A tour of SEPlib for new users

Joe Dellinger, Sándor Tálas and James Rickett [1]

/usr/local/src/our/sephelp/septour

ABSTRACT

SEPlib is a data-processing software package developed at the Stanford Exploration Project. Many researchers at universities and oil companies would find SEPlib extremely useful if they could only get over the initial hump of learning how SEPlib programs are used. By working through several illustrative examples this document attempts to painlessly introduce novice users to the fundamental SEPlib concepts they will need to understand the available SEPlib self-documentation and manual pages. Concepts covered include self-documenting programs, history files and data files, SEPlib-style pipes, command-line and history-file "getpar" parameters, auxiliary input and output, and how to use some of the more useful SEPlib utilities.

INTRODUCTION

Researchers spend too much time hacking together personal special-purpose programs to do their work, usually reinventing "square wheels" again and again in the process. By sharing programs in a common framework researchers can spend their programming time writing only those special-purpose programs they need for their particular work.

The SEPlib software package evolved at the Stanford Exploration Project over the last decade as a partial solution to this problem. SEPlib is organized like UNIX; most programs perform relatively basic operations. Complex manipulations are performed by stringing these simple programs together via pipes and files. If you like this philosophy of programming you will probably find SEPlib easy and natural to use. Advanced users fluent in FORTRAN or C can quickly write their own SEPlib utilities; if well written, these can be added to the pool of common programs available for all to use.

[1] **email:** james@sep.stanford.edu

While documentation does exist for SEPlib, it is minimal and mostly intended for users already familiar with UNIX, FORTRAN, and C. Traditionally, beginning SEPlib users have learned the basics directly from more experienced users, and only later with experience are they able to use the software-locating utilities and documentation that exists to discover and use new programs on their own. You may not have the opportunity to learn SEPlib this way, so we have prepared this document to help get you started.

Accessing SEPlib depends on your site, so we have relegated this information to an Appendix. Assuming you can type the name of a SEPlib program and have it run and have some sort of graphical output device available, this document will take you on a "guided tour" through several of the most important SEPlib programs and utilities. On the way, if you pay close attention, you should be able to pick up the basic knowledge you will need to proceed further on your own. There will be a brief summary of the important points at the end.

After you have finished reading and trying out the examples in this document, try reading Claerbout (1990), and at a more advanced level Claerbout (1986) to learn more.

GETTING GOING

On-line version

If you are already reading this document on a computer you will probably have no problems running SEPlib programs. Let's see whether it works or not.

Click on this box with the middle button of the mouse: **test**

Did it work? Great!

If you want to try the examples as you read along (and we strongly encourage you to try it) open a window for yourself or simply click on this box: **window**

Paper version

If you are reading the paper version of this document and you think that your computer is set up to run SEPlib, test it by carefully typing:

```
Spike n1=1000 k1=300 title="\s200 Welcome to \c2 SEP\c5 lib" \
 | Bandpass fhi=2. phase=1 | Wiggle | Tube
```

This might look pretty frightening at a first glance, but if it works you are evidently able to run SEPlib. If it did not work, go look at the Appendix on "Basic troubleshooting" before continuing.

General SEPlib features

Although SEPlib currently consists of over 100 different programs, they share many common features. First of all, by convention SEPlib programs start with a Capital Letter. More importantly, most SEPlib programs are "filters"; they read from standard input and write to standard output:

```
Prog < input   > output
```

Complex functions are created by joining filters with pipes:

```
Prog1 < in | Prog2 | Prog3 > out
```

We will use the SEPlib program "`Wiggle`" to demonstrate some basic SEPlib-program properties. "`Wiggle`" is just a simple program that converts raw data into wiggle traces, but it is used a lot because people are usually curious to see what they have done to their data.

Try typing:

```
Wiggle
```

You should get a couple of screens' worth of documentation. This is called "self documentation": run the program without arguments, input, or output, and it will display a brief documentation summary. Almost all SEPlib programs will self document, which is good because very few of them have real manual pages.

If you get an error message `doc(): No such file or directory`, complain to the person who installed SEPlib at your site! If you don't feel like complaining (perhaps because you are that person) and you know where the SEPlib source is installed, you can tell SEPlib programs an alternate place to look for their source by setting the environment variable `SEP_DOC_PATH`. See the "Basic troubleshooting" Appendix.

Getting the test data

Hopefully you have now worked out any software-environment problems you might have had before, and you are ready for your "test drive".

There should be three data files in the directory where you found this paper. `Txx.HH`, `Txy.HH` and `Txz.HH`. Plot one of these files by doing

```
Wiggle < Txx.HH | Tube
```

When you run the command above `Wiggle` creates a plot which `Tube` then tries to display on your screen. Did it work? Hopefully your screen will look something like the one in Figure 1. (If when you try it "`Tube`" does the plot using `Xtpen`, like in the figure, exit the program by clicking on the `QUIT` button at the top of the window. If

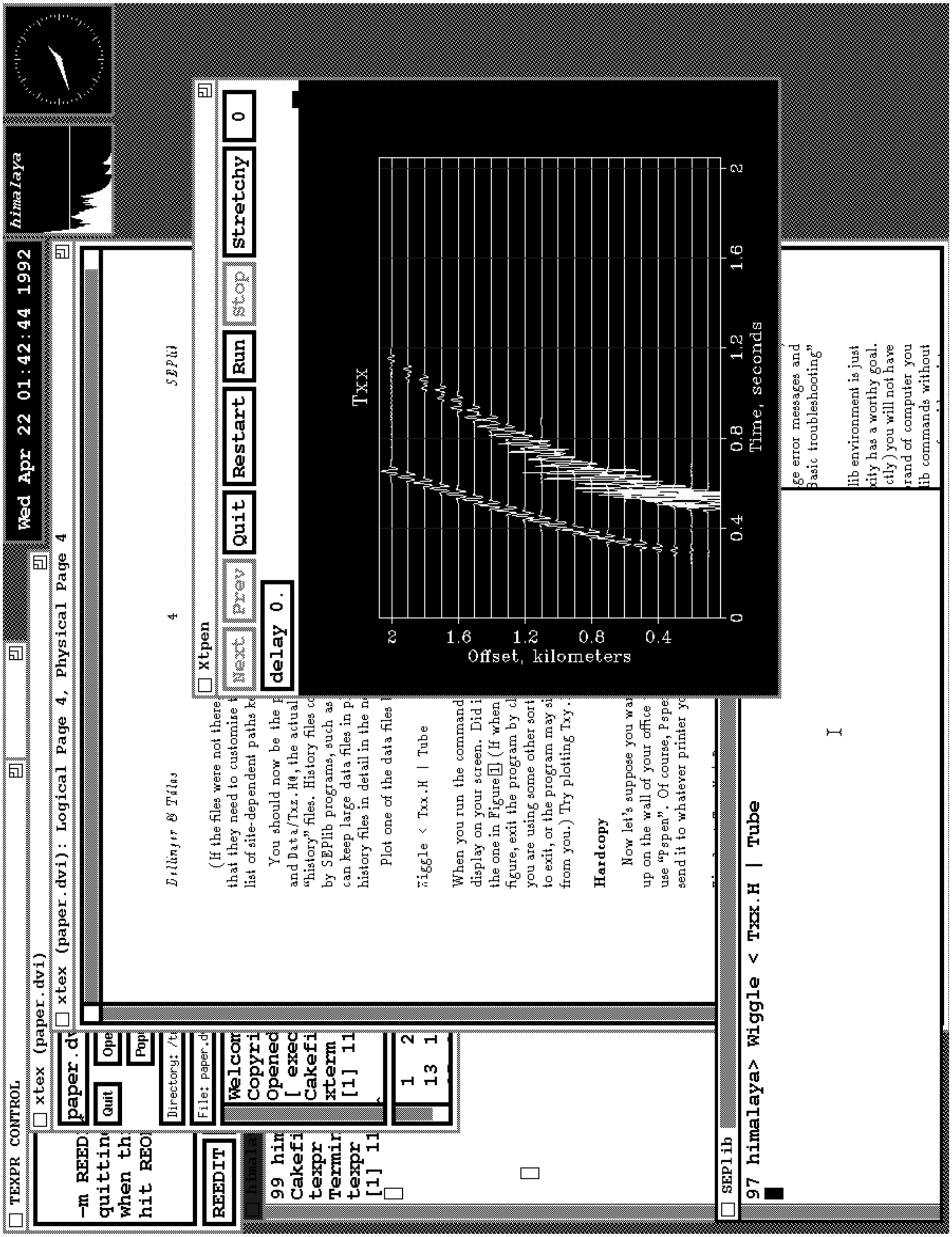

Figure 1: `Wiggle < Txx.HH | Tube` Screen [NR]

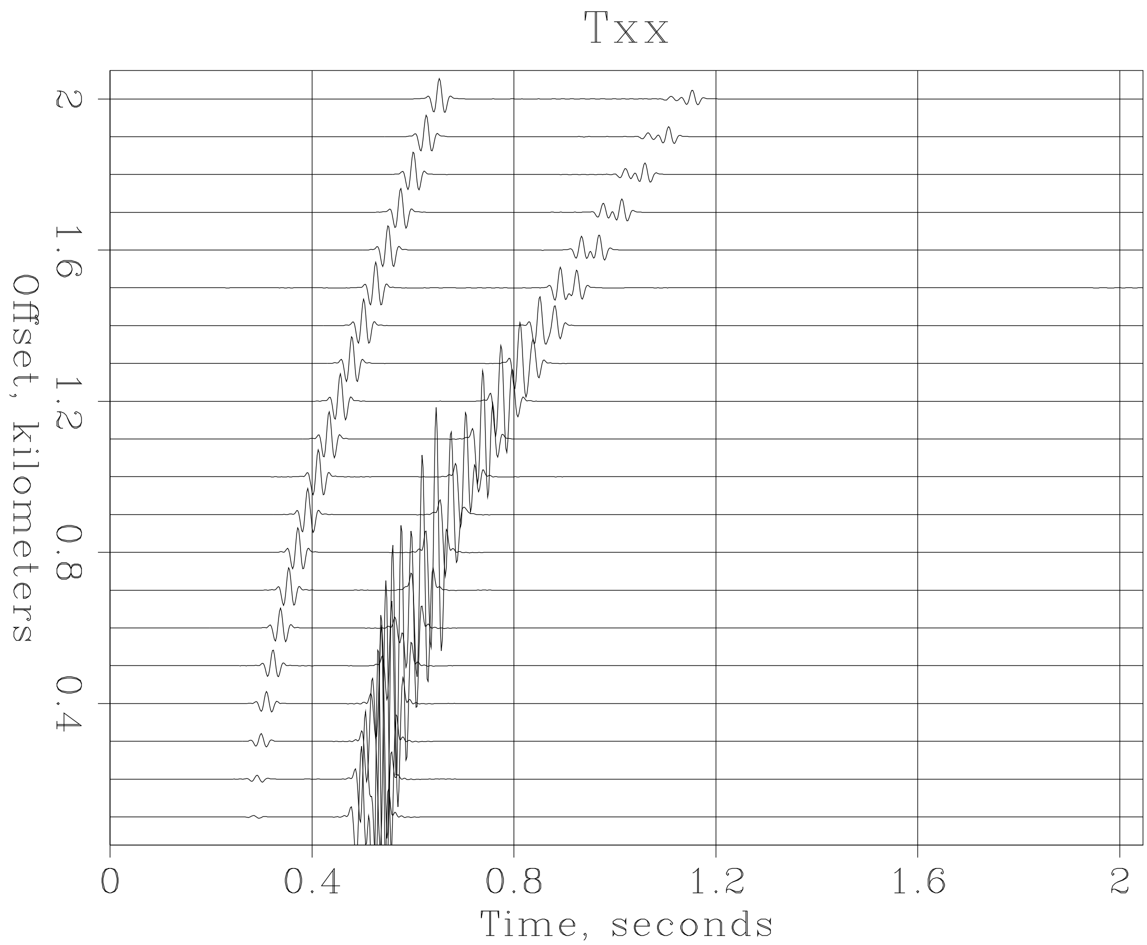

Figure 2: `Wiggle < Txx.HH | Pspen` Wiggle1 [ER]

you are using some other sort of graphics device you may have to hit return or space to exit, or the program may simply exit when the plot is done without any prompting from you.) Try plotting `Txy.H` and `Txz.H` too.

Hardcopy

Now let's suppose you want a hardcopy plot instead, something that you can put up on the wall of your office to demonstrate your efforts. Simple: instead of "`Tube`" use "`Pspen`". Of course, `Pspen` has to know where to send the plot. By default it will send it to whatever printer your local machine thinks is called "postscript". Try:

```
Wiggle < Txx.HH | Pspen
```

Hopefully this will work, producing something like Figure 2.

