A Course in Geophysical Image Processing with Seismic Unix: GPGN 461/561 Lab Fall 2015

Instructor: John Stockwell
Research Associate
Center for Wave Phenomena

January 12, 2016

Contents

List of Figures

Preface

I started writing these notes in 2005 to aid in the teaching of a seismic processing lab that is part of the courses Seismic Processing GPGN452 (later redesignated GPGN461) and Advanced Seismic Methods (GPGN561) in the Department of Geophysics, Colorado School of Mines, Golden, CO.

In October of 2005, Geophysics Department chairman Terry Young asked me if I would be willing to help teach the Seismic Processing Lab. This was the year following Ken Larner's retirement. Terry was teaching the lecture, but decided that the students should have a practical problem to work on. The choice was between data collected in the Geophysics Field Camp the previous summer, or the an industry dataset that was acquired near the Viking Graben in the North Sea. The latter dataset was brought by Terry from Carnegie Mellon University. We chose the latter, and decided that the students should produce as their final project a poster presentation similar to those seen at the SEG annual meeting. Terry seemed to think that we could just hand the students the SU User's Manual and the data, and let them have at it. I felt that more needed to be done to instruct students in the subject of seismic processing while simultaneously introducing them to the topics of navigating the Unix operating system, performing some simple shell language programming, and of course, using Seismic Unx.

In the years that have elapsed my understanding of the subject of seismic processing has continued to grow. In each successive semester I have gathered more examples and figured out how to apply more types of processing techniques to the data.

My vision of the material is that we are replicating the seismic processors' base experience, such as a professional might have obtained in the petroleum industry in the late 1970s. The idea is not to *train* students in a particular routine of processing, but to teach them how to think like geophysicists. Because seismic processing techniques are not exclusively used on petroleum industry data, the title of "Geophysical Image Processing" was chosen.

Chapter 1

Seismic Processing Lab- Preliminary issues

1.1 Motivation for the lab

In the lecture portion of the course GPGN452/561 (now GPGN461/561) (Advanced Seismic Methods/Seismic Processing) the student is given a word, picture, and chalkboard introduction of the process of seismic data acquisition and the application of a myriad of processing steps for converting raw seismic data into a scientifically useful picture of the earth's subsurface.

This lab is designed to provide students with practical hands-on experience in the *reality* of applying seismic processing techniques to synthetic and real data. The course, however, is not a "training course in seismic processing," as one might get in an industrial setting. Rather than training a student to use a particular collection of software tools, we believe that it is better that the student cultivate a broader understanding of the subject of seismic processing. We seek also to help students develop some practical skills that will serve them in a general way, even if they do not go into the field of oil and gas exploration and development.

Consequently, we make use of freely available open-source software (the Seismic Unix package) running on small-scale hardware (Linux-based PCs). Students are also encouraged to install the SU software on their own personal (Linux or Mac) PCs, so that they may work (and play) with the data and with the codes, at their leisure.

Given the limited scale of our available hardware and time, our goal is modest, to introduce students to seismic data processing through a 2D single-component processing application.

The intended range of experience is approximately that which a seismic processor of mid to late 1970s might have experienced on a vastly slower, more expensive, and more difficult to use processing platform.

Our technology is different from that of the 1970s geophysicist. This section is included to help familiarize the student with that technology.

1.2 Unix and Unix-like operating systems

The Unix operating system (as well as any other Unix-like operating system, which includes the various forms of Linux, UBUNTU, Free BSD Unix, and Mac OS X) is commonly used in the exploration seismic community. Consequently, learning aspects of this operating system is time well spent. Many users may have grown up with a "point and click" environment (or a "there is an app for that" environment), where a given program is run via a graphical user interface (GUI) featuring menus and assorted windows. Certainly there are such software applications in the world of commercial seismic processing, but none of these are inexpensive, and none give the user access to the source code of the application.

There is also an "expert user" level of work where such GUI-driven tools do not exist and programs are run from the *commandline* of a *terminal window* or are executed as part of a processing sequence in a *shell script*.

In this course we use the open source CWP/SU:Seismic Unix (called simply Seismic Unix or SU) seismic processing and research environment. This software collection was developed largely at the Colorado School of Mines (CSM) at the Center for Wave Phenomena (CWP), with contributions from users all around the world. The SU software package is designed to run under any Unix or Unix-like operating system, and is available as full source code. Students are free to install Linux and SU on their PCs (or use Unix-like alternatives) and thus have the software as well as the data provided for the course for home use, during, and beyond the time of the course.

The datasets are also open. The major dataset that we will use in the course was put in the public domain by Mobil corporation in the early 1990s. The student may keep both the data and the software for his/her own continuing education after the course is finished.

1.2.1 Steep learning curve

The disadvantage that most beginning Unix users face is a steep learning curve owing to the myriad of commands that comprise Unix and other Unix-like operating systems. The advantages of software portability and flexibility of applications, as well as superior networking capability, however, makes Unix more attractive to industry than Microsoft-based systems for these expert level applications. While a user in an industrial environment may have a Microsoft-based PC on his or her desk, the more computationally intensive processing work is done on a Unix-based system. The largest of these are clusters composed of multi-core, multiprocessor PC systems. It is not uncommon these days for such systems to have several thousand "cores," which is to say subprocessors. Thus, massive parallelism is available in the industry environment.

Because a course in seismic processing is of broad interest and may draw students with varied backgrounds and varied familiarity with computing systems, we begin with the basics. The reader familiar with these topics may skip to the next chapter.

1.3 Logging in

As with most computer systems, there is a prompt, usually containing the word "login" or the word "username" that indicates the place where the user types his or her login name. The user is then prompted for a password. Once on the system, the user either has a windowed user interface as the default, or initiates such an interface with a command, such as **startx** in some installations of Linux.

(If you are unable to login on the laboratory machines, you likely need to set your CSM MultiPass password. For this you will need your Colorado School of Mines E-Key, which you obtained when you registered at the school.)

1.4 What is a Shell?

Some of the difficult and confusing aspects of Unix and Unix-like operating systems are encountered at the very beginning of using the system. The first of these is the notion of a *shell*. Unix is an hierarchical operating system that runs a program called the *kernel* that is is the heart of the operating system. Everything else consists of programs that are run by the kernel and which give the user access to the kernel and thus to the hardware of the machine.

The program that allows the user to interface with the computer is called the "working shell." The basic level of shell on all Unix systems is called **sh**, the *Bourne shell*. Under Linux-based systems, this shell is actually an open-source rewritten version called **bash** (the Bourne again shell), but it has an alias that makes it appear to be the same as the **sh** that is found on all other Unix and Unix-like systems.

The common working shell environment that a user is usually set up to login in under may be **csh** (the C-shell), **tcsh** (the T-shell, which is a non proprietary version of **csh**, **ksh** (the Korn shell, which is proprietary), **zsh** which is an open source version of Korn shell, or **bash**, which is an open source version of the Bourne shell.

On Linux and Mac OS X systems bash is the default shell environment.

The user has access to an application called *terminal* in the graphical user environment, that when launched (usually by double clicking on an icon that looks like a small video monitor) invokes a window called a *terminal window*. (The word"terminal" harks back to an earlier day, when a physical device called a "terminal," consisting of a screen and keyboard (but no mouse), constituted the users' interface to the computer.) It is at the prompt on the terminal window that the user has access to a *commandline* where Unix commands are typed.

Most "commands" on Unix-like systems are not built in commands in the shell, but are actually programs that are run under the users' working shell environment. The shell commandline prompt is asking the user to input the name of an executable program. That program may be a system command, such as a directory (folder) listing, or it may be a program written by a third party, or by the user him/herself.

1.5 The working environment

In the Unix world all filenames, program names, shells, and directory names, as well as passwords are case sensitive in their input, so please be careful in running the examples that follow.

If the user types:

```
$ cd                                    <--- change directory with no argument
^                                           takes the user to his/her home
(don't type the dollar sign)                 directory
```

In these notes, the $ symbol will represent the commandline prompt. The user does not type this $. Because there are a large variety of possible prompt characters, or strings of characters that people use for the prompt, we show here only the dollar sign $ as a generic commmandline prompt. On your system it might be a %, a >, or some combination of these with the computer name and or the working directory and/or the commandline number.

```
$ echo $SHELL                           <--- returns the value of the users'
       ^                                      working shell environment
     type this dollar sign
```

The command **echo $SHELL** tells your working shell to return the value that denotes your working shell environment. In English this command might be translated as "print the value of the variable SHELL". In this context the dollar sign $ in front of SHELL should be translated as "value of". Thus, "echo value of SHELL".

Common possible shells are

```
/bin/sh                         <--- the Bourne Shell
/bin/bash                       <--- the Bourne again Shell
/bin/ksh                        <--- K-shell
/bin/zsh                        <--- Z-shell
/bin/csh                        <--- C-shell
/bin/tcsh                       <--- T-shell.
```

The environments **sh**, **bash**, **ksh**, and **zsh** are similar. We will call these the "sh-family." The environments **csh** and **tcsh** are similar to each other, but have many differences from the sh-family. We refer to **csh** and **tcsh** as the csh-family.

Again, on Linux and Mac OX systems **/bin/bash** is usually the default working shell environment.

1.6 Setting the working environment

Each of these programs have a specific syntax, which can be quite complicated. Each is a language that allows the user to write programs called "shell scripts." Thus Unix-like systems have scripting languages as their basic interface environment. This endows Unix-like operating systems with vastly more flexibility and power than other operating systems you may have encountered as point and click environments. Even those environments may have a shell command structure that the user is protected from by a windowed environment.

Wny have such a structure? The answer is that "point and click is not enough."The expert user needs to be able provide more complicated instructions to the computer, and the shell provides the languge of those instructions.

With more flexibility and power, there comes more complexity. It is possible to perform many configuration changes and personalizations to your working environment, which can enhance your user experience. For these notes we concentrate only on enough of these to allow you to work effectively on the examples in the text.

1.7 Choice of editor

To edit files on a Unix-like system the user must adopt an editor. The traditional Unix editor is **vi** or one of its non-proprietary clones **vim** (**vi**-improved), **gvim**, or **elvis**. The **vi** environment has a steep learning curve making it often unpopular among beginners. If a person is envisioning working on Unix-like systems a lot, then taking the time to learn **vi** is also time well spent. The **vi** editor is the only editor that is guaranteed to be on all Unix-like systems. All other editors are third-party items that may have to be added on some systems, sometimes with difficulty.

Similarly there is an editor called **emacs** that is popular among many users, largely because it is possible to write programs in the LISP language and implement these within the **emacs** environment. There is also a steep learning curve for this language. There is often substantial configuration required to get **emacs** working in the way the user desires.

A third editor is called **pico**, which comes with a mailer called "**pine**." **Pico** is easy to learn to use, fully menued, and runs in a terminal window.

The fourth class of editor consists of the "screen editors." Popular screen editors include **xedit**, **nedit**, and **gedit**. There is a windowed interfaced version of emacs called **xemacs** that is similar to the first two editors. These are all easy to learn and to use.

Not all editors are the best to use. The user may find that invisible characters are introduced by some editors, and that there may be issues regarding how wrapped lines are handled that may cause problems for some applications. These issues are another incentive for an expert user, such as a Unix system administrator to prefer **vi** over other more intuitive editors.

The choice of editor is often a highly personal one depending on what the user is familiar with, or is trying to accomplish. Any of the above mentioned editors, or similar

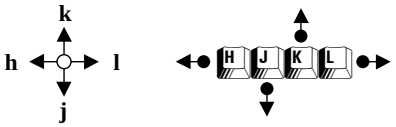
Vi Quick Reference

http://www.sfu.ca/~yzhang/linux

MOVEMENT

(**lines** - ends at <CR>; **sentence** - ends at puncuation-space; **section** - ends at <EOF>)

By Character



By Line

***n*G**	to line *n*
0, $	first, last position on line
^ or _	first non-whitespace char on line
+, -	first character on next, prev line

By Screen

^F, ^B	scroll foward, back one full screen
^D, ^U	scroll forward, back half a screen
^E, ^Y	show one more line at bottom, top
L	go to the bottom of the screen
z↵	position line with cursor at top
z.	position line with cursor at middle
z-	position line with cursor at

Marking Position on Screen

m*p*	mark current position as *p* (a..z)
`*p*	move to mark position *p*
'*p*	move to first non-whitespace on line w/mark *p*

Miscellaneous Movement

f*m*	forward to character *m*
F*m*	backward to character *m*
t*m*	forward to character before *m*
T*m*	backward to character after *m*
w	move to next word (stops at puncuation)
W	move to next word (skips punctuation)
b	move to previous word (stops at punctuation)
B	move to previous word (skips punctuation)
e	end of word (puncuation not part of word)
E	end of word (punctuation part of word)
), (	next, previous sentence
]], [[	next, previous section
}, {	next, previous paragraph
%	goto matching parenthesis () {} []

EDITING TEXT

Entering Text

a	append after cursor
A or $a	append at end of line
i	insert before cursor
I or _i	insert at beginning of line
o	open line below cursor
O	open line above cursor
c*m*	change text (*m* is movement)

Cut, Copy, Paste (Working w/Buffers)

d*m*	delete (*m* is movement)
dd	delete line
D or d$	delete to end of line
x	delete char under cursor
X	delete char before cursor
y*m*	yank to buffer (*m* is movement)
yy or Y	yank to buffer current line
p	paste from buffer after cursor
P	paste from buffer before cursor
"*b*dd	cut line into named buffer *b* (a..z)
"*b*p	paste from named buffer *b*

Searching and Replacing

/*w*	search forward for *w*
?*w*	search backward for *w*
/*w*/+*n*	search forward for *w* and move down *n* lines
n	repeat search (forward)
N	repeat search (backward)

:s/*old*/*new*	replace next occurence of *old* with *new*
:s/*old*/*new*/g	replace all occurences on the line
:*x*,*y*s/*old*/*new*/g	replace all ocurrences from line *x* to *y*
:%s/*old*/*new*/g	replace all occurrences in file
:%s/*old*/*new*/gc	same as above, with confirmation

Miscellaneous

n*>*m	indent *n* lines (*m* is movement)
n*<*m	un-indent left *n* lines (*m* is movement)
.	repeat last command
U	undo changes on current line
u	undo last command
J	join end of line with next line (at <cr>)
:r*f*	insert text from external file *f*
^G	show status

Figure 1.1: A quick reference for the **vi** editor.

third party editors likely are sufficient for the purposes of this course.

