVCW - SYNthetic seismograms for Linear Velocity function
SUSYNLVFTI - SYNthetic seismograms for Linear Velocity function in a factored
SUSYNVXZ - SYNthetic seismograms of common offset V(X,Z) media via
SUSYNVXZCS - SYNthetic seismograms of common shot in V(X,Z) media via
SUTAB - print non zero header values and data for non-graphic terminals
SUTAPER - Taper the edge traces of a data panel to zero.
SUTAUP - forwared and inverse T-X and F-K global slant stacks
SUTSQ -- time axis time-squared stretch of seismic traces
SUTTOZ - resample from time to depth
SUTVBAND - time-variant bandpass filter (sine-squared taper)
SUUNPACK1 - unpack segy trace data from chars to floats
SUUNPACK2 - unpack segy trace data from shorts to floats
SUVCAT -  append one data set to another, with or without an  ",
SUVELAN - compute stacking velocity semblance for cdp gathers
SUVIBRO - Generates a Vibroseis sweep (linear, linear-segment,
SUVLENGTH - Adjust variable length traces to common length
SUWEIGHT - weight traces by header parameter, such as offset
SUWIND - window traces by key word
SUXCOR - correlation with user-supplied filter
SUXEDIT - examine segy diskfiles and edit headers
SUZERO -- zero-out data within a time window

In CWPROOT/src/su/graphics/psplot:
SUPSCONTOUR - PostScript CONTOUR plot of a segy data set
SUPSCUBE - PostScript CUBE plot of a segy data set
SUPSGRAPH - PostScript GRAPH plot of a segy data set
SUPSIMAGE - PostScript IMAGE plot of a segy data set
SUPSMAX - PostScript of the MAX, min, or absolute max value on each trace
SUPSMOVIE - PostScript MOVIE plot of a segy data set
SUPSWIGB - PostScript Bit-mapped WIGgle plot of a segy data set
SUPSWIGP - PostScript Polygon-filled WIGgle plot of a segy data set

In CWPROOT/src/su/graphics/xplot:
SUXCONTOUR - X CONTOUR plot of Seismic UNIX tracefile via vector plot call
SUXGRAPH - X-windows GRAPH plot of a segy data set
SUXIMAGE - X-windows IMAGE plot of a segy data set
SUXMAX - X-windows graph of the MAX, min, or absolute max value on
SUXMOVIE - X MOVIE plot of a segy data set
SUXPICKER - X-windows  WIGgle plot PICKER of a segy data set
SUXWIGB - X-windows Bit-mapped WIGgle plot of a segy data set
```

```
In CWPROOT/src/tri/main:
GBBEAM - Gaussian beam synthetic seismograms for a sloth model
NORMRAY - dynamic ray tracing for normal incidence rays in a sloth model
TRI2UNI - convert a TRIangulated model to UNIformly sampled model
TRIMODEL - make a triangulated sloth (1/velocity^2) model
TRIRAY - dynamic RAY tracing for a TRIangulated sloth model
TRISEIS - Gaussian beam synthetic seismograms for a sloth model
UNI2TRI - convert UNIformly sampled model to a TRIangulated model

In CWPROOT/src/xtri:
SXPLOT - X Window plot a triangulated sloth function s(x1,x2)

In CWPROOT/src/tri/graphics/psplot:
SPSPLOT - plot a triangulated sloth function s(x,z) via PostScript

In CWPROOT/src/comp/dct/main:
DCTCOMP - Compression by Discrete Cosine Transform
DCTUNCOMP - Discrete Cosine Transform Uncompression
ENTROPY - compute the ENTROPY of a signal
WPTCOMP - Compression by Wavelet Packet Compression
WPTUNCOMP - Uncompress  WPT compressed data
WTCOMP - Compression by Wavelet Transform
WTUNCOMP - UNCOMPression of WT compressed data

In CWPROOT/src/comp/dwpt/1d/main:
WPC1COMP2 --- COMPress a 2D seismic section trace-by-trace using
WPC1UNCOMP2 --- UNCOMPRESS a 2D seismic section, which has been

In CWPROOT/src/comp/dwpt/2d/main:
WPCCOMPRESS --- COMPRESS a 2D section using Wavelet Packets
WPCUNCOMPRESS --- UNCOMPRESS a 2D section

Shells:

In CWPROOT/src/cwp/shell:
# Grep  - recursively call egrep in pwd
# ARGV - give examples of dereferencing char **argv
# COPYRIGHT - insert CSM COPYRIGHT lines at top of files in working directory
# CPALL , RCPALL - for local and remote directory tree/file transfer
# CWPFIND - look for files with patterns in CWPROOT/src/cwp/lib
# DIRTREE - show DIRectory TREE
# FILETYPE - list all files of given type
# NEWCASE - Changes the case of all the filenames in a directory, dir
# OVERWRITE - copy stdin to stdout after EOF
# PRECEDENCE - give table of C precedences from Kernighan and Ritchie
# REPLACE - REPLACE string1 with string2  in files
# THIS_YEAR - print the current year
# TIME_NOW - prints time in ZULU format with no spaces
# TODAYS_DATE - prints today's date in ZULU format with no spaces
# USERNAMES - get list of all login names
# VARLIST - list variables used in a Fortran program
# WEEKDAY - prints today's WEEKDAY designation
# ZAP - kill processes by name

In CWPROOT/src/par/shell:
# GENDOCS - generate complete list of selfdocs in latex form
# UPDATEDOC - put self-docs in ../doc/Stripped and ../doc/Headers
# UPDATEDOCALL - put self-docs in ../doc/Stripped
# UPDATEHEAD - update ../doc/Headers/Headers.all