If the printer your workstation calls "postscript" turns out to be old and slow, on another floor or in another building, or you often get strange error messages and partial plots when you attempt to plot big files, check the "Basic troubleshooting" Appendix for some advice.

At this point you may be thinking that setting up your SEPlib environment is just too tedious to be worth it. Don't despair; the apparent complexity has a worthy goal. The idea of SEPlib and "`Tube`" is that (if things are set up correctly) you will not have to worry about what device you are sitting at or even what brand of computer you are logged into. You should be able to just use the same SEPlib commands without worry on any of your local computers that run UNIX, and on any of a wide variety of graphics devices, and always get the same behavior and the same plots on your screen (or wall).

History files

Take a look at the format of the data by typing `more Txx.HH`. You will see that the first part of the file is ASCII text, and the second part of the file comes out as nonsense because it is binary data. These two parts are quite distinct, and, as we shall see later, are often stored in separate files. For this reason they have different names: the text part of the file is called the "history file", and the binary part of the file is called the "data file".

The ASCII parts of the data are called "history" files because they document the "history" of the corresponding data. (They are also called "header" files in some documents.) Programs *append* information onto the history file; it thus contains a history of the programs run (and often of all the parameters specified, so that from the history file you could recreate the file from scratch).

In our example so far three programs have been run. The line

```
Mallick:   joe@montebello   Wed Feb 19 00:27:17 1992
```

shows that this particular data originated in its SEPlib incarnation from a program called "Mallick", which was run by a "joe@montebello" in the early hours of a Wednesday morning in 1992. The lines

```
Mv:   matt@oas   Sat Sep 16 03:31:53 1995
```

and

```
Cp:   matt@oas   Sat Sep 16 03:41:38 1995
```

show that the data was then moved and copied by Matt to where it is now. Notice that more recent additions are added to the bottom.

The most important part of the history file is the line

```
in="stdin"
```

The parameter `in` tells where the data associated with the file `Txx.HH` actually is. In our case the data is attached to its history file, and so its location is described as the "standard input". However, if the data file is separated from the history file, this will be a pathname showing where the data described by the history file can be found. Either relative pathnames (begins with "`./`"), or absolute pathnames (begins with "`/`") can be used.

The rest of the file gives other important information:
`esize=4` indicates the data consists of elements 4 bytes long;
`data_format="xdr_float"` indicates these 4-byte long elements are in fact machine-independent IEEE floating-point numbers.
`n1=1024 n2=20` indicates the data is in a 2-dimensional array with a fast axis `1024` elements long and a slow axis `20` elements long.
`o1=0 d1=.002 label1="Time, seconds"` indicates the fast axis has dimensions of seconds, with the first element corresponding to time `0`, the second element time `.002`, the third `.004`, etc.
`o2=.1 d2=.1 label2="Offset, kilometers"` indicates the slow axis has dimensions of kilometers, with the first element corresponding to an offset of `.1` kilometers, the second element `.2`, the third `.3`, etc.

So far we have displayed our data without creating any new files at all. We did this by using UNIX pipelines ("`|`"). These tell the operating system we want to pass the information (both data and parameters) from one program to another, and we don't want to be bothered with having to keep track of any temporary intermediate files ourselves.

We could have done it differently by saving the output of the programs. Let's see how our favorite example

```
Wiggle < Txx.HH | Tube
```

can be split up into two separate steps[2]:

```
Wiggle < Txx.HH  > Out.H
Tube < Out.H
```

where `Out.H` is the output of `Wiggle`.[3] (Do "man vplot" if you are curious what sort of output `Wiggle` writes.)

Take a look at the the output file with `more Out.H`. This time you will only see the ASCII history file. So the question is: where has the actual data gone? This might be interesting if you are playing around with files of a size of several 100MBytes, given that we all have to face the fact that free diskspace is always limited.

Unless otherwise instructed, SEPlib will attempt to separate your data files from your history files. The advantage of this is that you may want to keep large, bulky data files somewhere away from your home directory, where you do most of your work.

[2]the fact that the original data has the suffix `HH` and the output only has a single `H` is nothing to do with SEPlib - it just stops the original data from being deleted when the directory is cleaned, with `gmake clean` for example

[3]If you just got an error message don't panic yet, just keep reading.

There are four options for directing your output:

By default SEPlib will attempt to put the raw data in a subdirectory with your username under the system-default SEPlib "scratch" directory. If you tried the commands above and got an error message something like

```
output(): No such file or directory
Wiggle: output() unable to open output file /sepcd5/scr/joe/_Out.H@
```

It means your default directory did not exist. Look to see what directory `Wiggle` was trying to use, and create it if you wish (and have sufficient permissions to do so) using the UNIX command "`mkdir`"; then try our example again. Be warned that to keep such "public" data areas from filling with junk, they are usually subject to swift and merciless disk mowing.

Alternatively, you can specify where you want the data files to go with a command-line parameter, for example:

```
Wiggle < Txx.HH  > Out.H out=./Data/Out.datafile
```

but it might get tiring to do this every single time! (For a quick experiment you might want to try the above example with and without the leading "`./`" in the `out=` argument, and note what `in=` gets set to in `Out.H` in each case. Unless the SEPlib output subroutine sees a leading "`./`", it automatically expands output file names to fully qualified paths.)

Another, better, solution is for you to create a personal directory to keep your data in somewhere, and tell SEPlib that's where you want it to put your data by default. Let's suppose you create a directory called "`Data`" under your home directory. You then tell SEPlib to put data files there by doing:

```
setenv DATAPATH ~/Data/
```

(Note in this example the leading ~ will get expanded to your full home directory name by the `csh` *before* the `DATAPATH` variable is set.) **Remember that binary data files will accumulate in the directory given by your `DATAPATH` if you are not careful. Make sure to use `Rm`, not `rm`, to delete SEPlib files!** (Examples of using `Rm` to remove SEPlib files can be found later in this document.)

Note that the `DATAPATH` is simply prepended as an arbitrary string to a slightly modified version of the history filename to get a name for the data file, so you probably want your `DATAPATH` to end with a "/", like the example above does. You can also set your datapath by creating a file called "`.datapath`" in your home directory (or in your current directory, with the one in your current directory taking precedence). The `.datapath` file should contain a line looking something like

```
datapath=/home/montebello/joe/Data/
```

Note in the file you have to expand out your full home-directory name yourself.

Finally, if you want the data to stay attached to the history file you can use the `out=` command line option again but this time send the data to the standard output along with the history file:

```
Wiggle < Txx.HH  > Out.H out="stdout"
```

The SEPlib datacube

If we have data frames of the same size (like shot records usually are) we can easily merge them into a "datacube" to make their processing easier. Let's try to merge the three files Txx.HH Txy.HH and Txz.HH into a datacube using the program "`Cat`", which does to SEPlib files something like what "`cat`" does to ASCII files.

```
Cat Txx.HH Txy.HH Txz.HH > Three.H
```

Now make a wiggle plot of this new file by doing:

```
Wiggle < Three.H | Tube
```

Pretty snazzy, eh? Tube realizes that there are three different panels, so it shows all three of them.[4]

Look at the history file `Three.H` to see how history "accumulated" in this example. In general, each successive SEPlib program writes more information onto the end of the history file. (`Cat` is a bit of a special case, since it is always called with multiple input files and doesn't use redirected input. Most SEPlib programs read a file from standard input and write a file to standard output. The SEPlib input and output subroutines shared by all such "standard" programs begin by automatically copying the input header straight across to the output unchanged. Anything the running program wants to add to the history then gets appended. `Cat` has to do the copying "by hand", so its output looks a little different.)

Note that lines setting parameters such as "`n3=1`" can occur multiple times in one history file, as various programs set old parameters to new values. The last-defined value is the only valid one, because it is the "most recent" and corresponds to the current data.

You might be thinking now that using "`more`" to examine history files to find the dimensions of the associated data file can get confusing and tedious if the history is long and complex. A quick way to examine the dimensions and properties of a SEPlib file is to use the command "`In`":

[4]Hardcopy devices will print out three separate pages of plots. On devices where you can see either the text screen or the graphics screen (but not both at the same time) you may have to hit "Return" to work through the sequence of plots one at a time. On most other screen devices the three plots will simply zip by in quick succession; you have to give an option to tell it to slow down or wait for a keypress. If your `pen` program is `Xtpen` it will animate the three frames for you, Movie-style.

```
In Three.H
```

gives the salient features of the dataset

```
Three.H:
        in="/usr/local/sep/scr/joe/Three.H@"
        expands to in="/usr/local/sep/scr/joe/Three.H@"
        esize=4
        n1=1024 n2=20   n3=3            61440 elem      245760 bytes
        d1=.002 d2=.1   d3=1
        o1=0    o2=.1   o3=0
        label1=Time, seconds
        label2=Offset, kilometers
```

Note that "`Three.H`" is a three-dimensional block of data, with `n3=3`.

When you are done with `Three.H`, delete it by doing

```
Rm Three.H
```

This will delete both the history file and its associated binary data file. If you slip and accidentally use `rm` instead, the binary data file will remain behind uselessly cluttering up your data directory, possibly forever if nothing ever looks for junk files to clean up there.

Warning: the default behavior of both `rm` and `Rm` is to **go ahead and delete without confirmation**. You probably have `rm` aliased to `rm -i` (you may have forgotten doing it), but you probably *don't* have `Rm` aliased to `Rm -i`.

ILLUSTRATIVE EXAMPLES

Playing with parameters

SEPlib programs generally do simple things. They are still very flexible, though, because their default behavior can be modified by appropriate command-line or history-file parameters. Most programs have at least a few options or parameters; some of them have hundreds. Let's look at some relatively simple examples.

We saw already that most of the Txx.HH data consists of zero values that are not very interesting for us. Let's trim the data a bit. The SEPlib utility `Window` is used for this purpose. If you remember, `Txx.HH` consists of 1 plane of 20 traces, with 1024 samples in each trace.[5] In SEPlib notation, `n3=1 n2=20 n1=1024`. Let's "zoom in" on the interesting part of the data between times `.4` and `.8` seconds and offsets from the smallest (`.1`) up to `1.` (i.e., we will keep samples 200 through 400 of the first 10 traces). To do its job `Window` needs to know the number of samples and the

[5] If not, you can always check by running `In Txx.HH`!

first sample to accept for each axis. (The defaults are "all of them that you can" and "begin at the beginning", respectively.) For our example we would have:

```
Window < Txx.HH n2=10 n1=200 f1=200 > Txx_Windowed.H
```

Fortunately `Window` is smart enough to understand your command using the values on the two axes too:

```
Window < Txx.HH min1=.4 max1=.8 max2=1. > Txx_Windowed.H
```

We must warn you that the second method is more risky, since it is possible that you have an error in the sampling parameters such as `o1` and `d1` in the history file, or you simply forgot to specify them at all (so they defaulted to `0` and `1` respectively). You may prefer to tell `Window` where to start and end using integer sample numbers. If you are sure that you would never make such mistakes, did you catch the discrepancy in the two examples above? You'll find they don't give quite the same results! Starting at `.4` and ending at `.8` is 201 samples, not 200. As usual for computer programs, `Window` does what you tell it to do, not necessarily what you mean for it to do.

Whichever way you did it,

```
In Txx_Windowed.H
```

shows us the file `Txx_Windowed.H` is much reduced in size:

```
Txx_Windowed.H:
in="/usr/local/sep/scr/joe/_Txx_Windowed.H@"
        expands to in="/usr/local/sep/scr/joe/_Txx_Windowed.H@"
        esize=4
        n1=200  n2=10   n3=1    n4=1            2010 elem       8040 bytes
        d1=.002 d2=.1   d3=1    d4=1
        o1=.4   o2=.1   o3=0    o4=0
        label1=Time, seconds
        label2=Offset, kilometers
```

Note in particular the new value for `o1`.