For this class, if you are not already familiar with **vi** or some other editor, I would recommend using **gedit**.

1.8 The Unix directory structure

As with other computing systems, data and programs are contained in "files" and "files" are contained in "folders." In Unix and all Unix-like environments "folders" are called "directories."

The structure of directories in Unix is that of an upside down tree, with its root at the top, and its branches—subdirectories and the files they contain—extending downward. The root directory is called "/" (pronounced "slash").

While there exist graphical browsers on most Unix-like operating systems, it is more efficient for users working on the commandline of a terminal windows to use a few simple commands to view and navigate the contents of the directory structure. Some of these commands are **pwd** (print working directory), **ls** (list contents), and **cd** (change directory).

Locating yourself on the system

If you type:

```
$ cd
$ pwd
$ ls
```

You will see your current working directory location, which is your called your "home directory." You should see something like

```
$ pwd
/home/yourusername
```

where "yourusername" is your username on the system. Other users likely have their home directories in

```
/home
```

or something similar depending on how your system administrator has set things up. The command **ls** (which is short for "list") will show you the contents of your home directory, which may consist of files or other subdirectories.

The codes for Seismic Unix are installed in some system directory path. We will assume that all of the CWP/SU: Seismic Unix codes are located in

```
/usr/local/cwp
```

This denotes a directory "cwp," which is the sub directory of a directory called "local," which is in turn is a subdirectory of the directory "usr," that itself is a sub directory of slash.

It is worthwile for the user to spend some time learning the layout of his or her directories. There is a command called

```
$ df
```

which shows the hardware devices that constitute the available storage on the users' machine. A typical output from typing "df"

```
$ df -h

Filesystem                Size  Used Avail Use% Mounted on
/dev/sda1                 286G   19G  253G   7% /
none                      4.0K     0  4.0K   0% /sys/fs/cgroup
udev                      3.9G  4.0K  3.9G   1% /dev
tmpfs                     795M  1.1M  794M   1% /run
none                      5.0M     0  5.0M   0% /run/lock
none                      3.9G  488K  3.9G   1% /run/shm
none                      100M   44K  100M   1% /run/user
fermat:/u                 2.0T  1.3T  664G  66% /u
fermat:/gpfc              3.0T  1.1T  1.8T  38% /gpfc
isengard:/class            15G  562M   14G   4% /class
isengard:/usr/local/cwp    20G   17G  2.2G  89% /usr/local/cwp
isengard:/scratch         378G  270G   90G  76% /scratch
isengard:/data             99G   52G   42G  56% /data
isengard:/data/cwpscratch  30G  6.9G   22G  25% /data/cwpscratch
```

Note items in the far left column. Those whose names that begin with "dev" are hardware **devices** on the specific computer. The items that begin with a machine name, in this case "isengard.mines.edu" exist physically on another machine (named "isengard"), but are **remotely mounted** as to appear to be on this machine. The second column from the left shows the total space on the device, the third column shows the amount of space used, while the fourth shows the amount available, the fifth column shows the usage as a percentage of space used. Finally the far right column shows the directory where these devices are mounted.

In Unix-like environments, devices are mounted in such a way that they appear to be files or directories. Under Unix-like operating systems, the user sees only a directory tree, and not individual hardware devices.

If you try editing files in some of these other directories you will find that you likely may not have permission to read, write, or modify the contents of many those directories. Unix is a multi-user environment, meaning that from an early day, the notion of

protecting users from each other and from themselves, as well as protecting the operating system from the users, has been a priority.

In none of these examples have we used a browser, yet there are browsers available on most Unix systems. There is no fundamental problem with using a browser, with the exception that you have to take your hands off the keyboard to use the mouse. The browser will not tell you where you are located within a terminal window. If you must use a browser, use "column view" rather than "icon view" as we will have many levels of nested directories to navigate.

1.9 Scratch and Data directories

Directories with names such as "scratch" and "data" are often provided with user write permission so that users may keep temporary files and data files out of their home directories. Like "scratch paper" a scratch directory is usually for temporary file storage, and is NOT BACKED UP! Indeed, on any computer system there may be other unbacked up directories. You need to be aware of which parts of your computer system are backed up and which are not. Because there are no backups on scratch directories, it is important for the user to purchase a USB device to back up his or her items from the scratch areas.

Some directories may be physically located on the specific machine were you are seated and may not be visible on other machines. Because the redundancy of backups require extra storage, most system administrators restrict the amount of backed up space to a relatively small area of a computer system. To restrict user access, quotas may be imposed that will prevent users from using so much space that a single user could fill up a disk. However, in scratch areas there usually are no such restrictions, so it is preferable to work in these directories, and save only really important materials in your home directory.

Users should be aware, that administration of scratch directories may not be user friendly. Using up all of the space on a partition may have dire consequences, in that the administrator may simply remove items that are too big, or have a policy of removing items that have not been accessed over a certain period of time. A system administrator may also set up an automated "grim file reaper" to automatically delete materials that have not been accessed after a period of time. **Because files are not always automatically backed up, and because hardware failures are possible on any system, it is a good idea for the user to purchase USB storage media and get in the habit of making personal backups on a regular basis.** A less hostile mode of management is to institute **quotas** to prevent single users from hogging the available scratch space.

You may see a scratch directory on any of the machines in your lab, but these are different directories, each located on a different hard drive. This can lead to confusion as a user may copy stuff into a scratch area on one day, and then work on a different computer on a different day, thinking that their stuff has been removed.

The availability and use of scratch directories is important, because each user has a quota that limits the amount of space that he or she may use in his/her home directory.

On systems where a scratch directory is provided, that also has write permission, the user may create his/her personal work area via

```
$ cd /scratch
$ mkdir yourusername                    <--- here "yourusername" is the
                                             your user name on the system
```

Unless otherwise stated, this text will assume that you are conducting further operations in your personal scratch work area.

For our system, the scratch directory that we will work in is gpfc so your instructions are to

```
$ cd /gpfc
$ mkdir yourusername                    <--- here "yourusername" is the
                                             your user name on the system
```

The directory gpfcyourusername will be your preferred scratch or working area.

1.10 Shell environment variables and path

The working shell is a program that has a configuration that gives the user access to executable files on the system. Recall that echoing the value of the SHELL variable

```
$ echo $SHELL                           <--- returns the value of the users'
                                             working shell environment
```

tells you what shell program is your working shell environment. There are other environmental variables other than SHELL. Again, note that if this command returns one of the values

```
/bin/sh
/bin/ksh
/bin/bash
/bin/zsh
```

then you are working in the SH-family and need to follow instructions for working with that type of environment. If, on the other hand, the **echo $SHELL** command returns one of the values

```
/bin/csh
/bin/tcsh
```

then you are working in the CSH-family and need to follow the alternate series of instructions given.

In the modern world of Linux, it is quite common for the default shell to be something called binbash an open-source version of binsh.

1.10.1 The path or PATH

Another important variable is the "path" or "PATH". The value path variable tells where the working shell to look for executable files. Usually, executables are stored in a sub directory "bin" of some directory. Because there may be many software packages installed on a system, there may be many such locations. If an executable file is not on the users' path, then the shell cannot see it.

To find out what paths you can access, which is to say, which executables your shell can see, type

```
$ echo $path
```

or

```
$ echo $PATH
```

The result will be a listing, separated by colons ":" of paths or by spaces " " to executable programs.

1.10.2 The CWPROOT variable

The variable PATH is important, but SHELL and PATH are not the only possible environment variable. Often programmers will use an environment variable to give a users' shell access to some attribute or information regarding a specific piece of software. This is done because sometimes software packages are of restricted interest.

For SU the path CWPROOT is necessary for running the SU suite of programs. We need to set this environment variable, and to put the suite of Seismic Unix programs on the users' path.

1.11 Shell configuration files

Because the users' shell has as an attribute a natural programming language, many configurations of the shell environment are possible. To find the configuration files for your operating system, type

```
$ ls -a                                  <--- show directory listing of all
                                              files and sub directories
$ pwd                                    <--- print working directory
```

then the user will see a number of files whose names begin with a dot ".".

1.12 Setting up the working environment

One of the most difficult and confusing aspects of working on Unix-like systems is encountered right at the beginning. This is the problem of setting up user's personal environment. There are two sets of instructions given here. One for the CSH-family of shells and the other for the SH-family.

1.12.1 The CSH-family

Each of the shell types returned by $SHELL has a different configuration file. For the csh-family (tcsh,csh), the configuration files are ".cshrc" and ".login". To configure the shell, edit the file .cshrc. Also, the "path" variable is *lower case*.

You will likely find a line beginning with

```
set path=(
```

with entries something like

```
set path=( /lib ~/bin /usr/bin/X11 /usr/local/bin /bin
                              /usr/bin  . /usr/local/bin /usr/sbin )
```

Suppose that the Seismic Unix package is installed in the directory

```
/usr/local/cwp
```

on your system.

Then we would add one line above to set the "CWPROOT" environment variable. And one line below to define the user's "path"

```
setenv CWPROOT /usr/local/cwp

set path=( /lib ~/bin /usr/bin/X11 /usr/local/bin /bin
                              /usr/bin  . /usr/local/bin /usr/sbin )

set path=( $path $CWPROOT/bin )
```

Save the file, and log out and log back in. You will need to log out completely from the system, not just from particular terminal windows.

When you log back in, and pull up a terminal window, typing

```
$ echo $CWPROOT
```

will yield

```
/usr/local/cwp
```

and

```
$ echo $PATH
```

will yield

```
/lib /u/yourusername/bin /usr/bin/X11 /usr/local/bin /bin
                        /usr/bin . /usr/local/bin /usr/sbin /usr/local/cwp/bin
```

1.12.2 The SH-family

The process is similar for the SH-family of shells. The file of interest has a name of the form ".profile," .bashrc," and the ".bash_profile." The ".bash_profile" is read once by the shell, but the ".bashrc" file is read everytime a window is opened or a shell is invoked. (Or vice versa, depending on the system. Mac OS X seems to have a strange convention.) Thus, what is set here influences the users complete environment. The default form of this file may show a path line similar to

```
PATH=$PATH:$HOME/bin:.:/usr/local/bin
```

which should be edited to read

```
export CWPROOT=/usr/local/cwp
PATH=$PATH:$HOME/bin:/usr/local/bin:$CWPROOT/bin:.
```

The important part of the path is to add the

```
:$CWPROOT/bin:.
```

on the end of the PATH line, no matter what it says.

The user then logs out and logs back in for the changes to take effect. In each case, the PATH and CWPROOT variables are necessary to be set for the users' working shell environment to find the executables of Seismic Unix.

1.13 Unix help mechanism- Unix man pages

Every program on a Unix or Unix-like system has a system manual page, called a **man page**, that gives a terse description of its usage. For example, type:

```
$ man ls
$ man cd
$ man df
$ man sh
$ man bash
$ man csh
```

to see what the system says about these commands. For example:

```
$ man ls
```

```
LS(1)                          User Commands                          LS(1)



NAME
       ls - list directory contents
```

```
SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).
       Sort entries alphabetically if none of -cftuvSUX nor --sort.

       Mandatory arguments to long options are  mandatory  for  short  options
       too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..
--MORE-
```

The item at the bottom that says –**MORE**– indicates that the page continues. To see the rest of the man page for **ls** is viewed by hitting the space bar. View the Unix man page for each of the Unix commands you have used so far.

Most Unix commands have options such as the **ls -a** which allowed you to see files beginning with dot "." or **ls -l** which shows the "long listing" of programs. Remember to view the Unix man pages of each new Unix command as it is presented.

References

Sobell, M. (2010), "A practical guide to Linux commands, editors, and shell programming" Pearson Education Inc., Boston, MA.

Chapter 2

Lab Activity #1 - Getting started with Unix and SU

Any program that has executable permissions and which appears on the users' PATH may be run by simply typing its name on the commandline. For example, if you have set your path correctly, you should be able to do the following

```
$ suplane | suxwigb  &
                     ^ this symbol, the ampersand, indicates that
                       the program is being run in background
          ^ the "pipe" symbol
```

The commandline itself is the interactive prompt that the shell program is providing so that you can supply input. The proper input for a commandline is an executable file, which may be a compiled program or a Unix shell script. The command prompt is saying, "Type program name here."

Try running this command with and without the ampersand &. If you run

```
$ suplane | suxwigb
```

The plot comes up, but you have to kill the plot window before you can get your commandline back, whereas

```
$ suplane | suxwigb  &
```

allows you to have the plot on the screen, and have the commandline.