In CWPROOT/src/psplot/shell:
# MERGE2 - put 2 standard size PostScript figures on one page
# MERGE4 - put 4 standard size PostScript plots on one page

In CWPROOT/src/su/shell:
# LOOKPAR - show getpar lines in SU code with defines evaluated
# MAXDIFF - find absolute maximum difference in two segy data sets
# RECIP - sum opposing (reciprocal) offsets in cdp sorted data
# RMAXDIFF - find percentage maximum difference in two segy data sets
# SUAGC - perform agc on SU data
# SUBAND - Trapezoid-like Sin squared tapered Bandpass filter via  SUFILTER
# SUDIFF, SUSUM, SUPROD, SUQUO, SUPTDIFF, SUPTSUM,
# SUDOC - get DOC listing for code
# SUENV - Instantaneous amplitude, frequency, and phase via: SUATTRIBUTES
# SUFIND - get info from self-docs
# SUFIND - get info from self-docs
# SUGENDOCS - generate complete list of selfdocs in latex form
# SUHELP - list the CWP/SU programs and shells
# SUKEYWORD -- guide to SU keywords in segy.h
# SUNAME - get name line from self-docs
# UNGLITCH - zonk outliers in data

Libs:

In CWPROOT/src/cwp/lib:
ABEL - Functions to compute the discrete ABEL transform:
AIRY - Approximate the Airy functions  Ai(x), Bi(x) and their respective
ALLOC - Allocate and free multi-dimensional arrays
ANTIALIAS - Butterworth anti-aliasing filter
AXB - Functions to solve a linear system of equations Ax=b by LU
BIGMATRIX - Functions to manipulate 2-dimensional matrices that are too big
BUTTERWORTH - Functions to design and apply Butterworth filters:
COMPLEX - Functions to manipulate complex numbers
COMPLEXD - Functions to manipulate double-precision complex numbers
COMPLEXF  - Subroutines to perform operations on complex numbers.
CONVOLUTION - Compute z = x convolved with y
CUBICSPLINE - Functions to compute CUBIC SPLINE interpolation coefficients
DBLAS - Double precision Basic Linear Algebra subroutines
DGE - Double precision Gaussian Elimination matrix subroutines  adapted
PFAFFT - Functions to perform Prime Factor (PFA) FFT's, in place
FRANNOR - functions to generate a pseudo-random float normally distributed
FRANUNI - Functions to generate a pseudo-random float uniformly distributed
HANKEL - Functions to compute discrete Hankel transforms
HILBERT - Compute Hilbert transform y of x
```

```
HOLBERG1D - Compute coefficients of Holberg's 1st derivative filter
IBMFLT_ - Convert IBM tape format to floats and back
INTCUB - evaluate y(x), y'(x), y''(x), ... via piecewise cubic interpolation
INTL2B - bilinear interpolation of a 2-D array of bytes
INTLIN - evaluate y(x) via linear interpolation of y(x[0]), y(x[1]), ...
INTSINC8 - Functions to interpolate uniformly-sampled data via 8-coeff. sinc
INTTABLE8 -  Interpolation of a uniformly-sampled complex function y(x)
MKDIFF - Make an n-th order DIFFerentiator via Taylor's series method.
MKHDIFF - Compute filter approximating the bandlimited HalF-DIFFerentiator.
MKSINC - Compute least-squares optimal sinc interpolation coefficients.
MNEWT - Solve non-linear system of equations f(x) = 0 via Newton's method
PFAFFT - Functions to perform Prime Factor (PFA) FFT's, in place
POLAR - Functions to map data in rectangular coordinates to polar and vise versa
PRINTERPLOT - Functions to make a printer plot of a 1-dimensional array
QUEST - Functions to ESTimate Quantiles:
RESSINC8 - Functions to resample uniformly-sampled data  via 8-coefficient sinc
RFWTVA - Rasterize a Float array as Wiggle-Trace-Variable-Area.
RFWTVAINT - Rasterize a Float array as Wiggle-Trace-Variable-Area, with
SBLAS - Single precision Basic Linear Algebra Subroutines
SCAXIS - compute a readable scale for use in plotting axes
SGA - Single precision general matrix functions adapted from LINPACK FORTRAN:
SHFS8R - Shift a uniformly-sampled real-valued function y(x) via
SINC - Return SINC(x) for as floats or as doubles
SORT - Functions to sort arrays of data or arrays of indices
SQR - Single precision QR decomposition functions adapted from LINPACK FORTRAN:
STOEP - Functions to solve a symmetric Toeplitz linear system of equations
STRSTUFF -- STRing manuplation subs
SWAPBYTE - Functions to SWAP the BYTE order of binary data
TEMPORARY_FILENAME - Creates a file name in a user-specified directory.
TRIDIAGONAL - Functions to solve tridiagonal systems of equations Tu=r for u.
VANDERMONDE - Functions to solve Vandermonde system of equations Vx=b
WAVEFORMS Subroutines to define some wavelets for modeling of seimic
XCOR - Compute z = x cross-correlated with y
XINDEX - determine index of x with respect to an array of x values
YCLIP - Clip a function y(x) defined by linear interpolation of the
YXTOXY - Compute a regularly-sampled, monotonically increasing function x(y)
ZASC - routine to translate ncharacters from ebcdic to ascii
ZEBC - routine to translate ncharacters from ascii to ebcdic