Now let's have a look at what we have done:

```
Wiggle < Txx_Windowed.H | Tube
```

So far so good, but let's suppose that the journal you are submitting to insists wiggle plots must have traces that run down instead of across, the order of the traces must go the other way, and the traces must have geophysical-style shading. No problem; from `Wiggle`'s self documentation you find there are three parameters that are likely to do what you need. Try them out:

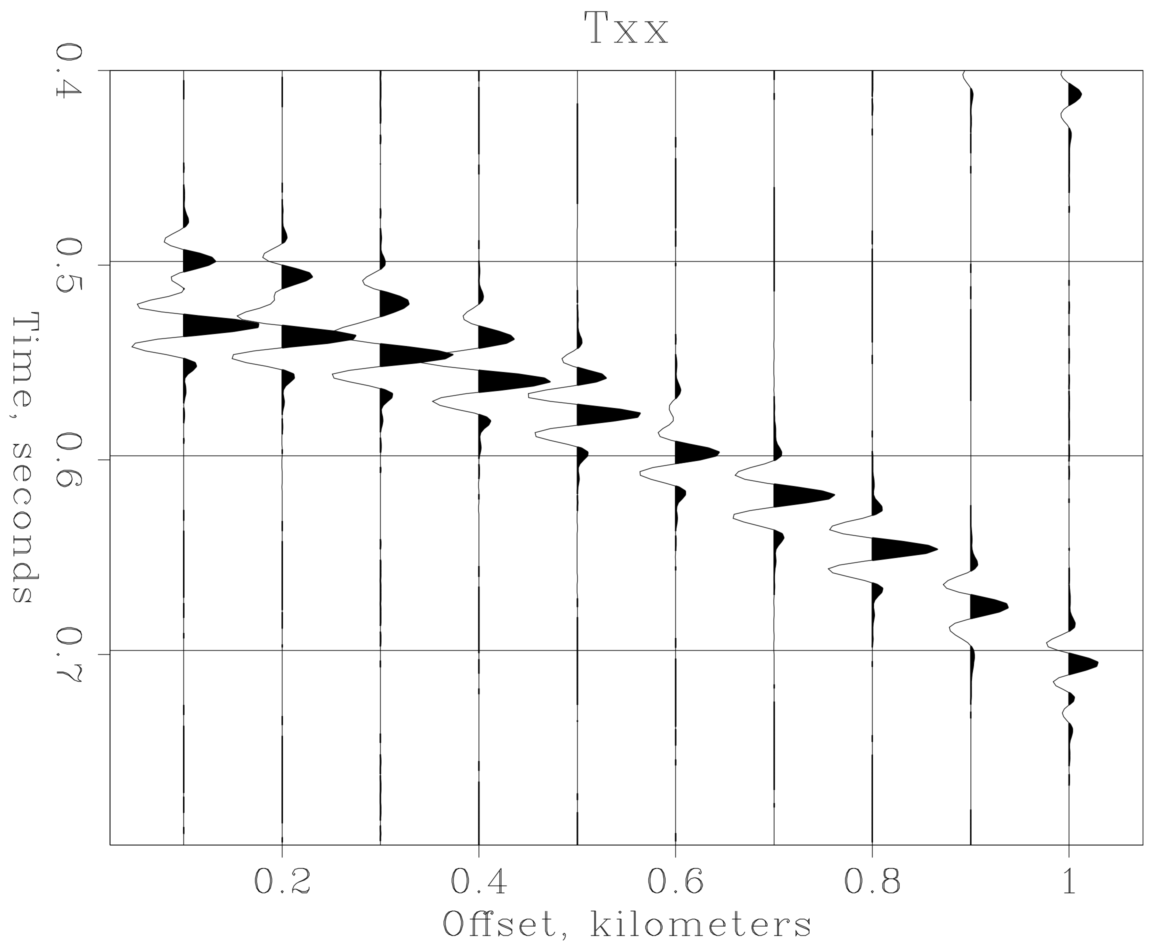

Figure 3: `Wiggle < Txx_Windowed.H par=plotpar.p | Pspen` Wiggle2 [ER]

```
Wiggle < Txx_Windowed.H transp=yes poly=yes yreverse=yes | Tube
```

Perhaps it would be better if the trace amplitudes were a little lower? The parameter `pclip` stands for "percentile clip"; the default is 98%, which is meant to scale the plotting using the effective maximum absolute value while ignoring a few huge abnormal spikes. Our data is mostly zero and has no abnormal spikes, so perhaps clipping on the maximum would be more appropriate:

```
Wiggle < Txx_Windowed.H transp=y poly=y yreverse=y pclip=100 | Tube
```

(Figure 3 shows what this plot should look like.)

Note most SEPlib programs don't care whether you type "`yes`" or "`no`", "`y`" or "`n`", or even "`1`" or "`0`". (Although a few old FORTRAN ones only accept the numbers, and a few old C ones only accept the letters.)

Looking at `Wiggle`'s self-doc you may have noticed that `Wiggle` also supports parameters like "`min1`" and "`max2`". These will usually work just like the ones in `Window` and probably seem like the preferred way to do windowing of plots. Unfortunately (for now at least) `Wiggle` does the windowing a lazy way. The whole plot is calculated (and plotted!) just as before, and the *graphics driver* does all the work of trimming away the excess. You can use these parameters of plot programs like `Wiggle` to make slight adjustments to the boundaries of a plot, or to make a plot smaller, but don't use them to "zoom in" very far! (We'll see an example of a legitimate use of these "dangerous" parameters in a few pages.)

Parameters, parameter files, and history files

If you find yourself despairing at having to remember and type huge lists of parameters like

```
transp=y poly=y yreverse=y  pclip=100
```

again and again, you will be happy to know there is a shortcut. Try putting the list of parameters above into a text file called "`plotpar.p`". Put the windowing parameters

```
n2=10 n1=200 f1=200
```

into another file called "`windowpar.p`". Then you could do

```
Window < Txx.HH par=windowpar.p | Wiggle par=plotpar.p | Tube
```

and it would be just like you had typed the full set of parameters at the "`par=plotpar.p`" and "`par=windowpar.p`". Files like "`plotpar.p`" and "`windowpar.p`" are called parameter files, and they can be nested simply by putting `par=` commands into the parameter files just like on the command line. An additional advantage of parameter files is that they can be as long as you want, so you don't have to cram everything onto one single line. (You can also put comments into a parameter file; anything after a "`#`" on a line in a parameter file is ignored, just like for csh scripts.)

What happens if the same parameter is set multiple times? The last occurrence is the only one that matters. You must pay special attention to how the parameters are written, though: a parameter can be "unset" by leaving the space after the "`=`" blank. An `=` with a space before it is ignored completely.

In summary:
`n1=1` sets `n1` equal to `1`;
`n1= 1` would unset any previous setting of `n1`, letting it default.
`n1 = 1` is a comment. It has NO EFFECT AT ALL on `n1`.

Now reread the previous paragraph again until you are sure you won't make the mistake of writing `n2 = 10` and wondering why it didn't work.

You may have already realized that a history file is just a special kind of parameter file. Before checking for parameters on the command line, SEPlib first looks for any relevant parameters in the input history file. That's how "`Wiggle`" knew the dimensions of the data in `Txx.HH` without having to be told. Of course, we can override the information in the input history file by setting another value on the command line. For example, if we do

```
Wiggle < Txx.HH par=plotpar.p n1=5000 | Tube
```

`Wiggle` will happily attempt to read past the end of the floating-point data set, resulting in an error message

```
sreed: Illegal seek
Wiggle: xdr error reading from ``in''
```

For another example of overriding a parameter set by the history file, how about changing the title of our plot from the boring "Txx" set in the history file `Txx.HH`?

```
Wiggle < Txx.HH par=plotpar.p title="\s200 \F11 Slava \c2 Trudu" | Tube
```

This changes it to a good old-fashioned *Russian* title, "Glory to Labor"! It is also possible to put superscripts, subscripts, etc, into labels and titles; do "`man vplottext`" for examples.

Even a single program like `Wiggle` has more options than we can hope to enumerate here. To see what other options are possible, look at the self-documentation and try them out. By all means don't neglect to check whether the program you are interested in might happen to have a manual page as well.

Some more plotting programs

You must be getting bored of wiggle plots by now, and we also want to show how once you get a feeling for SEPlib it is really easy to get around with new programs too. So, how about choosing another form of display? How about a contour plot instead? We know that it might sound strange to contour seismic traces, but let's try it! All we want to do is to demonstrate that since the plotting programs have a consistent interface, it is as simple as:

```
Contour < Txx_Windowed.H par=plotpar.p | Tube
```

Figure 4 shows the result.

"`Ta2vplot`" is a different kind of plotting program; it plots byte-deep rasters (`esize=1`). Since the data in `Txx.HH` is floating-point (`esize=4`), we have to plot using two steps:

```
Byte < Txx_Windowed.H pclip=100 | Ta2vplot par=plotpar.p | Tube
```

The program "`Byte`" converts from floats to bytes, and then `Ta2vplot` plots the bytes as grey-scale rasters.

Figure 5 shows the `Ta2vplot` version of our zoomed-in `Wiggle` and `Contour` plot. (Note that the rasters are centered on their associated "Offset" value.)

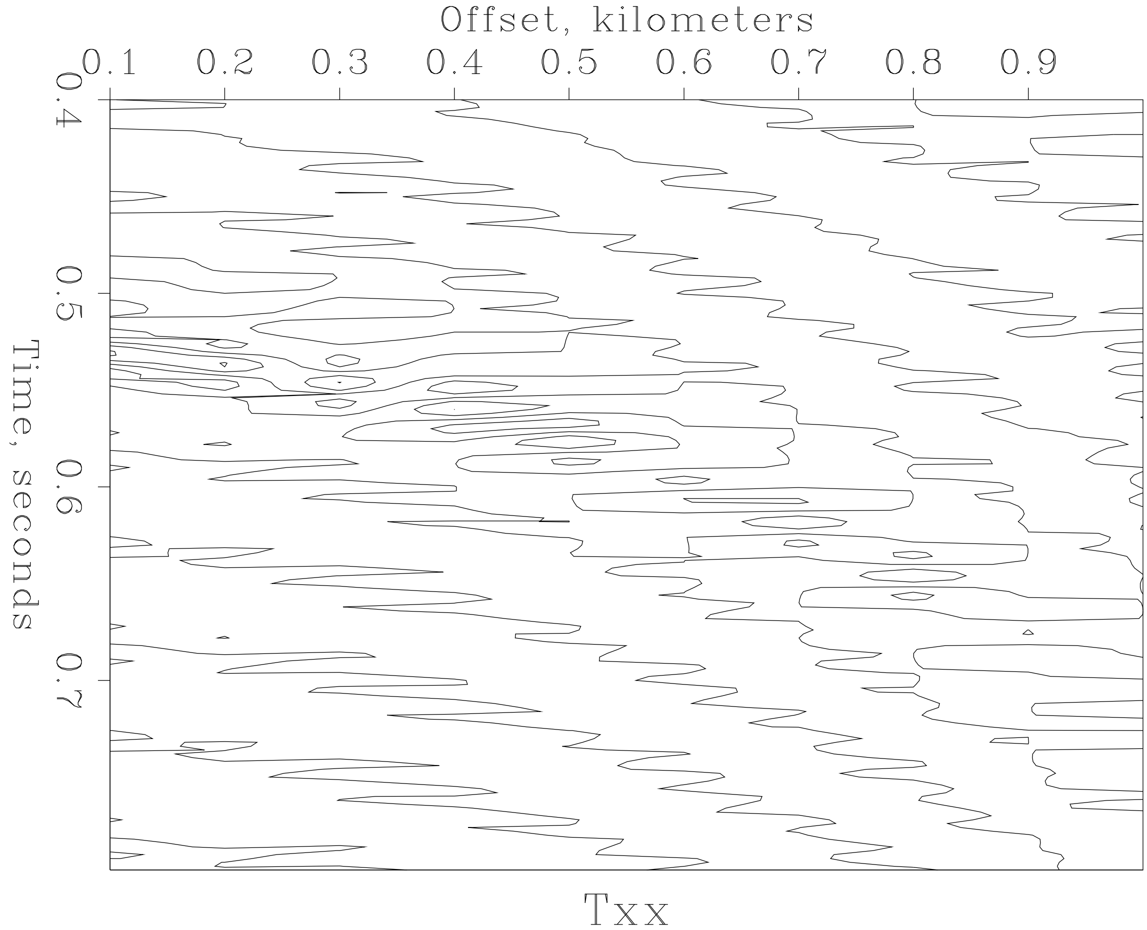

Figure 4: `Contour < Txx_Windowed.H par=plotpar.p | Pspen` Contour [ER]

Some fundamental data-manipulation programs

By this time you have learned many useful things, but as you are most probably a geophysicist or similar data-crunching creature, you may want to see something more "exciting" than just different displays. What about one of the SEP's favorite inverse operations, interpolation? Sounds much more fascinating than those boring displays, doesn't it?

We hope that you are already getting somewhat familiar with SEPlib, so from now on we will not explain all the programs in depth as we did before. Most of the programs are self-explanatory in these examples, and you should not forget about their self documentation (just type the name of the program). So let's start, and good luck!

Start by subsampling the data using `Window`:

```
Window < Txx.HH j1=2 > Txx_sub.H
```

This decimates the `1` (time) axis by a factor of 2.