To make the plot better we may add some axis labeling:

```
$ suplane | suxwigb title="suplane test pattern"
                            label1="time (s)" label2="trace number" &

                            ^ Here the command is broken across a line
                              so it will fit this page of this book.
                              On your screen it would be typed as one
                              long line.
```

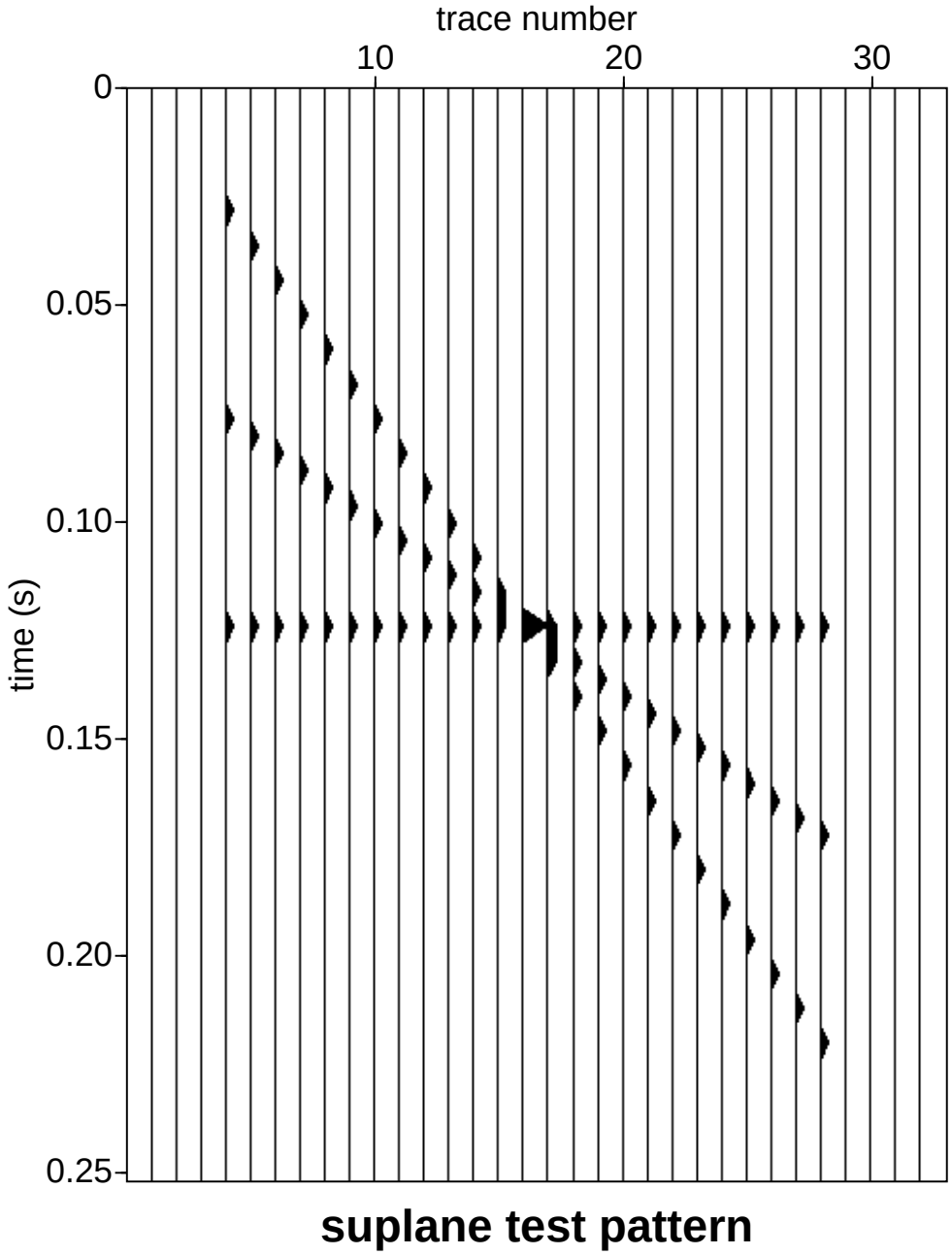


Figure 2.1: The **suplane test pattern.**

to see a test pattern consisting of three intersecting lines in the form of seismic traces. The data consist of seismic traces with only single values that are nonzero. This is *variable area* display in which each place where the trace is positive valued is shaded black. See Figure 2.1.

Equivalently, you should see the same output by typing

```
$ suplane > junk.su
$ suxwigb < junk.su  title="suplane test pattern"
                                label1="time (s)" label2="trace number" &
```

Finally, we often need to have graphical output that can be imported into documents. In SU we have graphics programs that write output in the PostScript language

```
$ supswigb < junk.su title="suplane test pattern"
                       label1="time (s)" label2="trace number" > suplane.eps
```

2.1 Pipe |, redirect in < , redirect out >, and run in background &

In the commands in the last section we used three symbols that allow files and programs to send data to each other and to send data between programs. The vertical bar | is called a "pipe" on all Unix-like systems. Output sent to standard out may be piped from one program to another program as was done in the example of

```
$ suplane | suxwigb  &
```

which, in English may be translated as "run **suplane** (pipe output to the program) **suxwigb** where the & says (run all commands on this line in background)." The pipe | is a memory buffer with a "read from standard input" for an input and a "write to standard output" for an output. You can think of this as a kind of plumbing. A stream of data, much like a stream of water is flowing from the program **suplane** to the program **suxwigb**.

The "greater than" sign > is called "redirect out" and

```
$ suplane > junk.su
```

says "run **suplane** (writing output to the file) junk.su. The > is a buffer which reads from standard input and writes to the file whose name is supplied to the right of the symbol. Think of this as data pouring out of the program **suplane** into the file junk.su. The file junk.su then, is like a bucket holding the data.

The "less than" sign < is called "redirect in" and

```
$ suxwigb < junk.su   &
```

says "run **suxwigb** (reading the input from the file) junk.su (run in background).

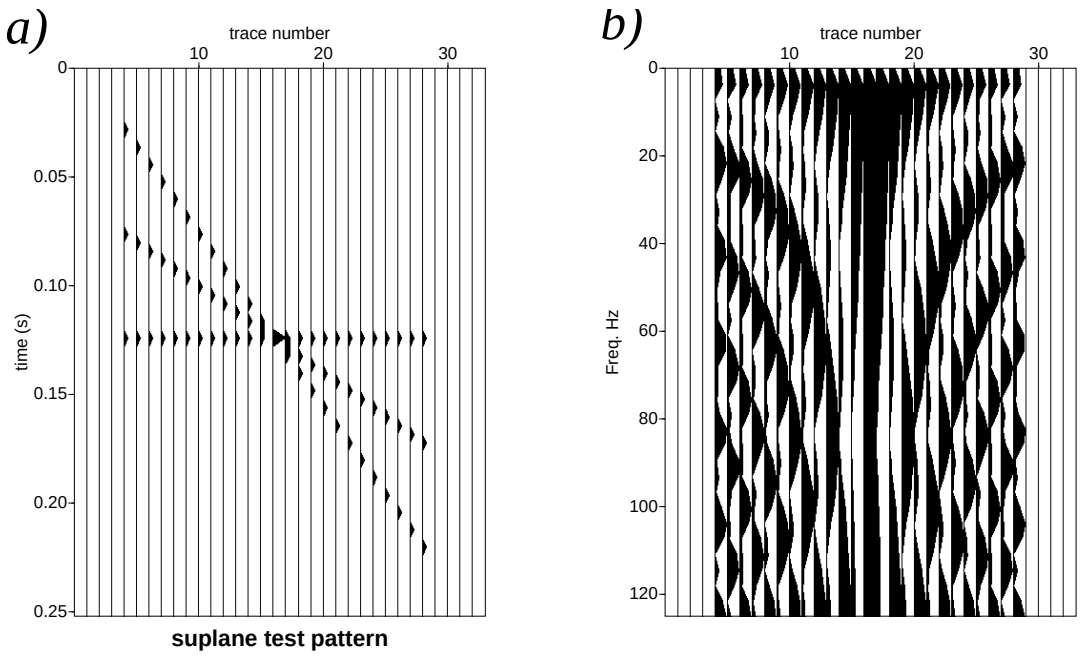


Figure 2.2: a) The **suplane** test pattern. b) the Fourier transform (time to frequency) of the **suplane** test pattern via **suspecfx**.

- program | program = pipe from program to program
- program > file = pour data from program to file (redirect out)
- program < file = pour data from file to program (redirect in)
- program ... & = run program in background

2.2 Stringing commands together

We may string together programs via pipes (|), and input and output via redirects (>) and (<). An example is to use the program suspecfx to look at the amplitude spectrum of the traces in data made with suplane:

```
$ suplane | suspecfx | suxwigb   &          --make suplane data, find
                                          the amplitude spectrum,
                                          plot as wiggle traces
```

Equivalently, we may do

```
$ suplane > junk.su                  --make suplane data, write to a file.
$ suspecfx < junk.su > junk1.su      --find the amplitude spectrum, write to
                                     a file.
$ suxwigb < junk1.su   &              -- view the output as wiggle traces.
```

This does exactly the same thing, in terms of final output as the previous example, with the exception that here, two files have been created. See Figure 2.2.

2.2.1 Questions for discussion

- What is the Fourier transform of a function?
- What is an amplitude spectrum?
- Why do the plots of the amplitude spectrum in Figure 2.2 appear as they do?

2.2.2 The FFT versus the DFT

Another example of the Fourier transform is the comparison of the FFT (Fast Fourier Transform) and the ordinary discrete Fourier transform (DFT). The FFT exploits some of the symmetries in the digital representation of the Fourier transform to produce faster algorithm. There is a difference in the performace of these two versions of the same transform.

We can make some constant frequency data with **suvibro**. The program **suvibro** is used to simulate vibroseis sweeps, but it also may be used to simulate constant frequency signals. For example

```
$ suvibro f1=30 f2=30 t1=0 t2=0 tv=100 > junk1.su
$ suvibro f1=31 f2=31 t1=0 t2=0 tv=100 > junk2.su
$ suxgraph < 30hz.su  &
$ suxgraph < 31hz.su  &
$ susum junk1.su junk2.su > junk.su
$ suxgraph < junk.su
```

generate, respectively 30 Hz and 31 Hz cosine waves. You might have to stretch the suxgraph to see the plots properly. Note the beat frequency of 1 Hz in summed version of the file junk.su. The signals differ by 1 Hz, so constuctively and destructively interfere with this fequency.

If we want to show the spectrum of these signals we can use the FFT algorithm

```
$ suspecfx < junk.su | suxwigb label1="Frequency Hz" label2="Spectral Amplitude"&
```

Alternatively the spectrum may be computed via the DFT

```
$ suslowft < junk.su | suamp mode=amp | suxwigb &
```

This will take considerably more time (hence the name) than the FFT version. The extra call to **suamp** is necessary because the output from **suslowft** are the real and imaginary parts of the Fourier transform output. The amplitude spectrum is the modulus of these complex numbers.

The corresponding FFT that outputs real and imagiary parts

```
$ sufft < junk.su | suamp mode=amp | suxwigb &
```

Working with NFS files

Files saved on the UITS central Unix computers Chrome, Cobalt, Zinc, Steel, EZinfo, and STARRS/SP are stored on the Network File Server (NFS). That means that your files are really on one disk, in directories named for the central Unix hosts on which you have accounts.

No matter which of these computers you are logged into, you can get to your files on any of the others. Here are the commands to use to get to any system directory from any other system:

cd /N/u/*username*/Chrome/
cd /N/u/*username*/Cobalt/
cd /N/u/*username*/Zinc/
cd /N/u/*username*/Steel/
cd /N/u/*username*/Ezinfo/
cd /n/u/*username*/SP/

Be sure you use the capitalization just as you see above, and substitute your own username for *username*.

For example, if Jessica Rabbit is logged into her account on Steel, and wants to get a file on her EZinfo account, she would enter:

cd /N/u/jrabbit/Ezinfo/

Now when she lists her files, she'll see her EZinfo files, even though she's actually logged into Steel.

You can use the ordinary Unix commands to move files, copy files, or make symbolic links between files. For example, if John Doe wanted to move "file1" from his Steel directory to his EZinfo directory, he would enter:

mv -i /N/u/jdoe/Steel/file1 /N/u/jdoe/Ezinfo/

This shared file system means that you can access, for example, your Chrome files even when you are logged into Cobalt, and vice versa. However, if you are logged into Chrome, you can only use the software installed on Chrome —only users' directories are linked together, not system directories.

Figure 2.3: UNIX Quick Reference card p1. From the University References

Unix commands reference card

Abbreviations used in this pamphlet

Ctrl/x	hold down control key and press x
d	directory
env	environment
f	filename
n	number
nd	computer node
var	variable
[y/n]	yes or no
[]	optional arg
...	list

August 1998

To access this guide on the
World Wide Web, set your browser to
http//www.indiana.edu/~uitspubs/b017/

Environment Control

Command	Description
cd *d*	*Change to directory d*
mkdir *d*	*Create new directory d*
rmdir *d*	*Remove directory d*
mv *f1 [f2...] d*	*Move file f to directory d*
mv *d1 d2*	*Rename directory d1 as d2*
passwd	*Change password*
alias *name1 name2*	*Create command alias*
unalias *name1*	*Remove command alias name1*
rlogin *nd*	*Login to remote node*
logout	*End terminal session*
setenv *name v*	*Set env var to value v*
unsetenv *name1 name2...]*	*remove environment variable*

Output, Communication, & Help

Command	Description
lpr -P *printer f*	*Output file f to line printer*
script *[f]*	*Save terminal session to f*
exit	*Stop saving terminal session*
mail *username*	*Send mail to user*
biff *[y/n]*	*Instant notification of mail*
man *name*	*UNIX manual entry for name*
learn	*Online tutorial*

Process Control

Command	Description
Ctrl/c *	*Interrupt processes*
Ctrl/s *	*Stop screen scrolling*
Ctrl/q *	*Resume screen output*
sleep *n*	*Sleep for n seconds*
jobs	*Print list of jobs*
kill [%n]	*Kill job n*
ps	*Print process status stats*
kill -9 *n*	*Remove process n*
Ctrl/z *	*Suspend current process*
stop *%n*	*Suspend background job n*
command&	*Run command in background*
bg [*%n*]	*Resume background job n*
fg [*%n*]	*Resume foreground job n*
exit	*Exit from shell*

Environment Status

Command	Description
ls *[d] [f...]*	*List files in directory*
ls *-l [f...]*	*List files in detail*
alias *[name]*	*Display command aliases*
printenv *[name]*	*Print environment values*
quota	*Display disk quota*
date	*Print date & time*
who	*List logged in users*
whoami	*Display current user*
finger *[username]*	*Output user information*
chfn	*Change finger information*
pwd	*Print working directory*
history	*Display recent commands*
! n	*Submit recent command n*

File Manipulation

Command	Description
vi [f]	*Vi fullscreen editor*
emacs *[f]*	*Emacs fullscreen editor*
ed *[f]*	*Text editor*
wc *f*	*Line, word, & char count*
cat *f*	*List contents of file*
more *f*	*List file contents by screen*
cat *f1 f2 > f3*	*Concatenates f1 & f2 into f3*
chmod *mode f*	*Change protection mode of f*
cmp *f1 f2*	*Compare two files*
cp *f1 f2*	*Copy file f1 into f2*
sort *f*	*Alphabetically sort f*
split *[-n] f*	*Split f into n-line pieces*
mv *f1 f2*	*Rename file f1 as f2*
rm *f*	*Delete (remove) file f*
grep *'ptn' f*	*Outputs lines that match ptn*
diff *f1 f2*	*Lists file differences*
head *f*	*Output beginning of f*
tail *f*	*Output end of f*

Compiler

Command	Description
cc *[-o f1] f2*	*C compiler*
lint *f*	*Check C code for errors*
f77 *[-o f1] f2*	*Fortran77 compiler*
pc *[-o f1] f2*	*Pascal compiler*

Press RETURN at the end of each command, except those marked by an asterisk (*).