In CWPROOT/src/par/lib:
VND *VNDop(int mode, long lwmax, int ndim, long *N, long npanels,
ATOPKGE - convert ascii to arithmetic and with error checking
DOCPKGE - Function to implement the CWP self-documentation facility
EALLOC - Allocate and free multi-dimensional arrays with error reports.
ERRPKGE - routines for reporting errors
FILESTAT - Functions to determine and output the type of a file from file
GETPARS - Functions to GET PARameterS from the command line. Numeric
MODELING - Seismic Modeling Subroutines for SUSYNLV and clones
SMOOTH - Functions to compute smoothing of 1-D or 2-D input data
SUBCALLS - routines for system functions with error checking
SYSCALLS - routines for SYSTEM CALLs with error checking
```

```
In CWPROOT/src/su/lib:
FGETTR - Routines to get an SU trace from a file
FPUTTR - Routines to put an SU trace to a file
HDRPKGE - routines to access the SEGY header via the hdr structure.
TABPLOT - TABPLOT selected sample points on selected trace
VALPKGE - routines to handle variables of type Value

In CWPROOT/src/psplot/lib:
BASIC - Basic C function interface to PostScript
PSAXESBOX - Functions to draw PostScript axes and estimate bounding box
PSAXESBOX3 -  Functions draw an axes box via PostScript, estimate bounding box
PSCAXESBOX - Draw an axes box for cube via PostScript
PSCONTOUR - draw contour of a two-dimensional array via PostScript
PSDRAWCURVE - Functions to draw a curve from a set of points
PSLEGENDBOX - Functions to draw PostScript axes and estimate bounding box
PSWIGGLE - draw wiggle-trace with (optional) area-fill via PostScript

In CWPROOT/src/xplot/lib:
AXESBOX - Functions to draw axes in X-windows graphics
COLORMAP - Functions to manipulate X colormaps:
DRAWCURVE - Functions to draw a curve from a set of points
IMAGE - Function for making the image in an X-windows image plot
LEGENDBOX - draw a labeled axes box for a legend (i.e. colorscale)
RUBBERBOX -  Function to draw a rubberband box in X-windows plots
WINDOW - Function to create a window in X-windows graphics
XCONTOUR - draw contour of a two-dimensional array via X vectorplot calls

In CWPROOT/src/Xtcwp/lib:
AXES - the Axes Widget
COLORMAP - Functions to manipulate X colormaps:
FX - Functions to support floating point coordinates in X
MISC - Miscellaneous X-Toolkit functions
RESCONV - general purpose resource type converters
RUBBERBOX -  Function to draw a rubberband box in X-windows plots

In CWPROOT/src/Xmcwp/lib:
RADIOBUTTONS -  convenience functions creating and using radio buttons
SAMPLES - Motif-based Graphics Functions

In CWPROOT/src/tri/lib:
CHECK - CHECK triangulated models
CIRCUM - define CIRCUMcircles for Delaunay triangulation
COLINEAR - determine if edges or vertecies are COLINEAR in triangulated
CREATE - create model, boundary edge triangles, edge face, edge vertex, add
DELETE - DELETE vertex, model, edge, or boundary edge from triangulated model
DTE - Distance to Edge
FIXEDGES - FIX or unFIX EDGES between verticies
INSIDE -  Is a vertex or point inside a circum circle, etc. of a triangulated
NEAREST - NEAREST edge or vertex in triangulated model
PROJECT - project to edge in triangulated model
READWRITE - READ or WRITE a triangulated model
```

```
In CWPROOT/src/cwputils:
CPUSEC - return cpu time (UNIX user time) in seconds
CPUTIME - return cpu time (UNIX user time) in seconds using ANSI C built-ins
WALLSEC - Functions to time processes
WALLTIME - Function to show time a process takes to run

In CWPROOT/src/comp/dct/lib:
DCT1 - 1D Discreet Cosine Transform Routines
DCT2 - 2D Discrete Cosine Transform Routines
DCTALLOC - ALLOCate space for transform tables for 1D DCT
GETFILTER - GET wavelet FILTER type
LCT1 - functions used to perform the 1D Local Cosine Transform (LCT)
LPRED - Lateral Prediction of Several Plane Waves
WAVEPACK1 - 1D wavelet packet transform
WAVEPACK2 - 2D Wavelet PACKet transform
WAVEPACK1 - 1D wavelet packet transform
WAVETRANS2 - 2D wavelet transform by tensor-product of two 1D transforms

In CWPROOT/src/comp/dct/libutil:
BUFFALLOC - routines to ALLOCate/initialize and free BUFFers
HUFFMAN - Routines for in memory Huffman coding/decoding
PENCODING - Routines to en/decode the quantized integers for lossless
QUANT - QUANTization routines
RLE - routines for in memory silence en/decoding

In CWPROOT/src/comp/dwpt/1d/lib:
WBUFFALLOC -  routines to allocate/initialize and free buffers in wavelet
WPC1 - routines for compress a single seismic trace using wavelet packets
WPC1CODING - routines for encoding the integer symbols in 1D WPC
wpc1Quant - quantize
WPC1TRANS - routines to perform a 1D wavelet packet transform

In CWPROOT/src/comp/dwpt/2d/lib:
WPC - routines for compress a 2D seismic section using wavelet packets
WPCBUFFAL - routines to allocate/initialize and free buffers
WPCCODING - Routines for in memory coding of the quantized coeffiecients
WPCENDEC -  Wavelet Packet Coding, Encoding and Decoding routines
WPCHUFF -Routines for in memory Huffman coding/decoding
WPCPACK2 - routine to perform a 2D wavelet packet transform
WPCQUANT - quantization routines for WPC
WPCSILENCE - routines for in memory silence en/decoding
Shells:
Libs:
Shells:
Libs:
Shells:
Libs:

To search on a program name fragment, type:
     "suname name_fragment <CR>"

For more information type: "program_name <CR>"
```

```
  Items labeled with an asterisk (*) are C programs that may
  or may not have a self documentation feature.