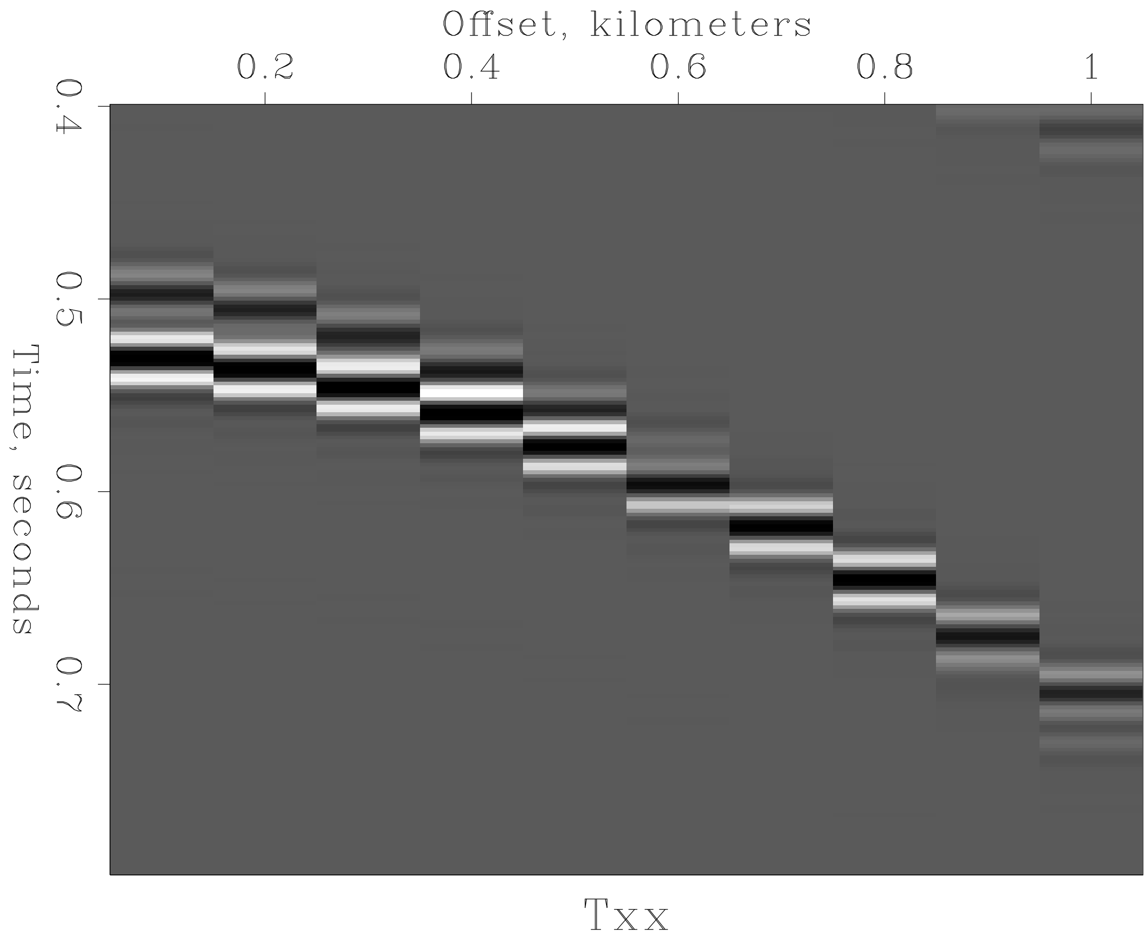

Figure 5: `Byte < Txx_Windowed.H pclip=100 | Ta2vplot par=plotpar.p | Pspen` Ta2vplot [ER]

Now suppose we accidentally lost the original data and only had the subsampled version left. If the original data was sufficiently oversampled, we should be able to recover the original by transforming to the Fourier domain, padding, and transforming back. Can we perform this sequence using only available SEPlib utilities?

The Fourier transform utility, "`Ft3d`", demands complex input (it will barf with "`Ft3d: esize must be 8`" if it doesn't get it), so we first convert from Real to Complex using `Rtoc`. Then we can use `Ft3d` to do the Fourier transform, with a command-line parameter to tell it we only want to transform over the `1` axis:

```
Rtoc < Txx_sub.H | Ft3d sign1=1 > Txx_Fft.H
```

You may enjoy examing a plot of Complex absolute value in the frequency domain:

```
Cabs < Txx_Fft.H | Wiggle | Tube
```

Now we pad from 512 back to 1024 points, inverse Fourier-transform, take the real part, make a Wiggle plot, and plot it (whew)!

```
Pad < Txx_Fft.H n1=1024 | Ft3d | Real | Wiggle par=plotpar.p | Tube
```

(We didn't have to tell `Ft3d` what to do because it looks at the parameters from its last run preserved in the history file, and by default does the inverse.) And the resulting plot is... JUNK!

Well, of course. "`Pad`" is a stupid program that just appends enough zeroes onto the end of a given axis to make a dataset of the requested size. That's fine for some applications, but not for this one. To do the job right we have to separate out the positive frequencies, negative frequencies, DC, and Nyquist and make sure they all end up in the right places. Zero frequency is in slot 0. We have 512 frequency points, so the positive frequencies must be in slots 1 to 255, the Nyquist is in slot 256, and the negative frequencies are in slots 257 to 511. (Almost all SEPlib programs use C numbering notation, not FORTRAN numbering notation... arrays start with element 0, not 1.)

Let's separate them all out.

```
Window < Txx_Fft.H n1=1 f1=0 squeeze=n > DC.H
Window < Txx_Fft.H n1=255 f1=1 squeeze=n > Pos.H
Window < Txx_Fft.H n1=1 f1=256 squeeze=n > Nyquist.H
Window < Txx_Fft.H n1=255 f1=257 squeeze=n > Neg.H
```

(If the `1` axis on output would have length 1, `Window` by default helpfully swaps it around and makes it the `2` axis. For this case it is not helpful. The "`squeeze=n`" option prevents this.)

The Nyquist has to go in twice, so we need to divide it by two:

```
Add Nyquist.H scale=.5 > Nyquisto2.H
```

And finally we put it back together with the padding in the middle, and bring it back to the time domain:

```
Merge axis=1 space=n DC.H Pos.H Nyquisto2.H | Pad > Junk n1=768
Merge axis=1 space=n Junk Nyquisto2.H Neg.H > Txx_Fft2.H
Ft3d < Txx_Fft2.H | Real > Junk
Add Junk scale=1.414213562 > Txx_resamp.H
Rm Junk DC.H Pos.H Nyquisto2.H Neg.H Txx_Fft.H
```

(`Merge` by default helpfully inserts a zero trace between merged blocks, so you can see where the boundaries are. "`space=n`" turns this off. We have to rescale by $\sqrt{2}$ to take care of the differing FFT normalizations.)

Compare the original and reconstructed data by doing

```
Cat Txx.HH Txx_resamp.H | Wiggle par=plotpar.p | Tube
```

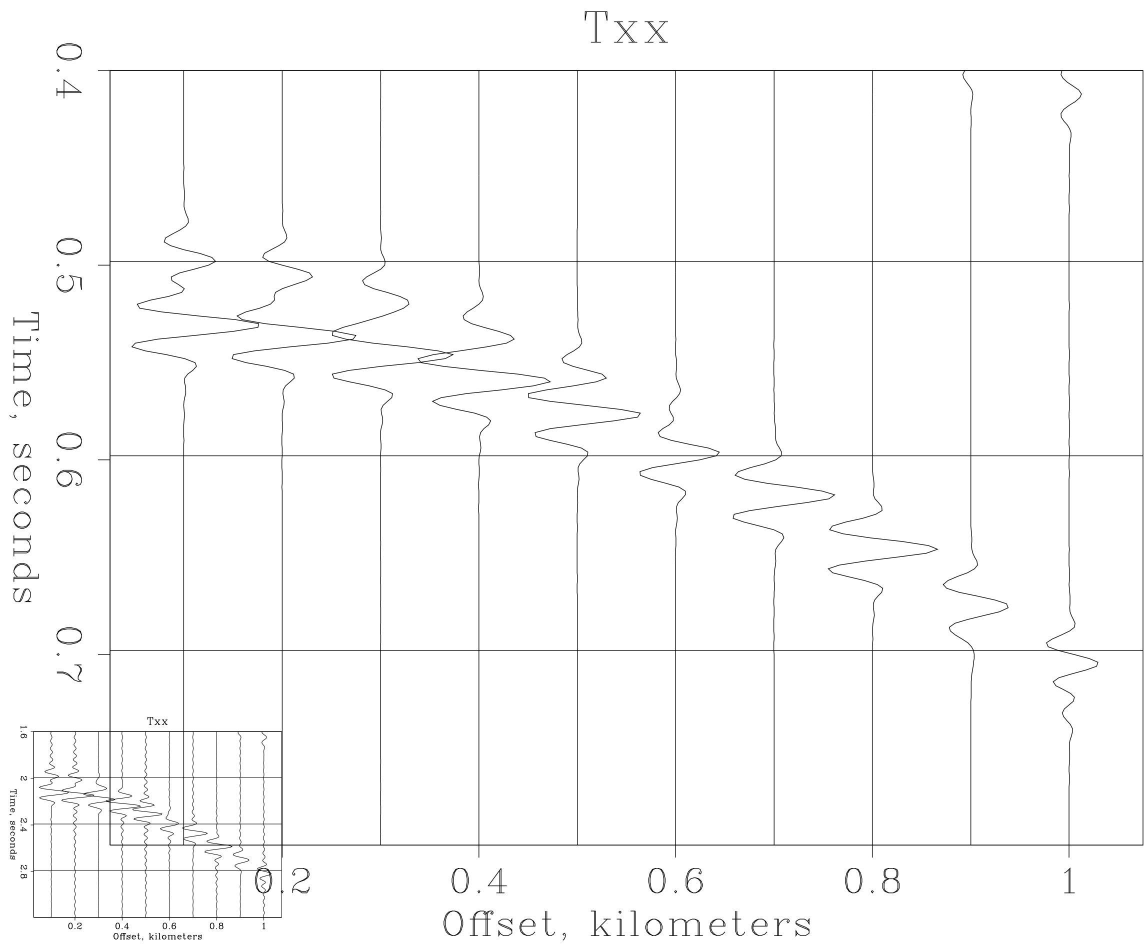

Figure 6: `Cat Txx.HH Txx_resamp.H | Window par=windowpar.p |\ Wiggle overplot=y par=plotpar.p poly=n | Pspen` Fourier [ER]

As you (hopefully) can see, we appear to have recovered the original dataset exactly.

Hopefully you enjoyed this example and you are ready to try more. If you are familiar with UNIX you can build a shell script to do all this automatically so that you don't have to retype it again and again. If you are really ambitious, create a shell script that lets you specify the power of two to subsample by. Figure 6 shows what happens if you subsample by a factor of *four* and reinterpolate back using our Fourier-domain method. (This time we subsampled too far and our sinc functions got out of control!)

Different people use SEPlib programs different ways. Some prefer to type everything themselves as we have been doing here. Some write everything as shell scripts. Others like to do everything from within Makefiles (or Cakefiles). How you use SEPlib is up to you.

A reflection-seismic utility

Now let's try something a little more Geophysical. First, try hyperbolic stacking over a sweep of velocities:

```
Velan < Txx.HH nv=50 v0=1.40 dv=.08 > Txx_Velan.H
echo 'label2="Velocity, km/s"' >> Txx_Velan.H
Contour < Txx_Velan.H | Tube
```

(`Velan` does not update `label2`, so we used the standard UNIX command "`echo`" to append a new label onto the end of the history file.)

The P-wave arrival seems to stack in at a velocity of about 3.4 km/sec; let's move out at that velocity:

```
NMO < Txx.HH vel=3.4 | Wiggle pclip=100 max1=.8 poly=y | Tube
```

Hyperbolic move-out flattens out the P wave almost perfectly.

The shear arrivals seem to stack in at a velocity of about 2.05 km/sec; let's move out at that velocity:

```
NMO < Txx.HH vel=2.05 | Wiggle pclip=100 max1=.8 poly=y | Tube
```

The picked move-out velocity appears to split the difference between two distinctly non-hyperbolic shear waves.[6] (See Figure 7).

Applying a filter

For a more time-series oriented example, how about convolving a dataset with a filter? First create the filter by entering a list of numbers into a text file (using your favorite editor). Call it "`filterfile`". Then use the SEPlib utility "`Dd`" to convert that ASCII list into a SEPlib floating-point data file:

```
Dd in=filterfile ein=0 esize=4 > filter.H
```

Note that in this example we specified where the actual data was by specifying "`in=filterfile`" directly on the command line. The command-line parameter "`ein=0`" tells `Dd` that the input is ASCII. `Dd` then creates an output SEPlib history file with `esize=4` (meaning floating-point binary data) and `n1` equal to the number of entries found in "`filterfile`".