Figure 2.4: UNIX Quick Reference card p2.

32

2.3 Unix Quick Reference Cards

The two figures, Fig 2.3 and Fig 2.4 are a Quick Reference cards for some Unix commands

References

Sobell, M. (2010), "A practical guide to Linux commands, editors, and shell programming" Pearson Education Inc., Boston, MA.

Chapter 3

Lab Activity #2 - viewing data

Just as scratch paper is paper that you use temporarily without the plan of saving for the long term, a "scratch directory" is temporary working space, which is not backed up and which may be arbitrarily cleared by the system administrator. Each computer in this lab has a directory called **/scratch** that is provided as a temporary workspace for users.

On our lab system there is a shared scratch space called **/gpfc**. It is in this location that you will be working with data. Create your own directory via:

```
$ mkdir /gpfc/yourusername
```

Here "yourusername" is the actual username that you are designated as on this system. Please feel free to ask for help as you need it.

A directory like **/gpfc** may reside physically on the computer where you are sitting, or it may be remotely mounted. In computer environments where the directory is locally on the a given computer, you will have to keep working on the same system. If you change computers, you will have to transfer the items from your personal scratch area to that new machine. In labs where the directory is remotely mounted, you may work on any machine that has the directory mounted.

Remember: **/scratch** directories are **not backed up**. If you want to save materials permanently, it is a good idea to make use of a USB storage device.

3.0.1 Data image examples

Three small datasets are provided. These are labeled "sonar.su," "radar.su," and "seismic.su" and are located in the directory

```
/data/cwpscratch/Data1/
```

We will pretend that these data examples are "data images," which is to say these are examples that require no further processing.

Do the following:

```
$ cd /gpfc/yourusername     (this takes you to /gpfc/yourusername)

^
This "$" represents the prompt at the beginning of the commandline.
Do not type the "$" when entering commands.


$ mkdir  Temp1               (this creates the directory Temp1)
$ cd  Temp1                  (change working directory to Temp1)
$ cp /data/cwpscratch/Data1/sonar.su  .
$ cp /data/cwpscratch/Data1/radar.su  .
$ cp /data/cwpscratch/Data1/seismic.su  .
                                        ^  This is a literal dot ".", which
                                       means "the current directory"
$ ls              ( should show  the file sonar.su )
```

For the rest of this document, when you are directed to make "Temp" directories, it will be assumed that you are putting these in your personal scratch directory.

3.1 Viewing an SU data file: Wiggle traces and Image plots

Though we are assuming that the examples **sonar.su**, **seismic.su**, and **radar.su** are finished products, our mode of presentation of these datasets may change the way we view them entirely. Proper presentation can enhance features we want to see, suppress parts of the data that we are less interested in, accentuate signal and suppress noise. Improper presentation, on the other hand, can take turn the best images into something that is totally useless.

3.1.1 Wiggle traces

A common mode of presentation of seismic data is the "wiggle trace." Such a representation consists of representing the oscillations of the data as a graph of amplitude as a function of time, with successive traces plotted side-by-side. Amplitudes of one polarity (usually positive) are shaded black, where as negative amplitudes are not shaded. Be aware that such presentation introduces a bias in the way we view the data, accentuating the positive amplitudes. Furthermore, wiggle traces may make dipping structures appear fatter than they actually are owing to the fact that a trace is a vertical slice through the data.

In SU we may view a wiggle trace display of data via the program **suxwigb**. For example, viewing the **sonar.su** data as wiggle traces is done by "redirecting in" the data file into "suxwigb"

```
$ suxwigb < sonar.su      &
                          ^ the ampersand (&) means "run in background"
                            so you get your commandline back
```

This should look horrible! The problem is that there are 584 wiggle traces, side by side. Place the cursor on the plot and drag, while holding down the index finger mouse button. This is called a "rubberband box." Try grabbing a strip of the data of width less than 100 traces, by placing the cursor at the top line of the plot, and holding the index finger mouse button while dragging to the lower right. Zooming in this fashion will show wiggles. The less on here is that you need a relatively low density of data on your print medium for wiggle traces.

Place the mouse cursor on the plot, and type "q" to kill the window.

Try the **seismic.su** and the **radar.su** data as wiggle traces via

```
$ suxwigb < seismic.su      &
$ suxwigb < radar.su      &
```

In each case, zoom in on the data until you are able to see the oscillations of the data.

3.1.2 Image plots

The seismic data may be thought of as an array of floating point numerical values, each representing a seismic amplitude at a specific (t, x) location. A plot consisting of an array of gray or color dots, with each gray level or color representing the respective value is called an "image" plot.

If we view An alternative is an image plot:

```
$ suximage < sonar.su      &
```

This should look better. We usually use image plots for datasets of more than 50 traces. We use wiggle traces for smaller datasets.

3.2 Greyscale

There are only 256 shades of gray available in this plot. If a single point in the dataset makes a large spike, then it is possible that most of the 256 shades are used up by that one amplitude. Therefore scaling amplitudes is often necessary. The simplest processing of the data is to amplitude truncate ("clip") the data. (The term "clip" refers to old time strip chart records, which when amplitudes were too large appeared if someone had taken scissors and clipped of the tops of the sinusoids of the oscillations.) Try:

```
$ suximage < sonar.su  perc=99     &
$ suximage < sonar.su perc=99 legend=1
```

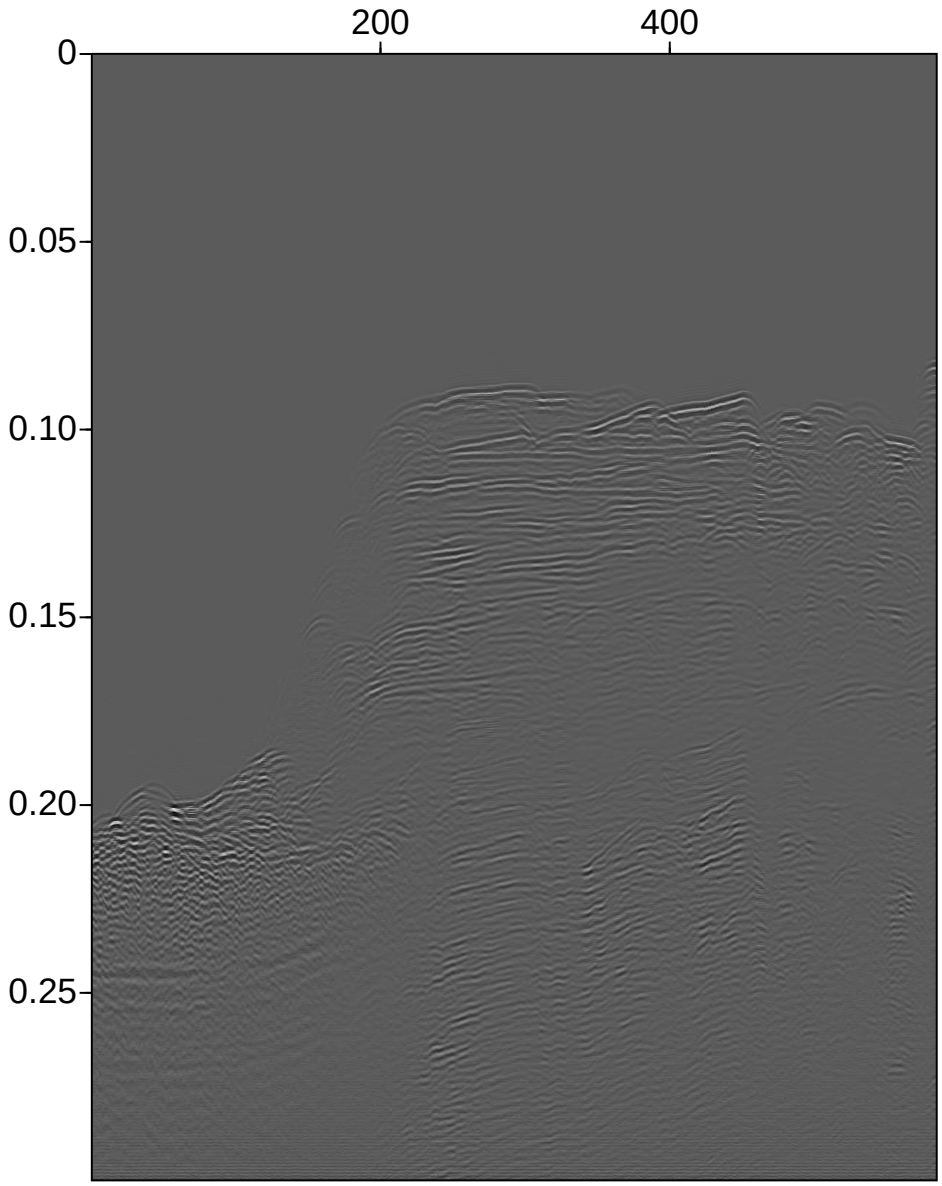


Figure 3.1: Image of **sonar.su** data (no perc). Only the largest amplitudes are visible.

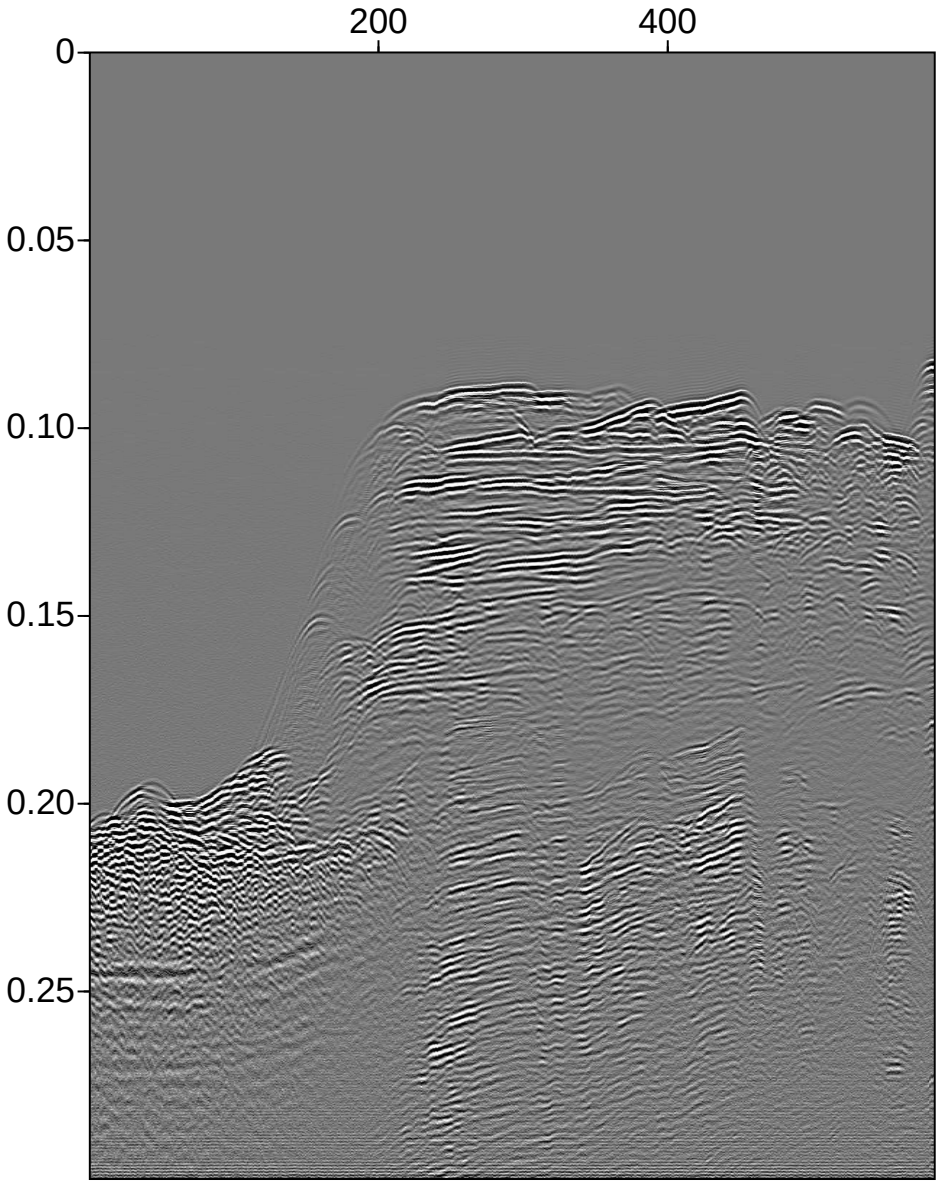


Figure 3.2: Image of **sonar.su** data with perc=99. Clipping the top 1 percentile of amplitudes brings up the lower amplitude amplitudes of the plot.