  Items labeled with a pound sign (#) are shell scripts that may,
  or may not have a self documentation feature.

 To find information about these codes type:   sudoc program_name
```

B.3 Suhelp.html

This is the text listing of Chris Liner's SU Help page.

```
                                 SeismicUn*x

                           Version 33 (5 April 1999)

                             An HTML Help Browser

   * This is a help browser for the SeismicUn*x free software package
     developed and maintained by the Center for Wave Phenomena at the
     Colorado School of Mines. The SU project is directed at CWP by John
     Stockwell.
   * The author of this help facility is Dr. Christopher Liner (an alumnus
     of CWP) who is a faculty member in the Department of Geosciences at The
     University of Tulsa.
   * Last updated January 16, 1998



        o The arrangement below is by funtionality
        o Clicking on a program name pulls up the selfdoc for that item
        o Your web browser's Find capability is useful if you have a
          fragment in mind (e.g. sort or nmo)
        o While programs may logically apply to more than one catagory
          below, each program appears only once
     -----------------------------------------------------------------------

  1. Functional Listing
       1. Data Compression
       2. Editing, Sorting and Manipulation
       3. Filtering, Transforms and Attributes
       4. Gain, NMO, Stack and Standard Processes
       5. Graphics
       6. Import/Export
       7. Migration and Dip Moveout
       8. Simulation and Model Building
       9. Utilities

    * Alphabetical name list             * 280 items
    * List is for scanning only, it       * For further info on an
      does not pull up the selfdoc of       interesting item use your
      a selected item.                      browser's find command, then
                                            follow the link

  -----------------------------------------------------------------------

Data Compression

 Discrete Cosine Transform
                                  * dctcomp             * dctuncomp

 Packing                          * supack1             * suunpack1
                                  * supack2             * suunpack2
                                  * wpc1comp2           * wpc1uncomp2
 Wavelet Transform                * wpccompress         * wpcuncompress
                                  * wptcomp             * wptuncomp
                                  * wtcomp              * wtuncomp

                                 Go to top

Editing, Sorting and Manipulation

                      * suabshw
                      * suazimuth               * suquantile
                                                * surange
 Edit + tools         * subset                  * sushw
                      * suchw
                      * suedit                  * sutab
                      * sugethw                 * suwind
                      * sukill                  * suxedit
 Sort                 * susort
                      * fcat                    * suramp
                      * maxdiff                 * surecip, recip
                      * suaddnoise              * suresamp, resamp
 Manipulate           * sudiff                  * suswapbytes
                      * suflip                  * sutaper
                      * suinterp                * suvcat
                      * sunull                  * suzero
                      * suop                    * swapbytes
                      * suop2                   * transp

                                 Go to top

Filtering, Transforms and Attributes

                                 * suband
                                                       * supef
 One-Dimensional filtering       * subfilt             * sushape
                                 * suconv
                                 * sufilter            * sutvband
                                 * sufrac              * suxcor
```

```
                                  * sudipfilt           * sumedian
 Two-Dimensional filtering        * sufxdecon           * sukfilter
                                  * suk1k2filter        * sukfrac

                                  * entropy
                                  * suamp               * sulog, suilog
                                  * suattributes        * suradon
                                                        * sureduce
 Transforms and Attributes        * suenv               * suspecfk
                                  * suhilb
                                  * sufft, suifft       * suspecfx
                                  * sugabor             * suspeck1k2
                                  * suharlan            * sutaup
                                  * sulog, suilog       * sutsq

                                  Go to top

Gain, NMO, Stack and Standard Processes

                               * suagc                    * sunmo
                                                          * supgc
 Standard Procs                * sudivcor                 * suresstat
                               * sugain
                               * sumix                    * sustack
                               * sumute                   * sustatic
                                                          * unglitch
                               * suacor
 Miscellaneous                 * suttoz
                               * suvibro
                               * suvlength
                               * dzdv
 Velocity Analysis             * sudivstack               * suvelan
                               * surelan                  * velpert

                                  Go to top

Graphics

 X-window ... for SU format
 input data                     * suxcontour
                                * suxgraph             * suxmovie
 (see binary form for full      * suximage             * suxpicker
 options)                       * suxmax               * suxwigb

 X-window ... for binary        * sxplot               * xmovie
 input data                     * xcontour             * xpicker
                                * xgraph
                                * ximage               * xwigb
 Postscript ... for SU
 format input data              * supscontour          * supsmovie (NeXT
                                * supscube               only)
```

(see binary form for full	* supsgraph	* supswigb
options)	* supsimage	* supswigp
	* psbbox	
Postscript ... for binary	* pscontour	* psmovie (NeXT only)
input data	* pscube	* pswigb
	* psgraph	* pswigp
	* psimage	* spsplot
	* fftlab	
	* h2b	* scmap
	* lcmap, lprop	* su3dchart
	* merge2, merge4	* suchart
Utilities + misc	* prplot	* supsmax
	* psepsi	* xrects
	* psmanager	* xepsb (NeXT only)
	* psmerge	* xepsp (NeXT only)
	* pslabel	* xepsp (NeXT only)

Go to top

Import/Export

	* a2b	
	* b2a	* segywrite
	* bhedtopar	* setbhed
	* dt1tosu	* suaddhead
		* suascii
Import/Export	* ftnstrip	* suget
	* ftnunstrip	
	* recast	* suoldtonew
	* segdread	* supaste
	* segyclean	* suput
	* segyhdrs	* sustrip
	* segyread	* z2xyz

Go to top

Migration and Dip Moveout

	* sugazmig	* sumigps
Poststack migration	* sumigfd	* sumigpspi
	* sumigffd	* sumigsplit
	* sumiggbzo	* sumigtk
Prestack/Poststack migration	* sukdmig2d	
	* sumigtopo2d	
	* sustolt	
Prestack migration	* sumigprefd	* sumigprepspi
	* sumigpreffd	* sumigpresp
Dip Moveout	* sudmofk	* sudmotx
	* sudmofkcw	* sudmovz