Having created a history file for the filter, we can apply it to the data in `Txx.HH` using the program `Filter`:

```
Filter < Txx.HH filter=filter.H > Txx_Filtered.H
Wiggle < Txx_Filtered.H par=plotpar.p | Tube
```

[6] Of course you should have expected an anisotropic example given the first author!

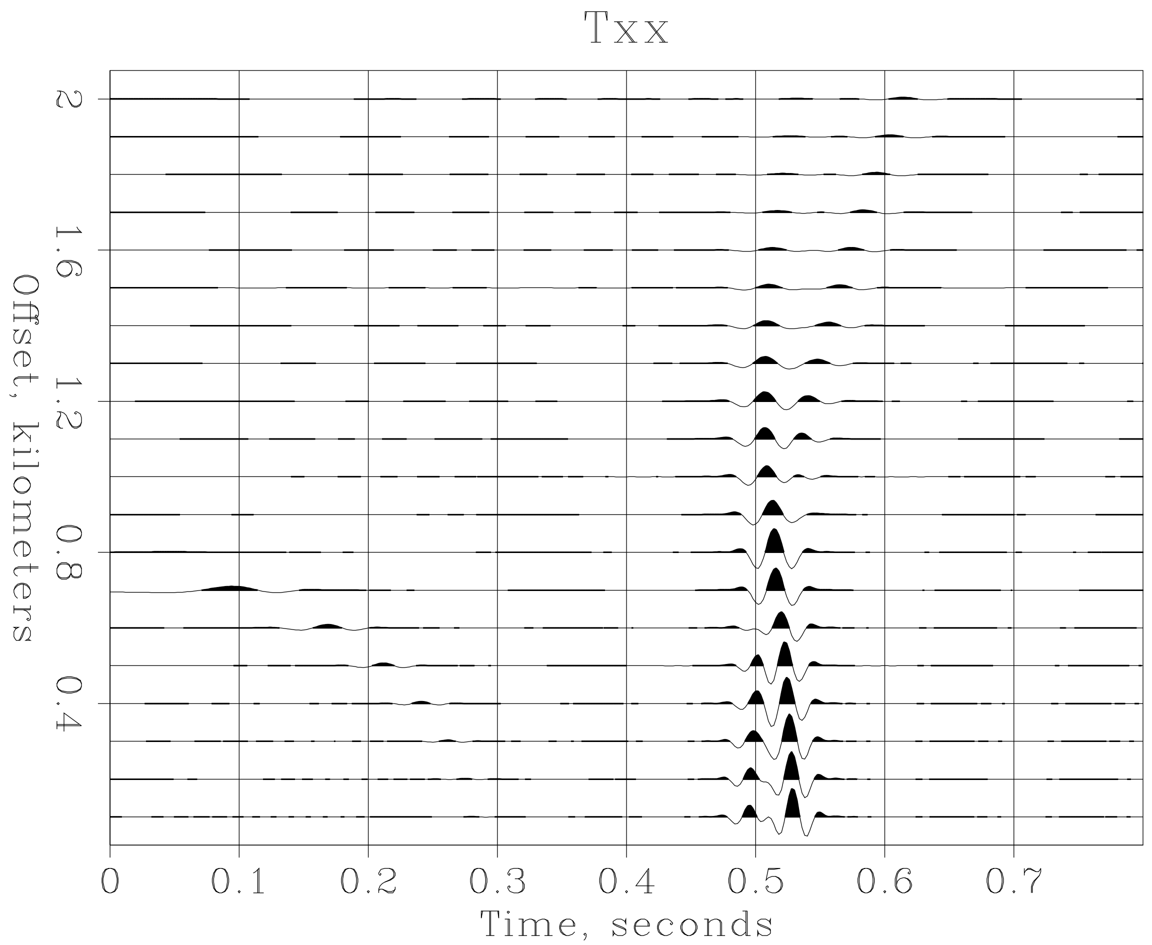

Figure 7:
`NMO < Txx.HH vel=2.05 | Wiggle pclip=100 max1=.8 poly=y | Pspen` NMO
[ER]

Here the file `filter.H` is an "auxiliary input file". (Note if the data file pointed to by the history file `filter.H` contained just one entry, a one, then `Txx_Filtered.H` would be an unmodified copy of `Txx.HH`.)

Auxiliary files are used by many programs (both for input and output) that require more inputs and outputs than the standard one-file-in one-file-out SEPlib structure allows. Usually one or more axes of the auxiliary files correspond to axes in the input or output files. For our example, axis `1` in the input history file `Txx.HH` corresponds to axis `1` in `filter.H`; thus they should have the same sampling rate, and the parameter `d1` in both files should ideally match. As it turns out `Filter` doesn't check for this. Some programs do, however; and in some SEPlib programs the standard parameters (`n1`, `n2`, `n3`, `d1`, `d2`, `d3`, `o1`, `o2`, `o3`) found in the auxiliary files may be required and significant.

Advanced plotting

By the time you reach this point you should know everything you need to start exploring SEPlib on your own. Before finishing up our tour we would like to introduce you to some tricks that might help you to demonstrate your results better.

Often it is useful and spectacular to plot a `Wiggle` plot on top of a raster plot. That is a little tricky, but not too hard.

First make our windowed input data again (if we don't still have it):

```
Window < Txx.HH min1=.4 max1=.8 max2=1. > Txx_Windowed.H
```

Now run `Ta2vplot` once to make the "background" raster plot:

```
Byte < Txx_Windowed.H pclip=100 | \
 Ta2vplot par=plotpar.p min1=.4 max1=.8 max2=1. min2=.05 \
 out=file1.v head=/dev/null
```

(For this example we won't need the history file with its parameters any more, just the data file containing vplot graphics commands. The `out=file1.v` tells `Ta2vplot` to write the output to "`file1.v`"; the `head=/dev/null` throws the output history file away to the UNIX garbage-can device "`/dev/null`".)

Finally we run `Wiggle` *twice*, once with thick "invisible" traces and once with half as thick standard yellow ones:

```
Wiggle < Txx_Windowed.H par=plotpar.p min1=.4 max1=.8 max2=1. min2=.05 \
 poly=no out=file2.v head=/dev/null plotcol=0 plotfat=10
Wiggle < Txx_Windowed.H par=plotpar.p min1=.4 max1=.8 max2=1. min2=.05 \
 poly=no out=file3.v head=/dev/null plotfat=5
```

Note how we had to specify *all four limits* to be sure that the plots produced by the very different plotting programs `Ta2vplot` and `Wiggle` would be compatible. (For some other plot programs even this isn't enough; we also have to turn off the "padding" between plot and axes.)

Now for the "advanced graphics": to combine the three plots, we use a special `pen` filter called "`vppen`", which reads in *and writes out* the vplot graphical files used by SEPlib programs.

```
vppen file1.v file2.v file3.v erase=once vpstyle=no | tube
```

`vppen` is not itself a SEPlib program, hence the lower-case initial letter and why we had to use `out=` above. (There is a SEPlib version called `Vppen` that we could have used, but history files become less useful at this point.)

Note that we didn't really need to use `vppen` above; we could have just done

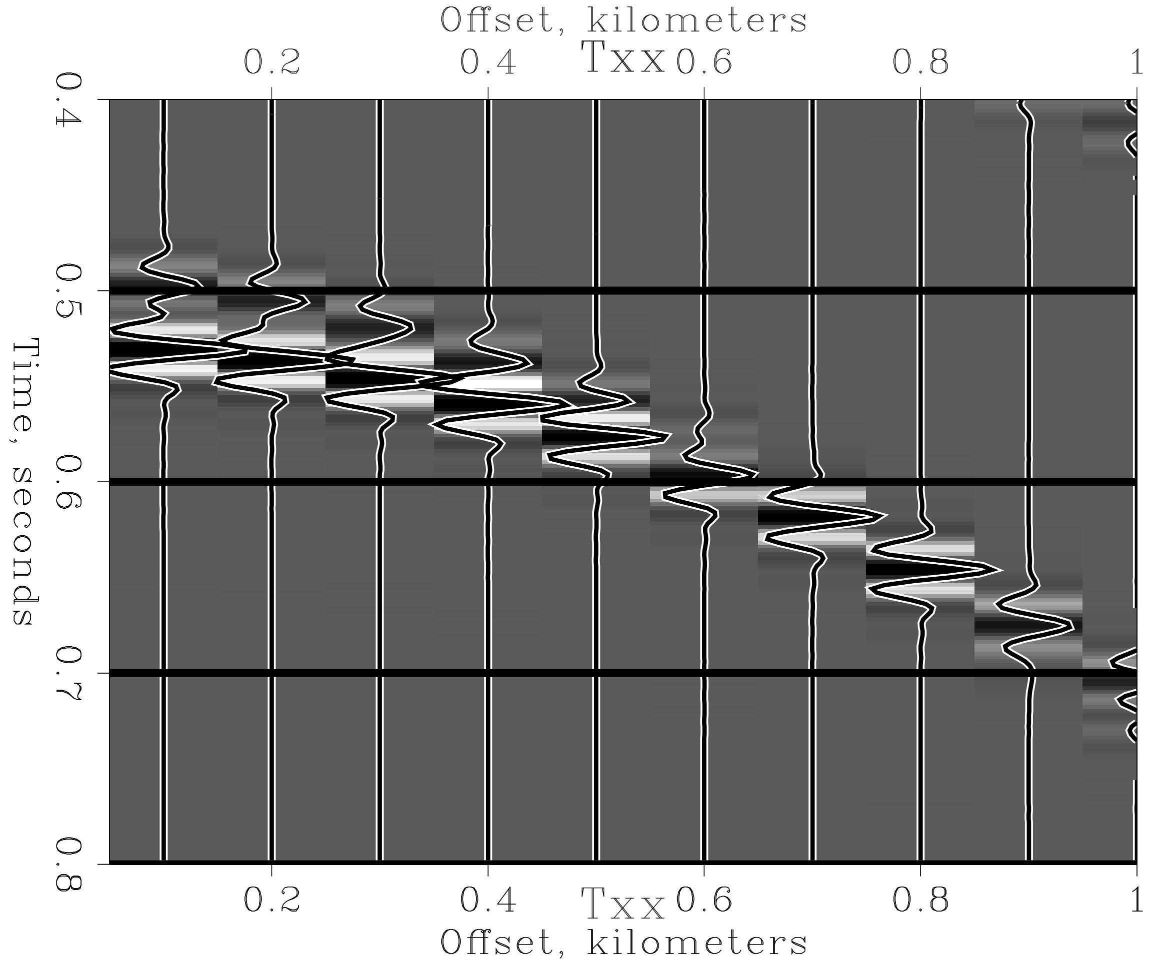

Figure 8: `vppen file1.v file2.v file3.v erase=once vpstyle=no | pspen` Overplot [ER]

```
tube file1.v file2.v file3.v erase=once
```

directly. It is useful to do such manipulations with **vppen** because it lets us save the composite plot as a single vplot file, in effect "flattening" the composite into a single plot. There is no magic involved here; **vppen** is a vplot pen filter just like **tube**, and shares almost all the same code. The only difference is that while **tube** draws plots on screens, **vppen** writes out "vplot graphical language". Suppose you plot something on your screen using **tube** with complex options and multiple input files, and want to somehow save your graphical masterpiece in such a way that you can reproduce it again later without so much trouble with options and files. Simple: just replace **tube** with **vppen** and redirect the output. You have now saved your masterpiece as a single vplot file.

If you are looking at our example plot on your screen right now you might be wondering why we went to the trouble of creating the "invisible wiggles" file `file2.v`. We hope that after you examine the hardcopy version in Figure 8 the reason should become evident!

Examining Figure 8 you may also notice there are *two* titles. While various plot programs like `Wiggle` and `Ta2vplot` are consistent in their options for things like where to put the title, which way to put axes, etc, for historical reasons they are *not* consistent in their defaults. So if you want to get plots from different programs to exactly overlay in general you have to specify everything. Since we did not specify whether to put the title above or below the plot, we got the (different) default for each. You shouldn't have trouble getting around such annoyances. (Usually you just turn off the axes and labels for all the parts but one.)

Plot matrices

Perhaps the previous example seemed a little technical for you. How about this one then? Often you want to compare two plots side by side (or squeeze more figures into your expanded abstract). `vppen` can do that too, but you may find it easier to use a utility shell that calls `vppen` with the correct arguments for you:

```
vp_SideBySideIso file1.v file3.v | tube
```

Figure 9 shows the results.