The perc=99 passes only those items of the 99th percentile and below in amplitude. (You may need to look up "percentile" on the Internet.) In other words, it "clips" (amplitude truncates) the data to remove the top 1 per cent of amplitudes, which might suck up the majority of shades of gray. Try different values of "perc" to see what this does.

3.3 Legend ; making grayscale values scientifically meaningful

To be scientifically useful, which is to say "quantitative" we need to be able to translate shades of gray into numerical values. This is done via a gray scale, or "legend". A "legend" is a scale or other device that allows us to see the meanings of the graphical convention used on a plot. Try:

```
$ suximage < sonar.su  legend=1     &
```

This will show a grayscale bar.

There are a number of colorscales available. Place the mouse cursor on the plot and press "h" you will see that further pressings of "h" will re plot the data in a different colorscale. Now press "r" a few times. The "h" scales are scales in "hue" and the "r" scales are in red-green-blue (rgb). It is important to see that the brightest part of each scale is chosen to emphasize a different amplitude.

With colormapping some parts of the plot may be emphasized at the expense of other parts. The issue of colormaps often is one of selecting the location of the "bright part" of the colorbar, versus darker colors. Even perfectly processed data may be rendered uninterpretable by a poor selection of colormapping. This effect may be seen in Figure 3.4.

Repeat the previous, this time clipping by percentile

```
$ suximage < sonar.su  legend=1  perc=99    &
```

The ease at which colorscales are defined, and the fact that there are no real standards on colorscales, mean that effectively every color plot you encounter requires a colorscale for you to be able to know what the values mean. Furthermore, some colors ranges are brighter than others. By moving the bright color to a different part of the amplitude range, you can totally change the image. This is a source of richness of display, but it is also a potential source of trouble, if the proper balance of color is not chosen.

3.4 Display balancing and display gaining

A common data amplitude balancing is to balance the colorscale on the median values in the data. The "median" is the middle value, meaning that half the values are larger than the median value and half the data are less than the median value. Thus, the traces are normalized by this middle value.

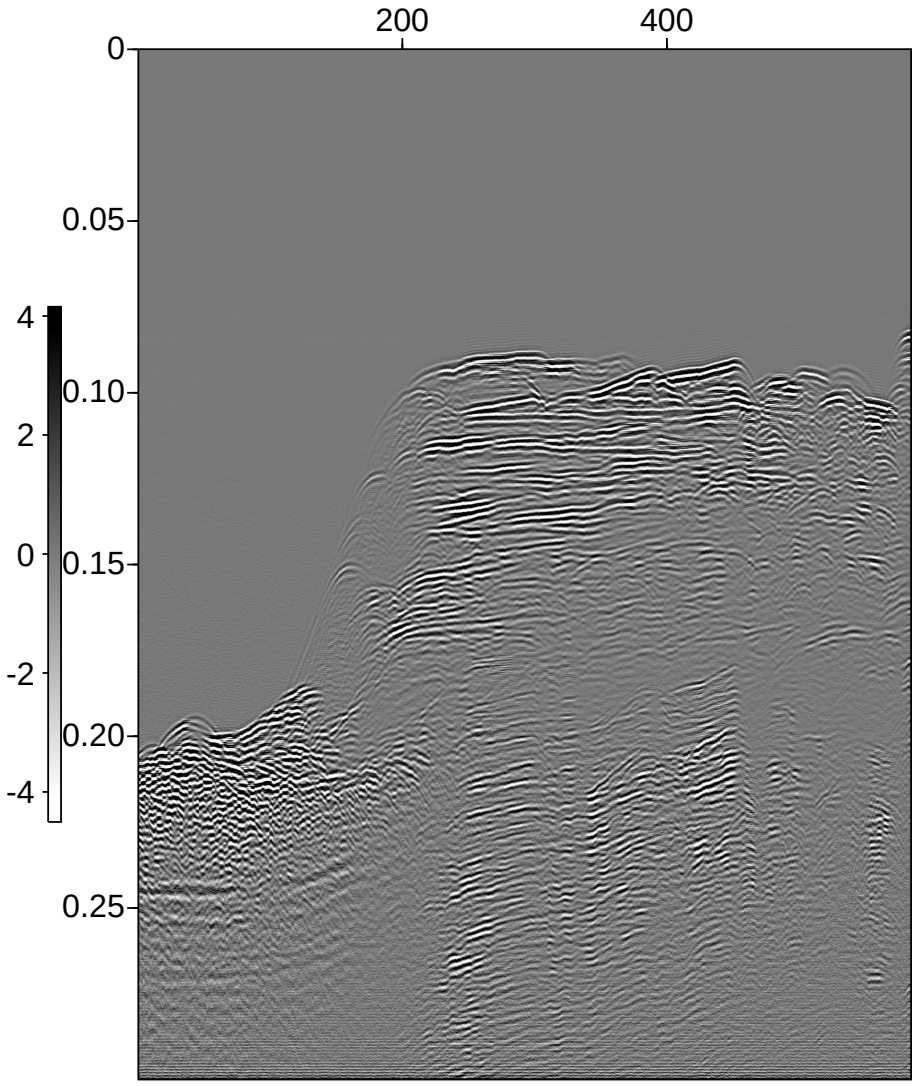


Figure 3.3: Image of **sonar.su** data with perc=99 and legend=1.

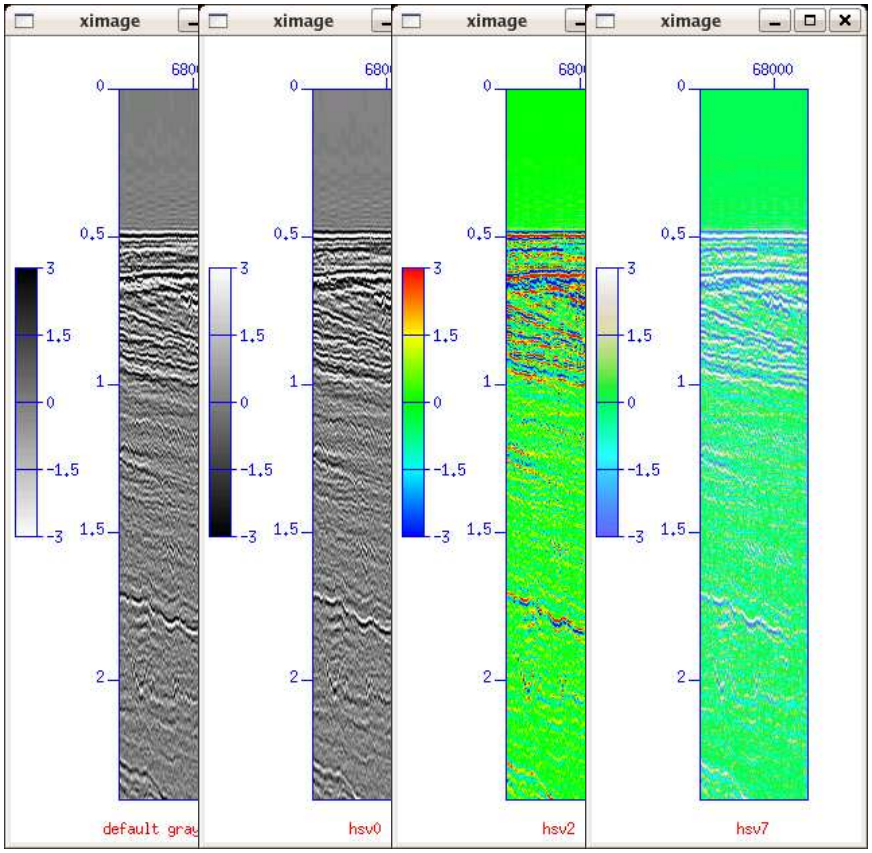


Figure 3.4: Comparison of the default, hsv0, hsv2, and hsv7 colormaps. Rendering these plots in grayscales emphasizes the location of the bright spot in the colorbar.

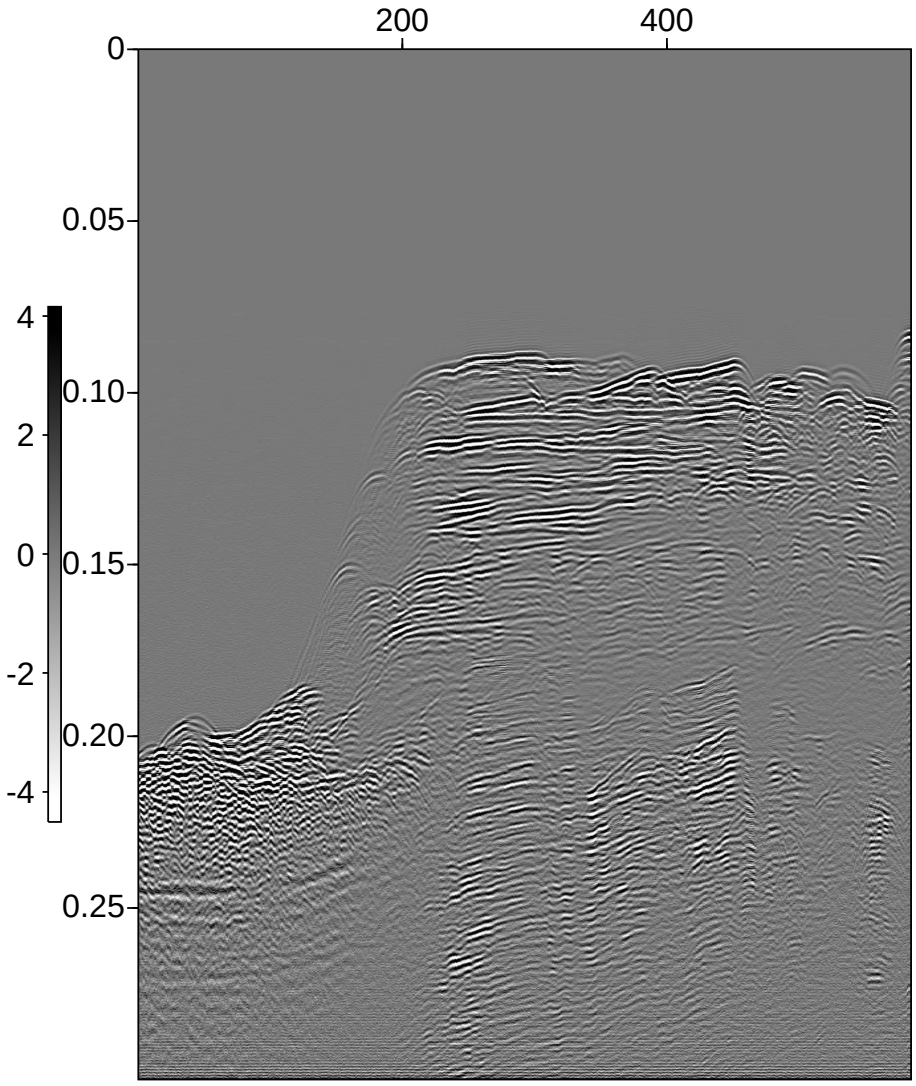


Figure 3.5: Image of **sonar.su** data with perc=99 and legend=1.

Another possibility is to scale traces by dividing by some constant value. For example dividing each trace by the square root of the average of the sum of the square of its values, the so called "root mean squared" (RMS) amplitude.

Type these commands to see that in SU:

```
$ sunormalize norm=balmed < sonar.su | suximage legend=1
$ sunormalize norm=balmed < sonar.su | suximage legend=1 perc=99
```

You may find **perc=99** to be useful. You may find that you have to apply an "RMS" balancing to make the data look a bit more uniform

```
$ sunormalize norm=balmed < sonar.su |
                    sunormalize norm=rms |  suximage legend=1
$ sunormalize norm=balmed < sonar.su |
                    sunormalize norm=rms |  suximage legend=1 perc=99
```

Again, these commands are written as one long line, and are broken here to fit on the page. You may zoom in on regions of the plot you find interesting.

If you put both the median normalized and simple perc=99 files on the screen side-by-side, there are differences, but these may not be striking differences. The program **suximage** has a feature that the user may change colormaps by pressing the "h" key or the "r" key. Try this and you will see that the selection of the colormap can make a considerable difference in the appearance of the image. Even with the same data, the colormap.

For example in Figure 3.7 we see the result of applying median balancing. We might consider applying sunormalize directly to the seismic data

```
$ suximage < seismic.su wbox=250 hbox=600 cmap=hsv4 clip=3 title="no median" &
```

compared with applying the median balancing

```
$ sunormalize norm=balmed < seismic.su  |
            suximage wbox=250 hbox=600
                        cmap=hsv4 clip=3 title="median filtering" &
```

This result looks bizarre because the traces individually have different median values and consequently have different ranges of amplitudes. An improved picture may be obtained by applying an RMS normalization to the traces after they have been median filtered via,

```
$ sunormalize norm=balmed < seismic.su  | sunormalize norm=rms |
       suximage wbox=250 hbox=600
                        cmap=hsv4 clip=3 title="median filtering" &
```

In each of these examples, the line is broken to fit on the page. When you type this, the pipe | follows immediately after the **seismic.su**.

There are other possibilities. We may consider simply normalizing the data by the maximum or minimum value, or by some other constant. Furthermore, we have the question of whether the process be applied trace by trace, or over the whole panel of data.