```
 Datuming                            * sudatumk2dr
                                     * sudatumk2ds

 Inversion (True amp migration)      * suinvvzco
                                     * suinvzco3d
```

Go to top

Simulation and Model Building

```
                              * gbbeam          * susyncz
                              * normray         * susynlv
                              * rayt2d          * susynlvcw
 Simulation (aka modeling)    * sufdmod2        * susynvxz
                              * suimp2d         * susynvxzcs
                              * suimp3d         * triray
                              * sukdsyn2d       * triseis
                              * sureflpsvsh     * wkbj
                              * makevel         * trimodel
                              * regrid3         * uni2tri
 Model Building               * suintvel        * unif2
                              * sustkvel        * unisam
                              * tetramod        * unisam2
                              * tri2uni         * velconv
                              * kaperture       * suspike
 Utilities                    * smooth2         * trip (Mesa 3D item)
                              * smooth3d        * viewer3 (Mesa 3D item)
                              * smoothint2      * vtlvz
                              * suplane         * xy2z
```

Go to top

Utilities

```
                     * copyright       * overwrite
                     * cpall           * pause               * time_now
                     * cpusec          * precedence          * todays_date
                     * cputime         * replace             * updatedoc
                     * ctrlstrip       * rmaxdiff            * updatedocall
                     * dirtree         * sumax               * updatehead
 Utilities           * downfort        * sumean              * upfort
                     * farith          * sumedian            * usernames
                     * filetype        * sunormalize         * varlist
                     * isatty          * supickamp           * wallsec
                     * lookpar         * sushift             * walltime
                     * maxints         * suweight            * weekday
                     * mkparfile       * t                   * zap
                     * newcase         * this_year
                     * cwpfind         * sugendocs
 Help utilities      * gendocs         * suhelp
                     * sudoc           * suname
                     * sufind          * sukeyword
```

Go to top

B.4 SUKEYWORD - list the SU datatype (SEGY) Keywords

The following is the listing put out by

```
typedef struct { /* segy - trace identification header */

int tracl; /* trace sequence number within line */

int tracr; /* trace sequence number within reel */

int fldr; /* field record number */

int tracf; /* trace number within field record */

int ep; /* energy source point number */

int cdp; /* CDP ensemble number */

int cdpt; /* trace number within CDP ensemble */

short trid; /* trace identification code:
1 = seismic data
2 = dead
3 = dummy
4 = time break
5 = uphole
6 = sweep
7 = timing
8 = water break
9---, N = optional use (N = 32,767)

Following are CWP id flags:

 9 = autocorrelation

10 = Fourier transformed - no packing
     xr[0],xi[0], ..., xr[N-1],xi[N-1]

11 = Fourier transformed - unpacked Nyquist
     xr[0],xi[0],...,xr[N/2],xi[N/2]

12 = Fourier transformed - packed Nyquist
       even N:
     xr[0],xr[N/2],xr[1],xi[1], ...,
xr[N/2 -1],xi[N/2 -1]
```

```
(note the exceptional second entry)
     odd N:
     xr[0],xr[(N-1)/2],xr[1],xi[1], ...,
xr[(N-1)/2 -1],xi[(N-1)/2 -1],xi[(N-1)/2]
(note the exceptional second & last entries)

13 = Complex signal in the time domain
     xr[0],xi[0], ..., xr[N-1],xi[N-1]

14 = Fourier transformed - amplitude/phase
     a[0],p[0], ..., a[N-1],p[N-1]

15 = Complex time signal - amplitude/phase
     a[0],p[0], ..., a[N-1],p[N-1]

16 = Real part of complex trace from 0 to Nyquist

17 = Imag part of complex trace from 0 to Nyquist

18 = Amplitude of complex trace from 0 to Nyquist

19 = Phase of complex trace from 0 to Nyquist

21 = Wavenumber time domain (k-t)

22 = Wavenumber frequency (k-omega)

23 = Envelope of the complex time trace

24 = Phase of the complex time trace

25 = Frequency of the complex time trace

30 = Depth-Range (z-x) traces

43 = Seismic Data, Vertical Component

44 = Seismic Data, Horizontal Component 1

45 = Seismic Data, Horizontal Component 2

46 = Seismic Data, Radial Component

47 = Seismic Data, Transverse Component

101 = Seismic data packed to bytes (by supack1)

102 = Seismic data packed to 2 bytes (by supack2)
*/

short nvs; /* number of vertically summed traces (see vscode
   in bhed structure) */
```

```
short nhs; /* number of horizontally summed traces (see vscode
   in bhed structure) */

short duse; /* data use:
1 = production
2 = test */

int offset; /* distance from source point to receiver
   group (negative if opposite to direction
   in which the line was shot) */

int gelev; /* receiver group elevation from sea level
   (above sea level is positive) */

int selev; /* source elevation from sea level
   (above sea level is positive) */

int sdepth; /* source depth (positive) */

int gdel; /* datum elevation at receiver group */

int sdel; /* datum elevation at source */

int swdep; /* water depth at source */

int gwdep; /* water depth at receiver group */

short scalel; /* scale factor for previous 7 entries
   with value plus or minus 10 to the
   power 0, 1, 2, 3, or 4 (if positive,
   multiply, if negative divide) */

short scalco; /* scale factor for next 4 entries
   with value plus or minus 10 to the
   power 0, 1, 2, 3, or 4 (if positive,
   multiply, if negative divide) */

int  sx; /* X source coordinate */

int  sy; /* Y source coordinate */

int  gx; /* X group coordinate */

int  gy; /* Y group coordinate */

short counit; /* coordinate units code:
for previous four entries
1 = length (meters or feet)
2 = seconds of arc (in this case, the
X values are longitude and the Y values
are latitude, a positive value designates
```