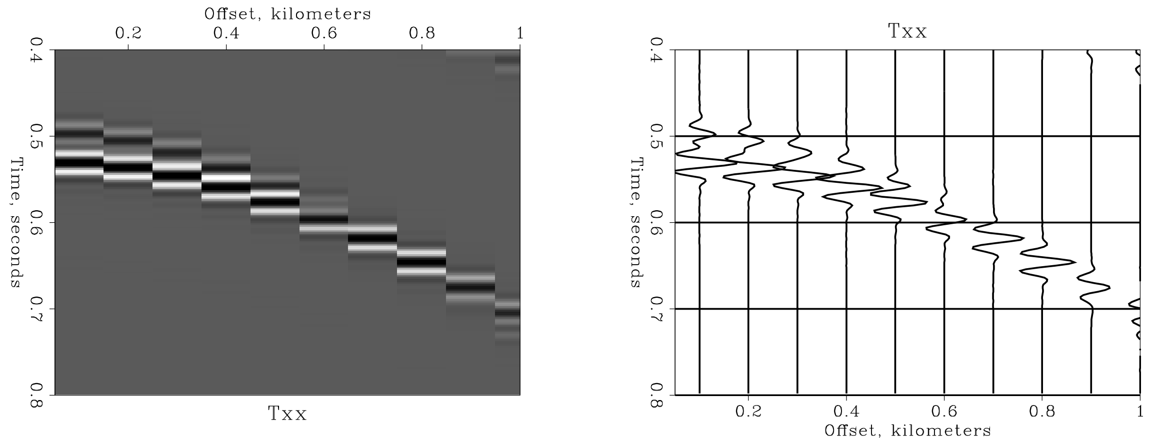

Figure 9: `vp_SideBySideIso file1.v file3.v | pspen` SidebySide [ER]

Don't forget to

```
rm file[1-3].v
```

with a *lower-case* `rm` when done!

CONCLUSIONS

This concludes your SEPlib tour. We don't expect you will become a SEPlib wizard in one reading, but hopefully after rereading this document a few more times and trying out some examples of your own you will have learned enough to get by.

QUICK SUMMARY

SEPlib programs begin with a capital letter. Almost all of them read the standard input and write to the standard output. Type the name of a SEPlib program without arguments and it will give you a brief documentation.

SEPlib parameter files contain lists of parameters in the format `parameter=value`. There is NO SPACE on either side of the `=`. Parameters can be specified either in a history file, directly on the command line, or by specifying `par=file` (where `file` is a parameter file) on the command line or in a parameter file. The last occurrence of a parameter is the one that is used.

SEPlib history files are a special kind of SEPlib parameter files. Programs *append* information onto the history file, so it always contains a history of the programs run and parameters specified that went into making the file.

History files must contain at a minimum the parameters `in`, `esize`, and `n1`. The parameter `in` gives the file system location of the associated data file, using either a relative or absolute pathname. The parameter `esize` gives the size of the individual elements in the data file. `esize=1` means bytes, `esize=4` means IEEE single-precision floating-point, `esize=8` means complex or floating-point coordinate pairs. (As a special case `esize=0` means the data file is ASCII. Only a few programs recognize this.)

The data file is a *flat* UNIX file, assumed to consist of a regular "cube" of data with fastest dimension `n1` (must be specified), next fastest `n2` (defaults 1 if not specified), and slowest `n3` (defaults 1 if not specified). The size of the data file in bytes should be equal to `esize * n1 * n2 * n3`. (As a special case `n1=-1` means the data file is of unknown length. Only a few programs recognize this.) Some programs allow 4-dimensional datasets using `n4`.

Optional but commonly used parameters in history files include:
`d1`, `d2`, `d3`: sampling rates for fastest, middle, and slowest axes.
`o1`, `o2`, `o3`: coordinate of first element on fastest, middle, and slowest axes.
`label1`, `label2`, `label3`: axis labels or physical axis dimensions.
`title`: title.

For other parameters recognized by particular programs refer to self-docs.

The programs were written by dozens of people over a span of years, and are of variable quality as to their robustness and generality. Every program included in the distribution has passed minimum tests of usefulness and usability.

ACKNOWLEDGMENTS

Robert Clayton introduced the original parameter-fetching method. Jon Claerbout introduced history files. Stew Levin got pipes to work and brought the code to the high standard we have enjoyed since about 1983. (The preceding was borrowed with trivial modification from Claerbout (1991).) We want to express our gratitude to everyone who contributed to the development of SEPlib and made it possible for all of us to use this powerful set of data processing tools.

Thanks to Lin Zhang for installing Joe's Hawaii bug fixes at Stanford. Thanks to Dave, Martin, Steve, and Jon for putting up with occasional barrages of e-mail from 2400 miles away, and with the even more trying questions asked in person at their desks. Any errors in this paper can be attributed to the fact that the two co-authors have never met (and used only e-mail for communication).

The test data is courtesy of Bertram Nolte and L. Neil Frazer.

REFERENCES

Claerbout, J., 1991, Earth Soundings Analysis: Processing versus Inversion, Appendix B, Seplib and SEP software: Blackwell Scientific; also SEP–**71**, 283–288.

Claerbout, J., 1990, Introduction to `seplib` and SEP utility software: SEP–**70**, 413–436.

Claerbout, J., 1986, A canonical program library: SEP–**50**, 281–290.

Cole, S., and Dellinger, J., Vplot: SEP's plot language: SEP–**60**, 349–389.

Dellinger, J., 1989, Why does SEP still use Vplot?: SEP–**61**, 327–335.

APPENDIX A: Basic troubleshooting

GENERIC UNIX ENVIRONMENT

Hopefully you are lucky enough to have everything installed so that you the user don't have to worry about any environment variables. Not all of us can be so lucky. If you do need to set some environment variables to run SEPlib, this appendix should tell you how to do it. Of course this appendix can only be correct if the file
`/sepcd5/tutorial/septour/sitepaths.tex`
has been updated for your site.

Executable path

If you type "`Wiggle`" without arguments and get:

```
Wiggle: Command not found.
```

Then `/sepcd5/MACHINE_TYPE/bin` is not in your `PATH`. Your `PATH` is the list of directories that the computer looks in for commands to execute; you can see it by doing

```
echo $PATH
```

Assuming your `PATH` does not already end in a "`:`", you can add SEPlib programs to your path by doing

```
setenv PATH $PATH":/sepcd5/MACHINE_TYPE/bin"
rehash
```

(If your `PATH` already had a colon at the end, don't add another because you'd get two colons in a row in your `PATH`, which is probably not a good idea.)

If the above path contains something like "ARCH_TYPE" or "MACHINE_TYPE", then your site is set up to allow multiple CPU architectures. You will need to replace that part of the path with a tag that depends on the kind of machine you are using.

If you still can't run SEPlib programs, try doing

```
ls /sepcd5/MACHINE_TYPE/bin
```

If the directory does not exist, or you don't have permission to run things there, or the directory appears to be empty, you will need to ask your computer support person to fix this problem.

SEPlib source root

If you type "`Wiggle`" and get

```
doc(): No such file or directory
```

you will need to:

```
setenv SEP_DOC_PATH /sepcd5/sep93.1
```

Getting the manual pages

If you type "`man seplib`" and get

```
No manual entry for seplib.
```

you will need to add
`/sepcd5/man`
to your `MANPATH`:

```
echo $MANPATH
setenv MANPATH $MANPATH":/sepcd5/man"
```

(assuming your `MANPATH` did not already end in a ":"). Note the "seplib" man page is intended for those interested in writing their own SEPlib programs.

You may find that sometimes SEPlib man pages get "hidden" behind unrelated man pages that happen to have the same name. For example if you do "`man vplot`" you might see

```
vplot - plot graphics for a Versatec printer
```

This is *not* the man page for SEPlib's vplot. If this happens you should be able to get the right man page by being more specific, for example you could try doing "`man l vplot`" to tell `man` you want only the "local" man pages. (This exact command might not work at your particular site; it depends on where the SEPlib man pages have been installed.)

Under some flavors of UNIX the "`man`" command is buggy and gets confused by man pages with "similar" names. For example if you do `man vplottext` you may always get the `vplot` man page, which isn't what you asked for. (This happens with CONVEX UNIX 9.0.) If all else fails look around in the directory where the man pages live,
`/sepcd5/man`,
find the man-page source for what you want, and do something like

```
nroff -man vplottext.l | more
```

to format it. (And of course complain to the vendor!)

Scratch directory

If you do

```
Spike n1=10 n2=10 > Junk.H
```

and get something like

```
output(): No such file or directory
Spike: output() unable to open output file /sepcd5/scr/joe/_Junk.H@
```

You need to create a subdirectory with your username under the directory

`/sepcd5/scr`

Alternatively, read the section "History files and datapaths" in the main body of the text and learn how to use datapaths.

Vplot directories

If you do

```
echo "e" | plas | vppen > file
```

and get

```
vppen: (warning) unable to create temporary file
vppen: (fatal) copy of piped input failed
```

Then either create the directory

`/sepcd5/scr/tmp`

or complain to the person who compiled SEPlib that they didn't set the `PEN_SPOOL` compile flag correctly when they did it.

If you do

```
Spike n1=10 n2=1 | Graph title="\F10 Test" | Tube
```

and get a warning message

```
(gentext) Perhaps you need to setenv VPLOTFONTDIR on this machine?
```

you need to do

`setenv VPLOTFONTDIR /sepcd5/MACHINE_TYPE/lib/vplotfonts`

Postscript printer

You can change the default printer used by `Pspen` by setting the environment variable `PSPRINTER`. If you wanted to use the printer "sp405" by default, for example, you would do

```
setenv PSPRINTER sp405
```

You can override this on `pspen` command line with the getpar option `printer=`.

When you try using `Pspen` you may sometimes get no plot or a partial plot. Probably in such a case you got the message
`lpr: copy file is too large`
or a flurry of
`/var: file system full`
complaints on the console of the workstation attached to the printer. Most probably you just generated a postscript file using `Pspen` that is bigger than your default printer spooler setup allows.

The best solution is to get your default printer spooler setup changed. If that is not possible, instead of `Pspen` do

```
Pspen out=file.ps
```

This causes `Pspen` to write the postscript output to "`file.ps`" instead of trying to ship it off to the printer directly. Next log in to the workstation attached to the postscript printer, `cd` to the directory you were working in, and send the plot on its way manually by doing

```
lpr -Pprinter_name -s file.ps
```

Here "`printer_name`" is the name of the printer you want to use (if you've set the `PSPRINTER` environment variable you could just use `-P$PSPRINTER`). The "`-s`" option tells `lpr` not to copy the file to the spool directory but to use it from where it is. Cumbersome and annoying, but what you may have to do at some sites.

Screen graphics, or lack thereof

If "`Tube`" complains

```
Tube: No Graphics available on that device!!
```

or calls up an inappropriate vplot "pen" filter for your device, remember `Tube` just looks at your environment variables to try to figure out what sort of graphics might work. If you don't have `TERM` set to something it understands, it will get confused.

If you think `Tube` is getting confused, try looking at the source for `Tube` by doing

```
more `which Tube`
```

and see if you can figure something out. The list of "pen" filters in `Tube` may not correspond to the set of installed filters at your site; do

```
ls /sepcd5/MACHINE_TYPE/bin/*pen
```

to see what vplot "pen" filters seem to be available.

If you simply don't have any screen graphics device or just can't get screen graphics to work, whenever you see "Tube" used to make screen plots in this document make a postscript hardcopy using "Pspen" instead (like in the figure captions).

Note there is a pen filter for Suntools (`Sunpen`), although it is a bit buggy and unmaintained, and there are always old tektronix 4010 emulators around (`Tekpen`). If you don't have a postscript printer, there is a "pen" for Imagen Impress-language printers that used to be SEP's workhorse for many years (`Ipen`). There is even a "pen" for printronix line printers that works surprisingly well, if you are happy making large plots (`Lprpen`). These pen filters may not be included in your particular SEPlib distribution, although they do exist and work.

MACHINE-SPECIFIC ENVIRONMENT

SEPlib is currently being used on three different types of workstations at Stanford: DEC 3100, IBM RS6000 and SUN4. Here we would like to provide some machine-specific information.

X applications resource directory

If you are using the X-window system (remember Sun's OpenWindows is a variant of X), look to see what your "X applications resource directory" is by doing the command "`echo $XAPPLRESDIR`". (Probably it is undefined.) If you are getting strange behavior or error messages from some SEPlib-related X programs and suspect it might be due to an X-resource problem, try setting the "X APPLications RESource DIRectory" by doing

```
setenv XAPPLRESDIR /sepcd5/MACHINE_TYPE/app-defaults
```

Unfortunately, other programs that you use on your system may require some different value for this environment variable. (Note it is a directory, not a path; you can only specify one place to look.) One possible solution is to have all the X-resource files lumped together into one directory. Another solution is to create a little shell script for the troublesome commands; the shell script resets `XAPPLRESDIR` to the required value and then calls the command. (Note this will not affect the value of `XAPPLRESDIR` *outside* the shell.)