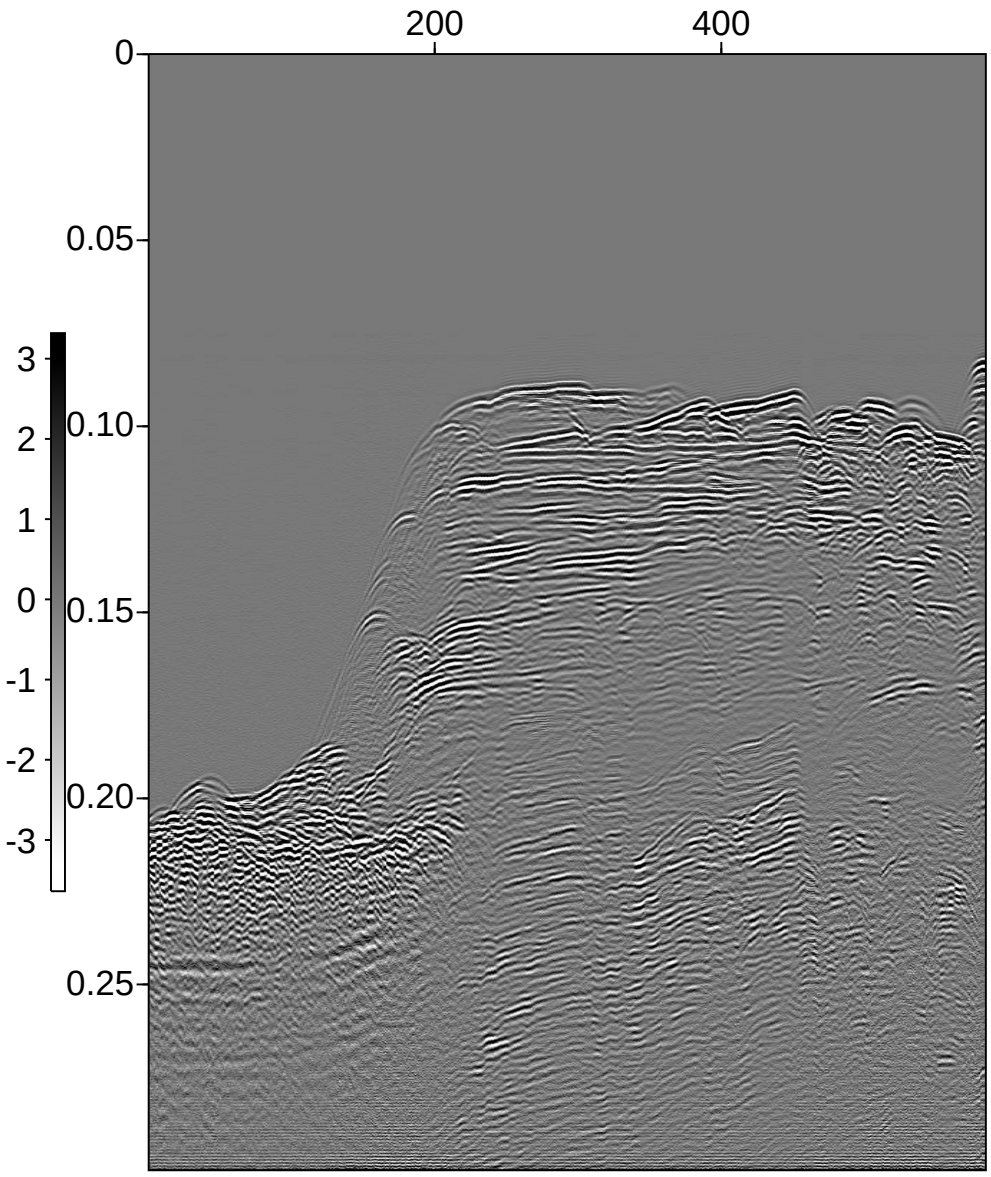


Figure 3.6: Image of **sonar.su** data with median balancing and perc=99

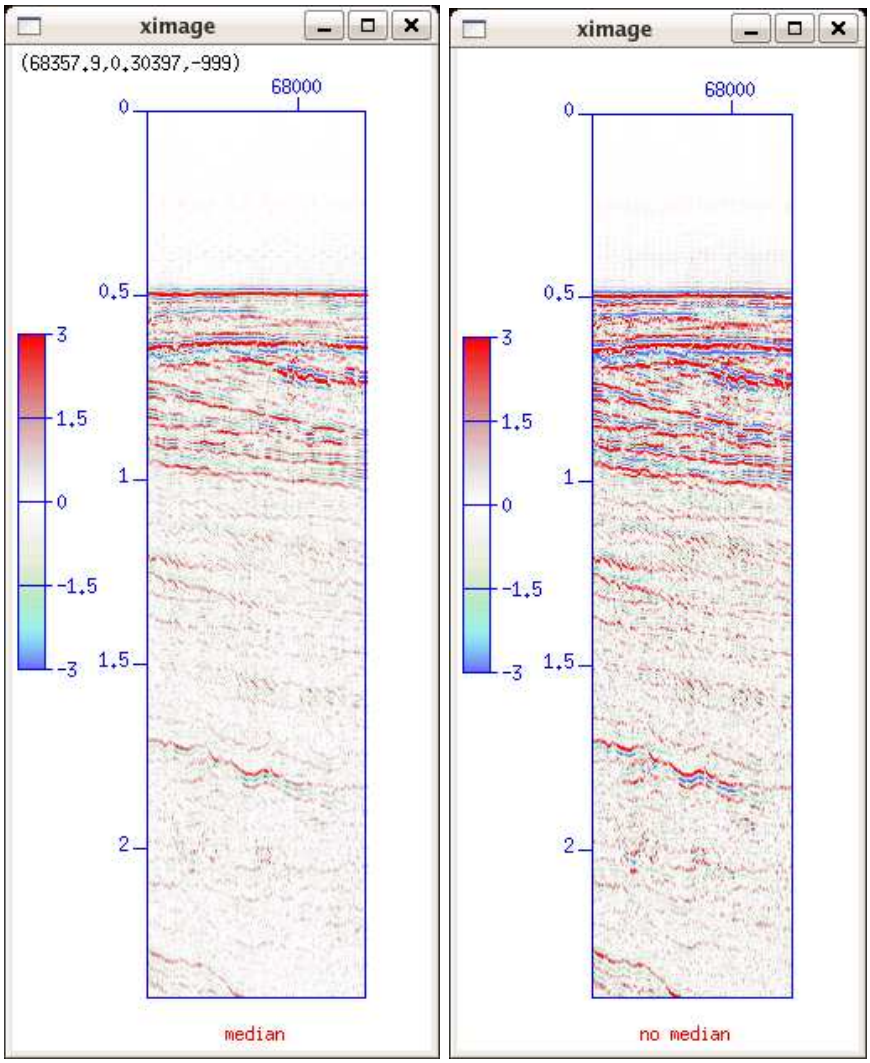


Figure 3.7: Comparison of **seismic.su** median-normalized, with the same data with no median balancing. Amplitudes are clipped to 3.0 in each case. Notice that there are features visible on the plot without median balancing that cannot be seen on the median normalized data.

3.5 Homework problem #1 - Due dates Thursday 3 Sept 2015 and Tuesday 8 September 2015

Repeat display gaining experiments of the previous section with "radar.su" and "seismic.su" to see what median balancing, and setting **perc=...** does to these data.

- Capture representative plots with axes properly labeled. You can use the Linux screen capture feature, or find another way to capture plots into a file, (such as by using **supsimage** to make PostScript plots) Feel free to use different values of perc and different colormaps than were used in the previous examples. Is median filtering better? Is it worse? Can you simply change the clip value and get a better picture?

  The OpenOffice (or LibreOffice) Word wordprocessing program is an easy program to use for this.

- Prepare a report of your results. The report should consist of:
  - Your plots (you are telling a story, show only images are relevant to your story)
  - a short paragraph describing what you saw. Think of it as a figure caption.
  - a listing of the actual commandlines that you ran to get the plots.
  - **Not more than 3 pages total!**
  - Make sure that your name, the due date, and the assignment number are at the top of the first page.

- Save your report in the form of a PDF file, and email to john@dix.mines.edu

3.6 Concluding Remarks

There are many ways of presenting data. Two of the most important questions that a scientist can ask when seeing a plot are "What is the meaning of the colorscale or grayscale of a plot?" and "What normalization or balancing has been applied to the data before the plot?" The answers to these questions may be as important as the answer to the question "What processing has been applied to these data?"

3.6.1 What do the numbers mean?

The scale divisions seen on the plots in this chapter that have been obtained by running **suximage** with **legend=1** show numerical values, values that are changed when we apply display gain. Ultimately, these numbers relate to the voltage recorded from a transducer (a geophone, hydrophone, or accelerometer). While in theory we should be able to extract information about the size of the ground displacement in, say *micrometers*,

or the pressure field strength in, say *megapascals* there is little reason to do this. Owing to detector and source coupling issues, and the fact that data must be gathered quickly, we really are only interested in relative values.

References

Stockwell, Jr. J. W. and J. K. Cohen (2008) The new SU users manual, available from http:cwp.mines.edu/cwpcodes

Chapter 4

Help features in Seismic Unix

Scientific data processing and manipulation packages usually contain many commands and options. Seismic Unix is no exception. As with any package there are help features to help you navigate the collection of programs and modules. The first thing that you must do with any software package is to locate and learn to use the help features in the package. Usually these help mechanisms are not very "helpful" to the beginner, but are really more like quick reference guides for people who are already familiar with the package.

There are a number of help features in SU; here we will discuss only three.

4.1 The selfdoc

All Seismic Unix programs have the feature that if the name of the program is typed with no arguments, a self-documentation feature called a **selfdoc** is listed.

Try:

```
$ suplane
$ suximage
$ suxwigb
$ sunormalize
```

For example:

```
$ suplane
```

yields

```
SUPLANE - create common offset data file with up to 3 planes

suplane [optional parameters] >stdout

Optional Parameters:
 npl=3                  number of planes
```

```
 nt=64          number of time samples
 ntr=32         number of traces
 taper=0                no end-of-plane taper
                        = 1 taper planes to zero at the end
 offset=400             offset
 dt=0.004               time sample interval in seconds
...plane 1 ...
        dip1=0          dip of plane #1 (ms/trace)
        len1= 3*ntr/4   HORIZONTAL extent of plane (traces)
        ct1= nt/2       time sample for center pivot
        cx1= ntr/2      trace for center pivot
...plane 2 ...
        dip2=4          dip of plane #2 (ms/trace)
        len2= 3*ntr/4   HORIZONTAL extent of plane (traces)
        ct2= nt/2       time sample for center pivot
        cx2= ntr/2      trace for center pivot
--More--
```

As with the Unix man pages, typing the space bar shows the rest of the help page.

Each of these programs has a relatively large number of possible argument settings. The programs "suxwigb" and "suximage" both call programs named, respectively, "xwigb" and "ximage". Type:

```
$ ximage
$ xwigb
```

All of the setting for "xwigb" and "ximage" apply to "suxwigb" and "suximage." That is a lot of settings.

Correspondingly, there are plotting programs that write out PostScript graphics output for plotting

```
$ supsimage
$ psimage
$ supswigb
$ pswigb
$ supswigp
$ pswigp
```

The "SU" versions of these programs call the respective programs that do not have the "su" prefix.

4.2 Finding the names of programs with: suname

SU is big package containing several hundred programs as well as hundreds of library functions, shell scripts, and associated files. Occasionally we would like to see the total scope of the package we are working with.

For an inventory of the SU programs, typing

```
$ suname
```

yields

```
 -----  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
CSHOTPLOT - convert CSHOT data to files for CWP graphers
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
--More(3%)--
```

Hitting the space bar shows the rest of the page. The **suname** output shows every library function, shell script, and main program in the package, and may be too much information for everyday usage.

What is more common is that we might want a bit more information than a selfdoc, but not a complete listing. This is where the **sudoc** feature is useful. Typing

```
$ sudoc NAME
```

yields the **sudoc** entry of the program NAME.

For example we might be interested in seeing information about **suplane**

```
$ sudoc suplane
```

and comparing that with the selfdoc for the same program

```
$ suplane
```

As the number of SU programs you come in contact increases, you will find it useful to continually be referring to the listing from suname.

The **sudoc** feature is an alternative to Unix man pages. The database of sudocs is captured from the actual selfdocs in the source code automatically via a shell script, so these do not go out of step with the actual code, the way a separately written man page might.

4.3 Lab Activity #3 - Exploring the trace header structure

You may have noticed that the plotting programs seem to know a lot about the data you have been viewing. Yet, you have never been asked to give the number of samples per trace or the number of traces. For example

```
$ suximage < sonar.su   perc=99  &
```

shows a plot without being told the dimensions of the data.

But how did the program know the number of traces and the number of samples per trace in the data? The program knows because this, and all other SU programs read information from a "header" that is present on each seismic trace.

4.3.1 What are the trace header fields-sukeyword?

If you type:

```
$ sukeyword -o
```

you will obtain a listing of the file **segy.h**, which defines the SU trace header format. The term "segy" is derived from SEG-Y a popular data exchange standard established by the Society of Exploration Geophysicists (SEG) in 1975 and later revised in 2005. The SU trace header is largely the same as that defined for the SEG-Y format.

The first 240 bytes of each seismic trace in a SEG-Y dataset consist of this trace header. The data are always uniformly sampled in time, so the "data" portion of the trace, consisting of amplitude values only, follows immediately after the trace header. While it may be tempting to think of a seismic section as an "array" of traces, in the computer, these traces simply follow one after the other.