```
the number of seconds east of Greenwich
or north of the equator */

short wevel; /* weathering velocity */

short swevel; /* subweathering velocity */

short sut; /* uphole time at source */

short gut; /* uphole time at receiver group */

short sstat; /* source static correction */

short gstat; /* group static correction */

short tstat; /* total static applied */

short laga; /* lag time A, time in ms between end of 240-
   byte trace identification header and time
   break, positive if time break occurs after
   end of header, time break is defined as
   the initiation pulse which maybe recorded
   on an auxiliary trace or as otherwise
   specified by the recording system */

short lagb; /* lag time B, time in ms between the time break
   and the initiation time of the energy source,
   may be positive or negative */

short delrt; /* delay recording time, time in ms between
   initiation time of energy source and time
   when recording of data samples begins
   (for deep water work if recording does not
   start at zero time) */

short muts; /* mute time--start */

short mute; /* mute time--end */

unsigned short ns; /* number of samples in this trace */

unsigned short dt; /* sample interval; in micro-seconds */

short gain; /* gain type of field instruments code:
1 = fixed
2 = binary
3 = floating point
4 ---- N = optional use */

short igc; /* instrument gain constant */

short igi; /* instrument early or initial gain */
short corr; /* correlated:
1 = no
2 = yes */

short sfs; /* sweep frequency at start */

short sfe; /* sweep frequency at end */

short slen; /* sweep length in ms */

short styp; /* sweep type code:
1 = linear
2 = cos-squared
3 = other */

short stas; /* sweep trace length at start in ms */

short stae; /* sweep trace length at end in ms */

short tatyp; /* taper type: 1=linear, 2=cos^2, 3=other */

short afilf; /* alias filter frequency if used */

short afils; /* alias filter slope */

short nofilf; /* notch filter frequency if used */

short nofils; /* notch filter slope */

short lcf; /* low cut frequency if used */

short hcf; /* high cut frequncy if used */

short lcs; /* low cut slope */

short hcs; /* high cut slope */

short year; /* year data recorded */

short day; /* day of year */

short hour; /* hour of day (24 hour clock) */

short minute; /* minute of hour */

short sec; /* second of minute */

short timbas; /* time basis code:
1 = local
2 = GMT
3 = other */
```

```
short trwf; /* trace weighting factor, defined as 1/2^N
   volts for the least sigificant bit */

short grnors; /* geophone group number of roll switch
   position one */

short grnofr; /* geophone group number of trace one within
   original field record */

short grnlof; /* geophone group number of last trace within
   original field record */

short gaps; /* gap size (total number of groups dropped) */

short otrav; /* overtravel taper code:
1 = down (or behind)
2 = up (or ahead) */

/* local assignments */
float d1; /* sample spacing for non-seismic data */

float f1; /* first sample location for non-seismic data */

float d2; /* sample spacing between traces */

float f2; /* first trace location */

float ungpow; /* negative of power used for dynamic
   range compression */

float unscale; /* reciprocal of scaling factor to normalize
   range */

int ntr;  /* number of traces */

short mark; /* mark selected traces */

        short shortpad; /* alignment padding */


short unass[14]; /* unassigned--NOTE: last entry causes
   a break in the word alignment, if we REALLY
   want to maintain 240 bytes, the following
   entry should be an odd number of short/UINT2
   OR do the insertion above the "mark" keyword
   entry */

float  data[SU_NFLTS];

} segy;
```

```
typedef struct { /* bhed - binary header */

int jobid; /* job identification number */

int lino; /* line number (only one line per reel) */

int reno; /* reel number */

short ntrpr; /* number of data traces per record */

        short nart; /* number of auxiliary traces per record */

unsigned short hdt; /* sample interval in micro secs for this reel */

unsigned short dto; /* same for original field recording */

unsigned short hns; /* number of samples per trace for this reel */

unsigned short nso; /* same for original field recording */

short format; /* data sample format code:
1 = floating point (4 bytes)
2 = fixed point (4 bytes)
3 = fixed point (2 bytes)
4 = fixed point w/gain code (4 bytes) */

short fold; /* CDP fold expected per CDP ensemble */

short tsort; /* trace sorting code:
1 = as recorded (no sorting)
2 = CDP ensemble
3 = single fold continuous profile
4 = horizontally stacked */

short vscode; /* vertical sum code:
1 = no sum
2 = two sum ...
N = N sum (N = 32,767) */

short hsfs; /* sweep frequency at start */

short hsfe; /* sweep frequency at end */

short hslen; /* sweep length (ms) */

short hstyp; /* sweep type code:
1 = linear
2 = parabolic
3 = exponential
4 = other */
```