Load library path on SUN

If you are running OpenWindows or the X-windows system on a Sun workstation and get something like

```
ld.so: warning: /usr/openwin/lib/libXt.so.4.0 has older revision than expected
ld.so: warning: /usr/openwin/lib/libX11.so.4.3 has older revision than expected
ld.so: Undefined symbol: _XtStrings
```

when you try running "`Xtpen`", the culprit is your `LD_LIBRARY_PATH`.

This is a list of directories the computer looks through to find runtime-loaded subroutines used by the executable but not contained within it. Examine your `LD_LIBRARY_PATH` by doing

```
echo $LD_LIBRARY_PATH
```

Assuming your `LD_LIBRARY_PATH` does not begin with ":", *prepend* the required X library directory to the beginning by doing

```
setenv LD_LIBRARY_PATH "/sepcd5/MACHINE_TYPE/lib:"$LD_LIBRARY_PATH
```

You need to do this so that certain X11 subroutines have the opportunity to be merged into the executable *before* older incompatible versions of the same routines (needed by OpenWindows) can force their way in. (OpenWindows programs will still work; the loader is smart enough not to attach new subroutines to an old program.)

Alternatively, ask the person who installed SEPlib at your site to relink everything using static linking. (It may also be possible for them to relink everything so that the correct load-library path for your site is automatically "built in" to the executables.)

Xtex under OpenWindows

If you are trying to use `xtex` on a Sun workstation but can't seem to get the figures to work because `xtex` always crashes and sends lots of gibberish like

```
process(0x430088, 'oas.Stanford.EDU NeWS client', runnable)
Error: /undefined
Command: XResource
Ostack: 11534436 /XLookupID
```

to your console window, you are trying to use OpenWindows 3 with an `xtex` written for OpenWindows 2.

Either get a new version of `xtex` that works, find a workstation still running 2.0, or rerun the document with figures turned off. (And of course complain to Sun for gratuitously changing their software in non backwards-compatible ways with each OpenWindows upgrade.)

APPENDIX B: SEPLIB PROGRAM GLOSSARY

This appendix gives brief descriptions of almost all of the SEPlib programs that come with the standard SEPlib release for widespread distribution. A few closely related non-SEPlib vplot utilities have been included as well, at the end.

To find out more, read the self-documentation. Most programs do not have manual pages (or very current manual pages), so if you need to know more than what you find in the self-doc you'll probably have to look at the source code.

The graphical language used by these programs is called "vplot". SEPlib's "vplot" has about as much to do with Versatec's "vplot" as calculus does with roman numerals. See the references for some articles describing vplot. While vplot does have several technical manual pages that describe it in great detail, a user's guide is unfortunately lacking.

SEPlib Programs

Note that the Names of these Programs all begin with an Upper-Case Letter.

Add A versatile program for doing element-by-element mathematical manipulations on one or more data files. It can be used to form linear combinations of two or more files, multiply two or more files, or divide one file by another. It can also be used to scale a file or take the absolute value, square root, logarithm, or exponential.

Again Take the arctangent of a floating-point data file element by element.

Agc Automatic gain control with first-arrival detection.

Aniso2d Two-dimensional anisotropic heterogeneous elastic modeling.

Arcy Convert SEGY tapes or files to SEP's version of SEGY ("sepy"), or (with an option) to SEPlib data cube files.

Balance Perform trace balancing.

Bandpass Butterworth bandpass filtering. See also **Lpfilt**.

Box **Box** outputs the vplot commands to draw a balloon-style label. Options let you position the box and pointer, control the size of the labeling, etc. (It is even possible to draw boxes with perspective.) The boxes can be overlayed onto vplot graphics files using the "erase=once" option of pen filters. For example, "`Tube < Wiggleout.H Boxout.H erase=once`". The interact option of pen filters can be used to determine the coordinate of the spot the box's pointer is to point at. (Alas, not all pen filters support the "interact" capability.) The special pen filter **Vppen** can be used to combine vplot files.

Byte Convert floating-point SEPlib data (esize=4) to byte-deep SEPlib raster data (esize=1). Usually used in conjunction with **Ta2vplot** or **X11movie**. The clip value is determined by the option "pclip" for "percentile clip". pclip=50 gives the median, pclip=100 the maximum, etc. The percentile clip can be calculated based on one particular input panel or all the input panels. (It is also possible to simply specify your own clip, which can speed the program up tremendously.) The input data is assumed to be about equally split between positive and negative values; an option is available for mapping input data that is all positive to the entire possible range of output values. Several other conversion options are available that are useful for bringing out hidden features in data, such as a sign-preserving gain parameter "gpow".

Cabs Complex (esize=8) to real (esize=4) conversion; take the absolute value of complex-valued data. (Alternatively, if you consider the input data to be (X,Y) coordinate pairs, output the Euclidean norm.)

Cat A synonym for "`Merge space=n`".

Cfft Complex fast-Fourier transform. Requires complex-valued input data (esize=8).

Clip Find a "clip" value for the input data, and put it into the header. With appropriate options performs several sorts of clipping on the data as well, such as changing all clipped or unclipped values to some given value, etc. The clip value is determined by the option "pclip" for "percentile clip". pclip=50 gives the median, pclip=100 the maximum, etc.

Cmplx Combine two real (esize=4) data files into one complex data file (esize=8). Note that some programs such as **Graph** treat esize=8 data files as (X,Y) coordinate pairs, so this program can also be thought of as a way to combine an "X" and a "Y" file into an "(X,Y)" file. See also **Real** and **Imag**.

Conj Take the complex conjugate for each element of a complex-valued dataset. (I.E., change the sign on the second real number in each element of an esize=8 dataset.)

Contour Input a real-valued (esize=4) dataset and output vplot commands for a two-dimensional contour plot of the data. (The vplot output can be viewed on a screen using the program **Tube**, or plotted on a postscript printer using **Pspen**.) **Contour** has many, many options to specify at what values to draw contours, where to position the plot on the page, how big to make the plot, which way to draw the axes, where to place tick marks and labels, etc, etc, etc. All of these parameters attempt to default reasonably. **Contour** also allows auxiliary input files which can be used to annotate the contour plot with symbols, curves, or arrows. You may find the utility programs **Window** and **Reverse** useful for pre-processing data to be plotted with **Contour**. See also **Vppen** and **Box** for a crude way of adding annotation, and **plas** and **pldb** for a crude way of editing.

Dd Convert data from one esize to another. Possibilities for esize are 0 (ASCII), 1 (byte-deep raster), 2 (short integer), 3 (real with least-significant byte truncated), 4 (real), and 8 (complex). **Dd** currently attempts to perform all conversions in core, so it is only useful for converting relatively small datasets.

Disfil Formatted display of a binary data file. Allowable input types are real (both IEEE and native), integer, complex, and byte. The default depends on the input esize but can be overridden from the command line. There are several options that can be used to control the format of the output ASCII data if you don't like the default. There are also options for changing the reading frame or only showing some subset of the input data. The default is to start at the beginning and show everything.

Dot Read in two files (one from standard input, one from an auxiliary input). The data read from standard input is passed to the output unchanged. The header file of the output file will have the value of the dot product of the two files appended onto the history as a new parameter "dotprod=".

Dots A program somewhat like **Wiggle**, but better in some ways because it tries to be smarter. The output style depends on the input n1 and n2. For loosely packed traces with only a few data points **Dots** plots the data as lollipops on strings, showing each data point clearly. There are also options for separately labeling each trace, omitting dead traces, making bar graphs, etc. As n1 and n2 increase **Dots** by default simplifies the output and eventually behaves almost the same as **Wiggle**. Unfortunately **Dots** does not use the axis drawing and plotting routines shared by **Wiggle**, **Contour**, **Graph**, and **Ta2vplot**, and so **Dots**' options and output plot size, position, and axes are currently incompatible with those for other plot programs.

Energy Calculate energy in running windows along fast axis of data.

Filter Filtering along the fast axis performed in the frequency domain. The filter is read from an auxiliary input file. (This old FORTRAN program does not do dynamic allocation; the input trace length is hardcoded to a limit of 4096 samples.)

Ft3d One, two, or three-dimensional Fast Fourier transform. The input and output are complex-valued (esize=8). The sign of the Fourier transform on each axis can be set from the command line or history file. **Ft3d** writes the opposite sign onto the output history file so a second application will automatically perform the inverse of the first. A sign of 0 skips transforming on that axis entirely. There are options to allow centering of the origin in the output domain, to make a graph of the output easier to understand.

Ftplot Output vplot commands to plot an input real time series and its Fourier amplitude or phase spectrum.

Gpow Raise each element of an input data file to a power, preserving sign. The power to use defaults to unity, so you probably want to specify it.

Graph The standard SEPlib program for making graphs of all sorts. The input data can be real numbers (esize=4) or coordinate pairs (esize=8). (See **Cmplx** for converting separate X and Y files into one (X,Y) file.) The output will consist of n3 graphs, each plotted with their own axes set according to the data in that graph. Each graph will have n2 traces with n1 points in each trace. (Although if the option `movie=y` is set then n3 will effectively become n1 times n2 and n2 will become one.) If esize=8 the (X,Y) pairs determine the coordinates to draw. If esize=4 the input value is taken as the Y of the pair, as you would expect, while the input d1, o1, and element number are used to calculate the associated X. **Graph** supports a bewildering variety of plotting options; almost every part of the plot can be moved about or turned on and off. The colors, symbols, line styles, line fatnesses, etc, for each trace can be specified. There are options to set the background screen color and the background graph color. Notably lacking is a mechanism for overlaying labels on the graph. (This is possible to do in a somewhat crude way by simply creating a vplot file with labels in the appropriate position using **Box**, and appending the plot containing the annotations onto the vplot file containing the graph using **Vppen**.) **Graph** sets the output clipping window hard against the limit of its graphing area, so the slightest bug in the positioning of the clipping window by the output "pen" filter may clip away extreme parts of the plot. (Single-pixel-off clipping windows is a bug that is unfortunately all too common among "pen" filters, so you will probably see this bug sooner or later.)

Halfint Half-order integration along the fast axis. (Also conjugate and inverse of this operation. The inverse is half differentiation.)

Hypint Velocity-space transformation via integration (along hyperbolas). (Also does conjugate transpose and pseudo-inverse; the conjugate transpose is hyperbola superposition.)

Imag Convert from complex (esize=8) to real (esize=4) by keeping only the imaginary component. Alternatively, pull the Y component out of (X,Y) coordinate pairs.

In Give useful information about a SEPlib header/data file pair. Tells you which of the canonical parameters are set in the header file and which are defaulted, and what values they have. Tells you the expected size of the data given the header and whether the data is actually that size. (Very useful if you want to check on the progress of a running program.) Warns if the data is all zeroes. Etcetera. Unlike most SEPlib programs, **In** understands four-dimensional datasets.

Interleave Merge two files on the 2-axis (1 axis is fast, 3 is slow) by interleaving them.

Interp2 Linear interpolation on the 2-axis. (Also can do the conjugate, transpose, and pseudoinverse of this operation.)

Lpfilt Butterworth lowpass filtering. See also **Bandpass**.

Ls Do "`ls`" on the data files associated with the given header files. (Also useful for directly referring to the data file associated with a SEPlib header file, so you can use SEPlib data files as input to non-SEPlib programs. For example instead of `Tube < Vplot.H` you could do `` tube < `Ls Vplot.H` ``.)

Merge Combine two or more SEPlib header files into one by concatenation. They can be merged along either the fast (1), intermediate (2), or slow (3) axes. By default **Merge** inserts a few elements' worth of padding at the boundaries between the input datasets.

Mv Copy (*not* move, despite the name) a SEPlib header and data file to another name. The output history file name is set on the command line; the associated data file name is determined by the usual rules (involving `out=`, `datapath=`, environment variable `DATAPATH`, a `.datapath` file, the name of the history file, etc.).

NMO Standard Normal MoveOut correction via linear interpolation; also can do the conjugate and pseudoinverse of this operation. Make sure to specify a reasonable velocity (or velocity function) for it to use. It wants NMO velocity, *not* interval velocity; by all means make sure to get the units right!

Noise Add random noise to data, either Gaussian or uniform.

Pad Append zeroes onto any of the fast (1), intermediate (2), or slow (3) axes to make a SEPlib file bigger. Defaults to bump up to the next power of two. Also can append zeroes onto the "0" axis, for example it pads esize=4 to esize=12 by appending two floating-point zeroes onto each element.