The part of the listing from sukeyword that is relevant at this point is

```
...skipping
```

```
typedef struct { /* segy - trace identification header */

int tracl; /* Trace sequence number within line
   --numbers continue to increase if the
   same line continues across multiple
   SEG Y files.
   byte# 1-4
 */

int tracr; /* Trace sequence number within SEG Y file
   ---each file starts with trace sequence
   one
   byte# 5-8
 */

int fldr; /* Original field record number
   byte# 9-12
*/

int tracf; /* Trace number within original field record
   byte# 13-16
*/

int ep; /* energy source point number
   ---Used when more than one record occurs
   at the same effective surface location.
   byte# 17-20
 */

int cdp; /* Ensemble number (i.e. CDP, CMP, CRP,...)
   byte# 21-24
*/

int cdpt; /* trace number within the ensemble
   ---each ensemble starts with trace number one.
   byte# 25-28
 */

short trid; /* trace identification code:
-1 = Other
         0 = Unknown
 1 = Seismic data
 2 = Dead
```

```
 3 = Dummy
 4 = Time break
 5 = Uphole
 6 = Sweep
 7 = Timing
 8 = Water break
 9 = Near-field gun signature
10 = Far-field gun signature
11 = Seismic pressure sensor
12 = Multicomponent seismic sensor
- Vertical component
13 = Multicomponent seismic sensor
- Cross-line component
14 = Multicomponent seismic sensor
- in-line component
15 = Rotated multicomponent seismic sensor
- Vertical component
16 = Rotated multicomponent seismic sensor
- Transverse component
17 = Rotated multicomponent seismic sensor
- Radial component
18 = Vibrator reaction mass
19 = Vibrator baseplate
20 = Vibrator estimated ground force
21 = Vibrator reference
22 = Time-velocity pairs
23 ... N = optional use
(maximum N = 32,767)

Following are CWP id flags:

109 = autocorrelation
110 = Fourier transformed - no packing
     xr[0],xi[0], ..., xr[N-1],xi[N-1]
111 = Fourier transformed - unpacked Nyquist
     xr[0],xi[0],...,xr[N/2],xi[N/2]
112 = 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)
113 = Complex signal in the time domain
     xr[0],xi[0], ..., xr[N-1],xi[N-1]
114 = Fourier transformed - amplitude/phase
     a[0],p[0], ..., a[N-1],p[N-1]
115 = Complex time signal - amplitude/phase
     a[0],p[0], ..., a[N-1],p[N-1]
116 = Real part of complex trace from 0 to Nyquist
117 = Imag part of complex trace from 0 to Nyquist
118 = Amplitude of complex trace from 0 to Nyquist
119 = Phase of complex trace from 0 to Nyquist
121 = Wavenumber time domain (k-t)
122 = Wavenumber frequency (k-omega)
123 = Envelope of the complex time trace
124 = Phase of the complex time trace
125 = Frequency of the complex time trace
126 = log amplitude
127 = real cepstral domain F(t_c)= invfft[log[fft(F(t)]]
130 = Depth-Range (z-x) traces
201 = Seismic data packed to bytes (by supack1)
202 = Seismic data packed to 2 bytes (by supack2)
   byte# 29-30
*/

short nvs; /* Number of vertically summed traces yielding
   this trace. (1 is one trace,
   2 is two summed traces, etc.)
   byte# 31-32
 */

short nhs; /* Number of horizontally summed traces yielding
   this trace. (1 is one trace
   2 is two summed traces, etc.)
   byte# 33-34
 */

short duse; /* Data use:
1 = Production
2 = Test
   byte# 35-36
 */
```

```
int offset; /* Distance from the center of the source point
   to the center of the receiver group
   (negative if opposite to direction in which
   the line was shot).
   byte# 37-40
 */

int gelev; /* Receiver group elevation from sea level
   (all elevations above the Vertical datum are
   positive and below are negative).
   byte# 41-44
 */

int selev; /* Surface elevation at source.
   byte# 45-48
 */

int sdepth; /* Source depth below surface (a positive number).
   byte# 49-52
 */

int gdel; /* Datum elevation at receiver group.
   byte# 53-56
*/

int sdel; /* Datum elevation at source.
   byte# 57-60
*/

int swdep; /* Water depth at source.
   byte# 61-64
*/

int gwdep; /* Water depth at receiver group.
   byte# 65-68
*/

short scalel; /* Scalar to be applied to the previous 7 entries
   to give the real value.
   Scalar = 1, +10, +100, +1000, +10000.
   If positive, scalar is used as a multiplier,
   if negative, scalar is used as a divisor.
   byte# 69-70
```

```
 */

short scalco; /* Scalar to be applied to the next 4 entries
   to give the real value.
   Scalar = 1, +10, +100, +1000, +10000.
   If positive, scalar is used as a multiplier,
   if negative, scalar is used as a divisor.
   byte# 71-72
 */

int  sx; /* Source coordinate - X
   byte# 73-76
*/

int  sy; /* Source coordinate - Y
   byte# 77-80
*/

int  gx; /* Group coordinate - X
   byte# 81-84
*/

int  gy; /* Group coordinate - Y
   byte# 85-88
*/

short counit; /* Coordinate units: (for previous 4 entries and
for the 7 entries before scalel)
   1 = Length (meters or feet)
   2 = Seconds of arc
   3 = Decimal degrees
   4 = Degrees, minutes, seconds (DMS)

In case 2, 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

In case 4, to encode +-DDDMMSS
counit = +-DDD*10^4 + MM*10^2 + SS,
with scalco = 1. To encode +-DDDMMSS.ss
counit = +-DDD*10^6 + MM*10^4 + SS*10^2
with scalco = -100.
```

```
   byte# 89-90
*/

short wevel; /* Weathering velocity.
   byte# 91-92
*/

short swevel; /* Subweathering velocity.
   byte# 93-94
*/

short sut; /* Uphole time at source in milliseconds.
   byte# 95-96
*/

short gut; /* Uphole time at receiver group in milliseconds.
   byte# 97-98
*/

short sstat; /* Source static correction in milliseconds.
   byte# 99-100
*/

short gstat; /* Group static correction  in milliseconds.
   byte# 101-102
*/

short tstat; /* Total static applied  in milliseconds.
   (Zero if no static has been applied.)
   byte# 103-104
*/

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
   byte# 105-106
*/

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
   byte# 107-108
*/

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)
   byte# 109-110
*/

short muts; /* mute time--start
   byte# 111-112
*/

short mute; /* mute time--end
   byte# 113-114
*/

unsigned short ns; /* number of samples in this trace
   byte# 115-116
*/

unsigned short dt; /* sample interval; in micro-seconds
   byte# 117-118
*/

short gain; /* gain type of field instruments code:
1 = fixed
2 = binary
3 = floating point
4 ---- N = optional use
   byte# 119-120
*/

short igc; /* instrument gain constant
   byte# 121-122
*/

short igi; /* instrument early or initial gain
   byte# 123-124
```

```
*/

short corr; /* correlated:
1 = no
2 = yes
   byte# 125-126
*/

short sfs; /* sweep frequency at start
   byte# 127-128
*/

short sfe; /* sweep frequency at end
   byte# 129-130
*/

short slen; /* sweep length in ms
   byte# 131-132
*/

short styp; /* sweep type code:
1 = linear
2 = cos-squared
3 = other
   byte# 133-134
*/

short stas; /* sweep trace length at start in ms
   byte# 135-136
*/

short stae; /* sweep trace length at end in ms
   byte# 137-138
*/

short tatyp; /* taper type: 1=linear, 2=cos^2, 3=other
   byte# 139-140
*/

short afilf; /* alias filter frequency if used
   byte# 141-142
*/
```

```
short afils; /* alias filter slope
   byte# 143-144
*/

short nofilf; /* notch filter frequency if used
   byte# 145-146
*/

short nofils; /* notch filter slope
   byte# 147-148
*/

short lcf; /* low cut frequency if used
   byte# 149-150
*/

short hcf; /* high cut frequncy if used
   byte# 151-152
*/

short lcs; /* low cut slope
   byte# 153-154
*/

short hcs; /* high cut slope
   byte# 155-156
*/

short year; /* year data recorded
   byte# 157-158
*/

short day; /* day of year
   byte# 159-160
*/

short hour; /* hour of day (24 hour clock)
   byte# 161-162
*/

short minute; /* minute of hour
   byte# 163-164
*/
```

```
short sec; /* second of minute
   byte# 165-166
*/

short timbas; /* time basis code:
1 = local
2 = GMT
3 = other
   byte# 167-168
*/

short trwf; /* trace weighting factor, defined as 1/2^N
   volts for the least sigificant bit
   byte# 169-170
*/

short grnors; /* geophone group number of roll switch
   position one
   byte# 171-172
*/

short grnofr; /* geophone group number of trace one within
   original field record
   byte# 173-174
*/

short grnlof; /* geophone group number of last trace within
   original field record
   byte# 175-176
*/

short gaps; /* gap size (total number of groups dropped)
   byte# 177-178
*/

short otrav; /* overtravel taper code:
1 = down (or behind)
2 = up (or ahead)
   byte# 179-180
*/

  ----------------------------------------
```

```
/* cwp local assignments */
float d1; /* sample spacing for non-seismic data
   byte# 181-184
*/

float f1; /* first sample location for non-seismic data
   byte# 185-188
*/

float d2; /* sample spacing between traces
   byte# 189-192
*/

float f2; /* first trace location
   byte# 193-196
*/

float ungpow; /* negative of power used for dynamic
   range compression
   byte# 197-200
*/

float unscale; /* reciprocal of scaling factor to normalize
   range
   byte# 201-204
*/

int ntr;  /* number of traces
   byte# 205-208
*/

short mark; /* mark selected traces
   byte# 209-210
*/

        short shortpad; /* alignment padding
   byte# 211-212
*/


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
   byte# 213-240
*/
#endif

float  data[SU_NFLTS];

} segy;
```

Not all of these header fields get used all of the time. Some headers are more important than others. The most relevant fields to normal SU usage are the header fields **tracl**, **tracr**, **dt**, **cdp**, **offset**, **sx**, **gx**, **sy**, **gy**, and **delrt**.

To see the header field ranges on **sonar.su**, **radar.su**, and **seismic.su** type

```
$ surange < sonar.su
584 traces:
tracl    1 584 (1 - 584)
cdp      1 584 (1 - 584)
muts     75
ns       3000
dt       100

$ surange < radar.su
501 traces:
tracl    1 501 (1 - 501)
tracr    1 501 (1 - 501)
trid     1
ns       463
dt       800
hour     11
minute   3 33 (3 - 33)
sec      0 59 (41 - 7)


$ surange < seismc.su
801 traces:
tracl    1200 2000 (1200 - 2000)
tracr    67441 115081 (67441 - 115081)
fldr     594 991 (594 - 991)
tracf    1 8 (2 - 2)
ep       700 1100 (700 - 1100)
```

```
cdp      1200 2000 (1200 - 2000)
cdpt     1 8 (2 - 2)
trid     1
nhs      57 60 (60 - 60)
gelev    -10
selev    -6
scalel   1
scalco   1
sx       18212 28212 (18212 - 28212)
gx       15000 25000 (15000 - 25000)
counit   3
mute     48
ns       601
dt       4000
```

In each case, where four numbers appear, these are the minimum and maximum values in the header followed by the first and last values in the data panel

You may use **sukeyword** to determine the meaning of any of the header field "keywords" seen here via

```
$ sukeyword   key
```

where "key" is the specific keyword. For example

```
$ sukeyword   tracl
```

returns

```
...
        int tracl;      /* Trace sequence number within line
                           --numbers continue to increase if the
                           same line continues across multiple
                           SEG Y files.
                         */
...
```

The first field **int** tells us that this is defined as type "integer" in the header. The short description is the SEG's definition for this field. This can be a big deal. Ofttimes users will want to define decimal values for the header fields.

Please note that the keyword names we use here are not an industry standard, but are peculiar to SU. These are an invention of Einar Kjartannson, the author of the original suite of programs call SY, that later became the basis of the SU package.

4.3.2 Types of data formats

In the world of scientific data there are three basic types of data formats. These are *acquisition*, *internal*, and *data exchange* formats.

Acquisition formats

An acquisition format is a data format that is natural to, or convenient for a particular instrument that is recording data. These are usually formats that are dictated by the available storage and the process by which the instruments collects and digitizes the data. Such formats often make sense in this usage, but may not be easy to work with in computer programs. The data may be multiplexed, or may be compressed in some other fashion

Examples of seismic acquisition formats include *SEG-D*, *SEG-B*, and *SEG-2*. Each of these were designed in conjunction with the needs of multichannel seismic acquisition systems. SEG-D is used for exploration seismic data, the other two are small seismograph systems. Some data acquisition systems give the user the option of writing out data in the "SEGY" format. However, in many cases this is not SEGY "by the book" but a version that is called the *DOS_SEGY* format. DOS_SEGY is based loosely on the SEGY format, but deviates from the official standard.

Internal formats

The term *internal* may refer to software, or to an organization such as a school or a company. Internal formats are just that, internal. Data in such a format is not generally for public consumption or for transport to other systems or exchange, but is a format that may make it easier for a particular suite of programs to operate. The SU data format is an internal format. Every commercial seismic package has its internal format. Such systems would include PROMAX, DISCO, etc.

Some software packages that specialize in GPR or near surface (engineering geophysics) applications may expect data to be written in the SEG-2 or SEG-B formats.

Data excahnage formats

For data to be shared between companies or other users, yet a third class of data format is required. Such formats are called *data exchange* formats. The most popular is **SEG-Y** though it is possible that data in **SEG-D**, **SEG-B**, **SEG-2**, or other format.

Any format that is relatively stable may effectively become a data exchange format, whether or not the originators of that format had this in mind. The SU data format is treated as a data exchange format by some software developers.

4.4 Concluding Remarks

Every data processing package has help features and internal documentation. None of these are usually perfect, and all are usually aimed at people who already understand the package. Look for the help features and demos of a package.