```
short schn; /* trace number of sweep channel */

short hstas; /* sweep trace taper length at start if
   tapered (the taper starts at zero time
   and is effective for this length) */

short hstae; /* sweep trace taper length at end (the ending
   taper starts at sweep length minus the taper
   length at end) */

short htatyp; /* sweep trace taper type code:
1 = linear
2 = cos-squared
3 = other */

short hcorr; /* correlated data traces code:
1 = no
2 = yes */

short bgrcv; /* binary gain recovered code:
1 = yes
2 = no */

short rcvm; /* amplitude recovery method code:
1 = none
2 = spherical divergence
3 = AGC
4 = other */

short mfeet; /* measurement system code:
1 = meters
2 = feet */

short polyt; /* impulse signal polarity code:
1 = increase in pressure or upward
    geophone case movement gives
    negative number on tape
2 = increase in pressure or upward
    geophone case movement gives
    positive number on tape */

short vpol; /* vibratory polarity code:
code seismic signal lags pilot by
1 337.5 to  22.5 degrees
2  22.5 to  67.5 degrees
3  67.5 to 112.5 degrees
4 112.5 to 157.5 degrees
5 157.5 to 202.5 degrees
6 202.5 to 247.5 degrees
7 247.5 to 292.5 degrees
8 293.5 to 337.5 degrees */
```

```
short hunass[170]; /* unassigned */

} bhed;

/* DEFINES */
#define gettr(x) fgettr(stdin, (x))
#define vgettr(x) fvgettr(stdin, (x))
#define puttr(x) fputtr(stdout, (x))
#define gettra(x, y)    fgettra(stdin, (x), (y))

/* The following refer to the trid field in segy.h */
/* CHARPACK represents byte packed seismic data from supack1 */
#define CHARPACK 101
/* SHORTPACK represents 2 byte packed seismic data from supack2 */
#define SHORTPACK 102

/* TREAL represents real time traces  */
#define TREAL 1
/* TDEAD represents dead time traces  */
#define TDEAD 2
/* TDUMMY represents dummy time traces  */
#define TDUMMY 3
/* TBREAK represents time break traces  */
#define TBREAK 4
/* UPHOLE represents uphole traces  */
#define UPHOLE 5
/* SWEEP represents sweep traces  */
#define SWEEP 6
/* TIMING represents timing traces  */
#define TIMING 7
/* WBREAK represents timing traces  */
#define WBREAK 8

/* TCMPLX represents complex time traces  */
#define TCMPLX 13
/* TAMPH represents time domain data in amplitude/phase form */
#define TAMPH 15
/* FPACK represents packed frequency domain data  */
#define FPACK 12
/* FUNPACKNYQ represents complex frequency domain data  */
#define FUNPACKNYQ 11
/* FCMPLX represents complex frequency domain data  */
#define FCMPLX 10
/* FAMPH represents freq domain data in amplitude/phase form */
#define FAMPH 14
/* REALPART represents the real part of a trace to Nyquist */
#define REALPART 16
/* IMAGPART represents the real part of a trace to Nyquist */
#define IMAGPART 17
/* AMPLITUDE represents the amplitude of a trace to Nyquist */
#define AMPLITUDE 18
/* PHASE represents the phase of a trace to Nyquist */
```

```
#define PHASE 19
/* KT represents wavenumber-time domain data  */
#define KT 21
/* KOMEGA represents wavenumber-frequency domain data */
#define KOMEGA 22
/* ENVELOPE represents the envelope of the complex time trace */
#define ENVELOPE 23
/* INSTPHASE represents the phase of the complex time trace */
#define INSTPHASE 24
/* INSTFREQ represents the frequency of the complex time trace */
#define INSTFREQ 25
/* DEPTH represents traces in depth-range (z-x) */
#define TRID_DEPTH 30
/* 3C data...  v,h1,h2=(11,12,13)+32 so a bitmask will convert  */
/* between conventions */
/* TVERT represents the vertical component */
#define     TVERT          43
/* TVHOZ1 represents the horizontal-1 component */
#define     THORZ1         44
/* TVHOZ2 represents the horizontal-2 component */
#define     THORZ2         45
/* TRADIAL represents the radial component */
#define     TRADIAL        46
/* TTRANS represents the transverse component */
#define     TTRANS         47

/* #define ISSEISMIC(id) (( (id)==0 || (id)==TREAL || (id)==TDEAD || (id)==TDUMMY ) ? cwp_true : cw
#define ISSEISMIC(id) (( (id)==0 || (id)==TREAL || (id)==TDEAD || (id)==TDUMMY || \
 (id)==TVERT || (id)==THORZ1 || (id)==THORZ2 || \
 (id)==TRADIAL || (id)==TTRANS  ) ? cwp_true : cwp_false )

/* FUNCTION PROTOTYPES */
#ifdef __cplusplus /* if C++, specify external linkage to C functions */
extern "C" {
#endif

int fgettr(FILE *fp, segy *tp);
int fvgettr(FILE *fp, segy *tp);
void fputtr(FILE *fp, segy *tp);
int fgettra(FILE *fp, segy *tp, int itr);

/* hdrpkge */
void gethval(const segy *tp, int index, Value *valp);
void puthval(segy *tp, int index, Value *valp);
void getbhval(const bhed *bhp, int index, Value *valp);
void putbhval(bhed *bhp, int index, Value *valp);
void gethdval(const segy *tp, char *key, Value *valp);
void puthdval(segy *tp, char *key, Value *valp);
char *hdtype(const char *key);
char *getkey(const int index);
int getindex(const char *key);
void swaphval(segy *tp, int index);
void swapbhval(bhed *bhp, int index);
void printheader(const segy *tp);

void tabplot(segy *tp, int itmin, int itmax);