Phase Phase-shift migration and diffraction.

Pow Raise data to a power, preserving sign.

Pspen The vplot "pen" filter for postscript devices.

Radial Transform to radial traces. Also can do the conjugate and pseudoinverse of this operation.

Radnmo Transform to radial traces and do NMO at the same time. Also can do the conjugate and pseudoinverse (for constant velocity) of this operation.

Raspen The vplot "pen" filter for "the generic byte-deep raster device". You can specify the number of pixels in each dimension, the pixel size and aspect ratio, and the number of device colors available. Output is a seplib esize=1 raster file and an associated color table.

Real Convert from complex (esize=8) to real (esize=4) by keeping only the real component. Alternatively, pull the X component out of (X,Y) coordinate pairs.

Reverse Flip one or more axes of the dataset.

Rm Remove a SEPlib header file and its associated data file. Not to be confused with lower-case `rm`.

Rtoc Real (esize=4) to complex (esize=8) conversion; the imaginary part is set to zero. This function can also be done as a special case of **Pad**.

Scale Trivial data scaling program; multiply a dataset by the parameter "dscale". (Beware the special-case behavior of "dscale=1"!) **Add** can also be used to scale a dataset, but **Add** cannot use pipes which makes it somewhat less convenient. Unlike **Add**, **Scale** can be used to automatically normalize a dataset so the maximum is (plus or minus) unity. (There are several options governing how much of the dataset is to be normalized at a time.)

Spectrum Calculate average Fourier-domain amplitude spectra. Accepts real or complex input.

Spike Everyone's favorite program for creating a SEPlib dataset out of thin air. Can be used to create a dataset that is all zeroes, all ones, or all some specified constant value. It is most often used to create a dataset that is mostly zeroes except for one or more "spikes" of unit magnitude. Beware the FORTRAN-style notation: the first element is numbered 1, not 0, as it would be for most other SEPlib programs.

Stack Sum a dataset over the intermediate (2) axis.

Stoltt Stolt migration and diffraction (also does the conjugate, transpose, and inverse of Stolt migration).

Stretch A general t-squared x-squared stretching and conversion program that is usually called under one of the aliases **NMO**, **Unmo**, **Radnmo**, **Radial**, or **Stoltt**.

Sunpen The unsupported vplot "pen" filter for Sun's defunct Suntools window system.

Swab Do byte-swapping appropriate for converting native 4-byte numbers between byte-swapped and normal machines. (Reorders the bytes 1-2-3-4 to 2-1-4-3.) **Swab** is its own inverse. (Note **Swab** does not claim to convert from, say, VAX native floating point to IEEE. It just swaps bytes... although that alone will usually get you within a constant scale factor, which is usually good enough.)

Ta2vplot Input a SEPlib raster (esize=1) dataset and output vplot commands for a two-dimensional raster plot of the data. (The vplot output can be viewed on a screen using the program **Tube**, or plotted on a postscript printer using **Pspen**.) **Ta2vplot** has many, many options to specify generic plotting things such as where to position the plot on the page, how big to make the plot, which way to draw the axes, where to place tick marks and labels, etc, etc, etc. All of these parameters attempt to default reasonably. There are also several options unique to **Ta2vplot** to control things such as orientation of the raster and what color table to use (user-specified color tables are allowed). **Ta2vplot** also accepts esize=3 input which it interprets as (R,G,B) byte-deep triples. This option is most useful for plotting raster that is meant to have a "multidimensional" color table. (Warning: the esize=3 option is very slow if used with the standard linear Movie-style color tables, although it does work. esize=3 input is really meant to be used with "RGB" color tables.) You may find the utility programs **Window** and **Reverse** useful for pre-processing data to be plotted with **Ta2vplot**. See also **Vppen** and **Box** for a crude way of adding annotation.

Taplot A synonym for **Byte**.

Tekpen The vplot "pen" filter for Tektronix 4010 emulators.

Thplot Pseudo three-dimensional hidden-line plotting program.

Tpow Multiply each element of a seismogram by time raised to a given power. (The dimension associated with the fast (1) axis is assumed to be time.) A power of 2 often seems to be a good one for balancing the early and late time of a seismogram without the loss of amplitude information and other annoying problems associated with automatic AGC. (By default **Tpow** attempts to find a good default value for the time-power parameter automatically. Unfortunately the routine that does this has been broken by a careless programmer and currently always core dumps; for now (mid 1992) you must specify the tpow yourself.)

Transp Transpose two of the three dimensions of a SEPlib data cube. (An option lets you select which two.)

Triconv Convolve the input with a triangle filter along both the fast (1) and intermediate (2) axes. You should specify the width of the triangle; the default is a spike (and so the default output is just the same as the input).

Tube The generic vplot "pen" filter for screen devices. (In reality it's just a `csh` script that calls the appropriate pen filter for your device by searching case-by-case for the value of your `TERM` environment variable in a switch.) The "pen" manual page gives a canonical list of device-independent **Tube** (and **tube**) options. Additional options may apply, depending on which pen filter **Tube** calls.

Unmo The pseudoinverse of the program **NMO** (standard Normal MoveOut correction via linear interpolation). Make sure to specify a reasonable velocity for it to use!

Velan Velocity analysis of common-midpoint gathers.

Vppen The vplot "pen" filter for the virtual vplot device. This program is widely used to do various utility sorts of transformations on vplot files. It can be used to automatically center and size vplot files, to report statistics, rotate, scale, shift, fatten, thin, scale text, scale dash patterns, etc, etc, etc. It can also be used to combine multiple vplot files in various ways. For example it can overlay one vplot file on top of another, combine them as successive frames in a single file, or most usefully combine multiple plot frames into one superplot by arranging the individual subplots in a grid. As you can guess, **Vppen** has a frightening number of options; they are enumerated with some explanation in both the **vppen** and **pen** man pages. (These man pages are at least current and complete, though some people have claimed that they are just too dense and wordy to be usable.) See also **vppen** (lower case), **plas**, **pldb**, and the vplot manual pages (vplot, pen, vplottext, vplotraster, libvplot).

Vslan Does slant stacking, transpose slant stacking, and the least-squares inverse of slant stacking and transpose slant stacking. The slant stack domain can be in either terms of ray-parameter "p" and frequency "ω", or ray-parameter "p" and reduced time "tau".

Wavelet Wavelet generation program; used by modeling programs as input to provide a source time function. Beware the dreaded "echo" bug that can occur if the output time duration is too much longer than the duration of the wavelet. (After the wavelet is zero and should remain forevermore zero, a new scaled-down copy of the wavelet fires off again.)

Wiggle Inputs a real-valued (esize=4) dataset and outputs vplot commands for a geophysical-style wiggle plot of the data. (The vplot output can be viewed on a screen using the program **Tube**, or plotted on a postscript printer using **Pspen**.) **Wiggle** has many, many options to specify how closely to pack the plot traces, how much to overlap the wiggles, whether to fill under the positive excursions, where to position the plot on the page, how big to make the plot, which way to draw the axes, where to place tick marks and labels, etc, etc, etc. All of these parameters attempt to default reasonably, although **Wiggle** can behave stupidly if the input data is constant or consists mostly of zeroes. In such cases you may have to set the clip value yourself. You may find the utility programs **Window** and **Reverse** useful for pre-processing data to be plotted with **Wiggle**. See also **Vppen** and **Box** for a crude way of adding annotation. Also see the related program **Dots**, which is superior to **Wiggle** for some applications.

Window Window out a portion of a dataset. This includes dropping elements from the beginning or end of an axis (or both); it also includes decimation. The beginning and ending elements can be specified in terms of physical units (dependent on the "o" and "d" parameters of the axis) or in terms of element number. Note **Window**, like most SEPlib programs, uses "C" style numbering: the first element is numbered 0, not 1 as may seem natural to you if you grew up on FORTRAN. Beware the special-case behavior of **Window** if one of the axes is reduced to length 1: it automatically does a transpose to shift unit-length axes to the end. This behavior is usually desirable, but can be an unpleasant surprise if unexpected. (If not desired there is an option to turn it off.) Unlike most SEPlib programs, **Window** understands four-dimensional datasets.

X11movie An interactive "movie" program for X11 windows. Input should be byte-deep (esize=1) raster. It associates a color table with this raster and rapidly animates it frame by frame (stepping down the slow (3) axis) on the X11 display. A variety of other animation formats are available as well. There is no attempt at such niceties as axes, although there is a primitive facility for frame-dependent labels. Note you have to put the mouse in the **X11movie** window to get the right colors, but you have to move the mouse out of the window for **X11movie** to accept your interactive keyboard commands. This program is great when it works, the fastest animator around. It also works surprisingly well over slow and distant network connections that cause most other interactive programs to crash. If **X11movie** should leave your screen "hung" (happens fairly often, unfortunately) you will find that the "`reset`" command will almost always fix it. Note: while **X11movie** is included among the distributed SEPlib software sources, by default it is neither compiled nor installed.

X11pen An old vplot "pen" filter for X11. **Xtpen** is preferred.

Xtpen The vplot "pen" filter for the MIT X toolkit. One of the snazziest vplot filters around, but still under active development (as of mid 1992).

Xvpen The vplot "pen" filter for Sun's Xview. **Xtpen** is preferred.

Useful non-SEPlib programs

Note that the names of these programs all begin with a lower-case letter.

plas **plas** reads in a human-readable ASCII version of the vplot graphics language and writes out standard binary vplot (the same stuff written into the output SEPlib data file by SEPlib graphics programs like **Graph**, **Wiggle**, **Contour**, **Ta2vplot**, **Dots**, etc...). **plas** is the inverse of **pldb**. **plas** is primarily used to convert the output of **pldb** back into regular vplot, but it can also be used to generate trivial vplot files from scratch. To do this, use your favorite editor (or even "`cat`" or "`echo`") to create an ASCII human-readable version of a vplot file, then use **plas** to turn it into a genuine (binary) vplot file. You can find the documentation for both the ASCII and binary vplot file formats in the vplot man page, although it is probably easier to learn by using **pldb** to generate a few examples from known files. (Note **plas** is not a SEPlib program: it does not begin with a capital letter. It does not read in or write out SEPlib history files!)

pldb **pldb** reads a vplot file from standard input (not a SEPlib header file pointing to a vplot data file, but a raw vplot data file) and writes out a human-readable and editable ASCII version. By default the units are in integer "vplot units", 600 to the inch. You may find the options of **pldb** to express everything in units of inches or centimeters make the output ASCII vplot file easier to work with. **pldb** is often used to perform trivial editing operations on a vplot graphics file. For example, if you want to change the color of some object and can't easily regenerate the plot from scratch, you can convert the binary vplot to ASCII using **pldb**, edit one or two lines so the color changes and changes back again at the right times, and then use **plas** to turn the file back into standard binary vplot. (Note **pldb** is not a SEPlib program: it does not begin with a capital letter. It does not read in or write out SEPlib history files.) See also **plas** (the inverse of **pldb**).

pspen The non-SEPlib version of **Pspen**; takes straight vplot files as input instead of SEPlib history files that point to vplot files.

tube The non-SEPlib version of **Tube**; takes straight vplot files as input instead of SEPlib history files that point to vplot files.

vp_OverUnderAniso Stack two or more vplot plots one over the other, with the first on bottom and the last on top. Stretch the files anisotropically to "fill the screen". The multiple plots to be stacked can be spread through multiple input files, or can be multiple frames of plots (separated by erases) within a single vplot file.

vp_OverUnderIso Stack two or more vplot plots one over the other, with the first on bottom and the last on top. Do not stretch the files to "fill the screen"; preserve aspect ratios. The multiple plots to be stacked can be spread through multiple input files, or can be multiple frames of plots (separated by erases) within a single vplot file.

vp_SideBySideAniso Like **vp_OverUnderAniso**, but stacks plots side by side from left to right.

vp_SideBySideIso Like **vp_OverUnderIso**, but stacks plots side by side from left to right.

vp_Unrotate Unrotate old-style plots. (In former days SEPlib programs put the plot origin at the upper-left corner of the screen, with the X axis going down and the Y axis going left to right across the page.) Hopefully you will never have a need for this utility, but the past never completely dies.

vppen The non-SEPlib version of **Vppen**; takes straight vplot files as input instead of SEPlib history files that point to vplot files. The user interface to **vppen** can be intimidating to the new user. Those intimidated may find the shells **vp_OverUnderAniso**, **vp_OverUnderIso**, **vp_SideBySideAniso**, **vp_SideBySideIso**, and **vp_Unrotate** useful. These shells use **vppen** to do their work but have a much simpler user interface (since they aren't trying do to everything under the sun in one program). See also **plas**, **pldb**, **tube**, and the vplot manual pages.