#ifdef __cplusplus /* if C++, end external linkage specification */
}
#endif

#endif
```

Appendix C

DEMOS - a brief descriptions of the demos

The directiory $CWPROOT/src/demos contains a number of subdirectories containing ready to run shell scripts to demostrate SU programs. This collection is nowhere as complete as it should be, and is continually changing and being updated.

- BC_Examples
- Block: Demo for the program "block" in the free package UNCERT
- 3D_Model_Building:
  - Tetramod: Demo for "tetra," the seismic wavespeed model building code.
- Datuming: Examples of datuming
  - Finite_Difference: Traditional Finite difference Datuming
  - Kirchhoff_Datuming: Kirchhoff datuming
    - Migration_with_topography: Migration with topographics effects
- Deconvolution: Deconvolution demos
  - Wiener_Levinson: prediction error filtering
  - FX: frequency-wavenumber
- Dip_Moveout: Dip Moveout
  - Ti: Dip moveout in transversely isotropic media
- Diversity_Stacking:
- Filtering: Filtering demos
  - Subfilt: Butterworth filter
  - Sudipfilt: Dip (slope) filtering
  - Sufilter: Polygonal zero-phase frequency filtering
  - Sugabor: time-frequency filtering via the Gabor transform
  - Suk1k2filter: Wavenumber filtering (box)
  - Sukfilter: Wavenumber filtering (annular)
  - Sumedian: Median filtering

- Making_Data: Making synthetic seismic profiles
  - CommonOffset: Common-offset and Zero-offset datasets
  - ShotGathers: Common-shot datasets
- Migration_Inversion: Migration (Inversion)
- 3D Migration or inversion of 3D datasets
  - Suinvco3d: Common offset migration in v(z) media
- Finite_Difference: traditional Claerbout FD migration
- Fourier_Finite_Difference: Fourier FD migration
- Gazdag: traditional Gazdag phaseshift migration
- Kirchhoff: Kirchhoff migration
  - Sukdmig2d_common_offset: Common-offset migration
  - Sukdmig2d_common_shot: Common-shot migration
- Phase_Shift_Plus_Interpolation: PSPI migration
- Split_Step: Split step migration
- Stolt: Stolt migration
- Suinvvxzco: Inversion in v(x,z) media for common-offset data sets
- Sumigpsti: Phaseshift Migration in Transversely Isotropic media
- NMO: Traditional Normal Moveout correction
- Offset_Continuation: Offset continuation to smaller offsets
- Picking: Trace Picking
  - Supickamp: Automated picking
- Plotting: Plotting demos
  - CurveOnImage: Draw a curve on an image plot
- Refraction: Refraction applications
  - GRM: Generalized Reciprocal Method for a single layer
- Residual_Moveout: Velocity analysis by residual moveout analysis
- SUManual: Shell scripts used to make plots in the SU User's Manual
  - Plotting: Plotting demos
    - Psmerge: Merge PostScript files with **psmerge**
    - Xmovie: Make X-windows movies with **suxmovie**
- Selecting_Traces: Selecting traces by header values for specific operations
- Sorting_Traces: Sorting traces with **susort**
  - Demo: Sorting traces demo
  - Tutorial: Sorting traces tutorial
- Statics: Static time shift corrections

  - Residual_Refraction_Statics: applying residual refraction static corrections
  - Residual_Statics: residual source-receiver statics
- Synthetic: Making synthetic seismograms
  - Finite_Difference: finite difference 2D modeling
  - Impulse: impulse testpatterns
  - Kirchhoff: Kirchhoff modeling
  - Reflectivity: Reflectivity modeling
  - Tetra: Tetrahedrized models
  - Tri: Triangulated models
  - Wkbj: Upwind finite differencing of the eikonal equation
  - CSHOT: see the demos in $CWPROOT/src/Fortran/CSHOT
- Tau_P: Tau-p or slant stack filtering
  - Suharlan: Bill Harlan's slant stack noise suppression algorithm
  - Sutaup: Traditional Tau-p transform
- Time_Freq_Analysis: operations in the time-frequency domain
- Utilities: Miscellaneous (and experimental) utilities
  - Sucommand: Demo for program **sucommand**
- Velocity_Analysis: NMO-based velocity analysis demo
- Velocity_Profiles: Make velocity profiles
  - Makevel: make velocity profiles with **makevel**
  - Unif2: uniformly sampled 2D velocity profiles with **unif2**
  - Unisam2: uniformly sampled 2D velocity profiles with **unisam2**
  - Vel2stiff: convert velocity profiles to stiffness profiles **vel2stiff**
- Vibroseis_Sweeps: Synthetic vibroseis sweeps with **suvibro**

The beginner is encouraged to run the demos in the following order:

The Making_Data demo shows the basics of making synthetic data shot gathers and common offset sections using susynlv. Particular attention is paid to illustrating good display labeling.

The Filtering/Sufilter demo illustrates some real data processing to eliminate ground roll and first arrivals. The demos in the Filtering subdirectories give instructions for accessing the data from the CWP ftp site.

The Deconvolution demo uses simple synthetic spike traces to illustrate both dereverberation and spiking decon using supef and other tools. The demos include commands for systematically examining the effect of the filter parameters using loops.

The Sorting_Traces Tutorial is an interactive script that reinforces some of the basic UNIX and SU lore discussed in this document. The interactivity is limited to allowing you to set the pace. Such tutorials quickly get annoying, but we felt that one such was needed to cover some issues that didn't fit into our standard demo format. There is also a standard, but less complete, demo on this topic.

The next step is to activate the Selecting_Traces Demo. Then proceed to the NMO Demo. Beyond that, visit the Demo directories that interest you. The **demos** directory tree is still under active development—please let us know if the demos are helpful and how they can be improved.

As the user becomes more familiar with the logic of SU, then he or she may feel free to run any of the other demos either in their original form, or the user may modify the shell scripts to his or her own ends.