When receiving a dataset, the most important questions that a scientist can ask about a dataset that he or she receives are: "What is the format of the data?" "Are the data

uniformly sampled?" For seismic data: "Have the headers been set?" and "What are ranges of the the header values?". "Do the header values make sense?"

Note also, that data coming in from the field, frequently requires that the headers be set in the data. Transferring header information from seismic observers logs into seismic trace headers is called "setting geometry." Setting geometry can be one of the biggest headaches in preparing data for processing.

Vendors may remap the SEG-Y header fields both in terms of the meaning of the header field and with respect to the data type. Obtaining a "header map" of data when you obtain seismic data can prevent confusion and save you a lot of work.

When receiving data on tape, remember that "tape reading is more of an art than a science." It is best to ask for data in a format you can use, rather than allow someone else to dictate that format to you.

References

Stockwell, Jr. J. W. and J. K. Cohen (2008) The new SU users manual, available from http:cwp.mines.educwpcodes

Chapter 5

Lab Activity #4 - Depth conversion of "Data images"

Geophysical imaging, often called "migration" in the seismic context, is an example of a general topic called "inverse-scattering imaging." Simply stated, inverse scattering imaging is the process of "making pictures with echos." We have all encountered examples of this in our daily lives.

Our eyes operate by making images of the world around us from scattered light. Medical ultrasound uses the echos of high frequency sound waves to image structures within the human body. Ultrasound is also used in an industrial setting for nondestructive testing (NDT). Seismic prospectors look for oil using the echos of seismic waves. Earthquake seismologists determine the internal structure of the deep earth with echos of waves from earthquakes.

Near surface investigators use the echos of ground penetrating radar waves to image objects in the shallow subsurface.

5.1 Imaging as the solution to an inverse problem

Acoustic and elastic waves echo off of jumps in the wavespeed and/or the density of the medium. In the case of electromagnetic scattering, the signal is coming from a volume of material or a layer, rather than a boundary between layers, which has a differing conductivity from the surrounding material.

In each case, the propagating wave impinges on the reflector at some angle, and is reflected from at an angle determined by the *law of reflection* for the medium. For scalar waves, which is to say waves that do not experience mode conversion, the angle of incidence equals the angle of reflection. For elastic waves, the angle of reflection is a function of the angle of incidence and of the velocities and densities of the media on either side of the reflector.

The scattered wave therefore carries information about both the orientation of the reflector and its location. Thus, an image formed from such data is a solution to an inverse problem wherein the wavespeed of the medium and the location and orientation

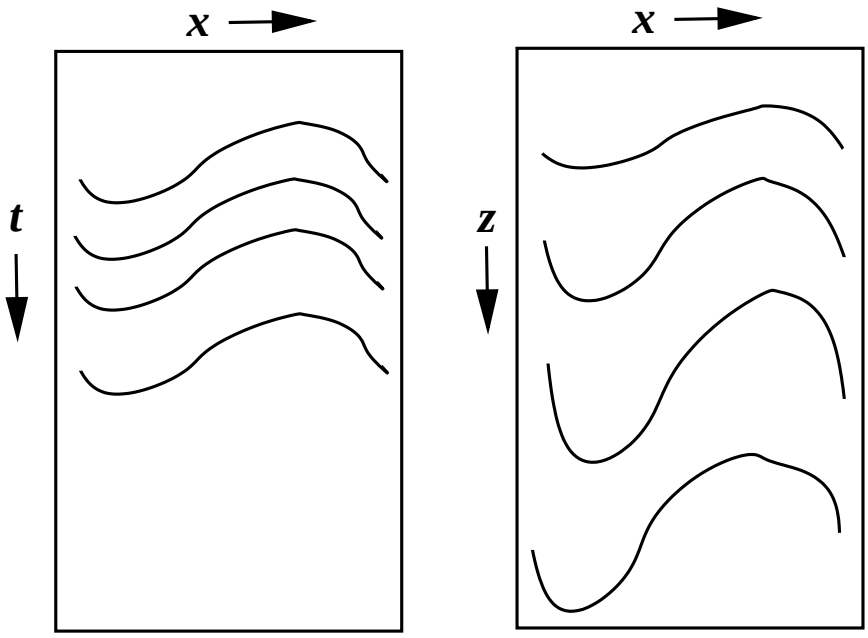


Figure 5.1: Cartoon showing the simple shifting of time to depth. The spatial coordinates $\boldsymbol{x}$ do not change in the transformation, only the time scale t is stretched to the depth scale z. Note that vertical relief looks greater in a depth section as compared with a time section.

of the reflector are the unknown variables for which we are attempting to solve.

5.2 Inverse scattering imaging as time-to-depth conversion

In geophysics there are three common types of inverse-scattering imaging techniques that may be encountered. These are "acoustic," "ground penetrating radar (GPR)," and "reflection seismic." Acoustic methods include **sonar** as well as any other echo imaging method that employs sound waves in the air or water.

In each case a species of wave is introduced into the subsurface. This wave is reflected off of structures within the Earth and travels back up to the surface of the Earth where it is recorded. In the raw form, the coordinates of the data consist of the spatial coordinates of the recording position and traveltime, which may be represented as the ordered triple of numbers $Data(x_1, x_2, t)$.

5.2.1 Migration as a mapping of data from time to space

It is implied that some form of processing is needed to convert data collected in the input coordinates of space and time $Data(x_1, x_2, t)$ into an image in the output coordinates that are purely spatial $DepthImage(y_1, y_2, y_3)$ or are new spatial coordinates and a "migrated

time coordinate" $TimeImage(y_1, y_2, \tau)$. When the output is in space and migrated time, we call the process "time migration" and the output a "time section". When the output is in purely spatial coordinates, we call the process "depth migration" and the output a "depth section". Each type of section is found useful in exploration seismic. We consider "time migration" as the focusing of the data into an image, and "depth migration" as focusing combined with depth conversion, in the simplest description.

5.2.2 Migration as focusing followed by depth conversion

Thus, for our "migration as depth conversion" we will consider the final step of processing as a process that converts the data from $Data(y_1, y_2, \tau)$ to data in $DepthImage(y_1, y_2, y_3)$ in purely spatial coordinates.

The simplest cases of such processing occur when the output spatial coordinates on the recording surface are such that $y_1 = x_1$ and $y_2 = x_2$. Then the remaining problem is to "trade time for depth". Often the symbol z is used to represent depth, with z increasing positively as we go deeper into the earth.

Clearly, special circumstances are needed for this simple case to exist. Effectively, such an imaging problem is one dimensional. This type of problem may result from the construction of synthetic well logs from migrated seismic data or making depth sections from migrated time sections.

Sonar and GPR data usually have the attribute that the same piece of equipment is used as both source and receiver. Furthermore, this source-receiver array is likely highly directional, forming a beam of energy that travels straight down into the Earth, with reflections being recorded by a detector that can only see near vertical arrivals. For sonar, this works because the scattering occurs from roughness in the structures ("rough surface scattering") of the subsurface. Thus we may consider the reflection to have occurred directly below the receiver, with little energy coming in from angles far from vertical.

To perform time-depth conversion, we need to know something about the velocities of the subsurface.

5.3 Time-to-depth with suttoz ; depth-to-time with suztot

The simplest approach to depth conversion is to use a simple velocity profile expressed as a function of time $v(t)$. How can we have velocity as a function of time? The idea is intuitive. We expect the image to show reflectors. These reflectors are the boundaries between media of different wavespeeds. Given auxiliary information about geology, well logs or the result of seismic velocity analysis, we expect to be able to relate arrivals on the seismic section to specific depth horizons, for which, in turn, we have wavespeed information.

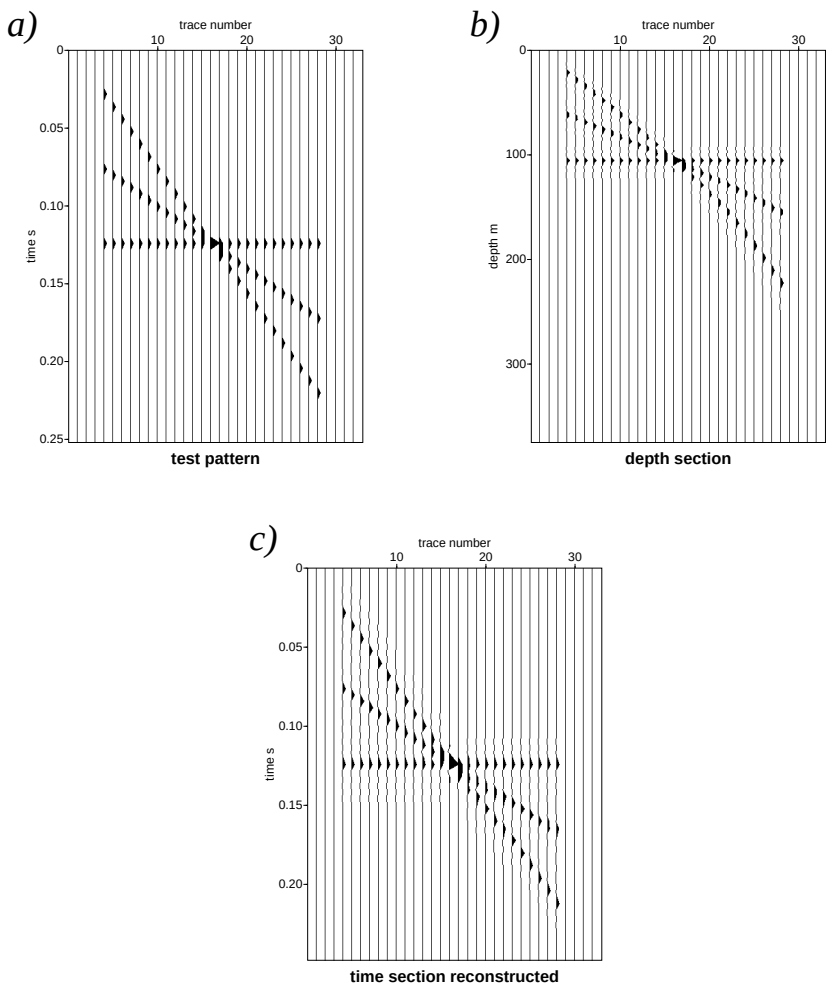


Figure 5.2: a) Test pattern. b) Test pattern corrected from time to depth. c) Test pattern corrected back from depth to time section. Note that the curvature seen depth section indicates a non piecewise-constant $v(t)$. Note that the reconstructed time section has waveforms that are distorted by repeated sinc interpolation. The sinc interpolation applied in the depth-to-time calculation has not had an anti-alias filter applied.

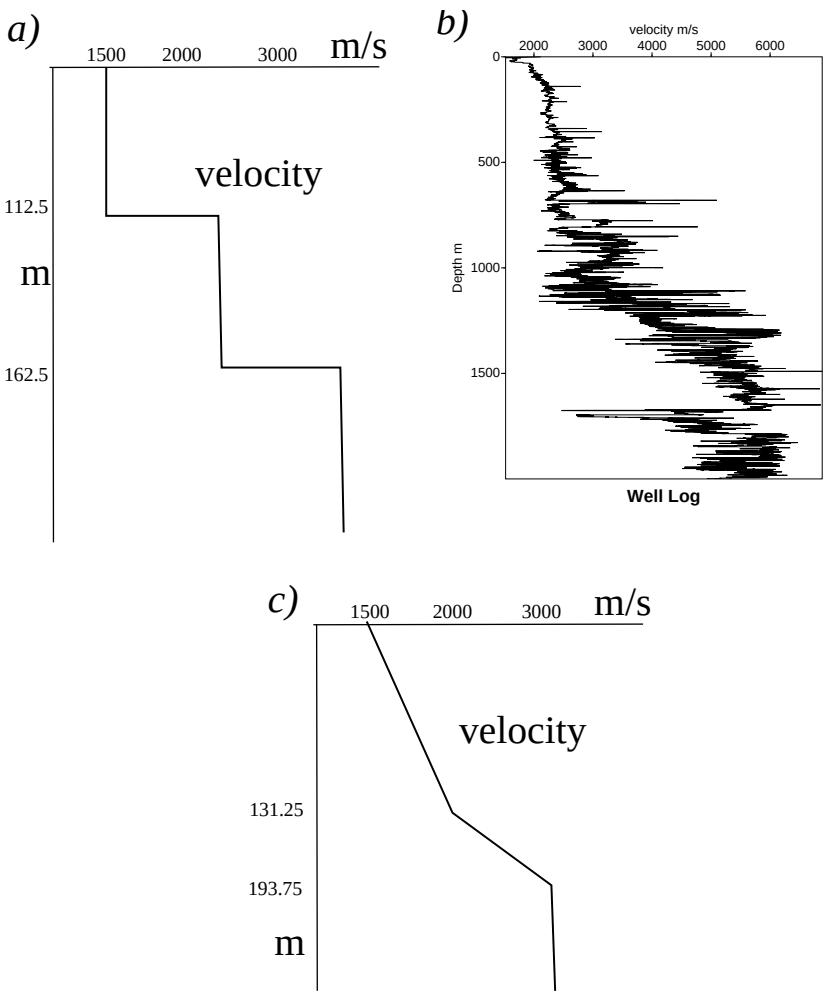


Figure 5.3: a) Cartoon showing an idealized well log. b) Plot of a real well log. A real well log is not well represented by piecewise constant layers. c) The third plot is a linearly interpolated velocity profile following the example in the text. This approximation is a better first-order approximation of a real well log.

5.4 Time to depth conversion of a test pattern

To see what the problem of time-to-depth and depth-to-time is all about, we may try **suttoz** on a test pattern made with suplane

```
$ suplane > junk.su
$ suttoz < junk.su t=0.0,.15,.2 v=1500,2000,3000 > junk1.su
$ suxwigb < junk.su title="test pattern" &
$ suxwigb < junk1.su title="depth section" &
```

The program suztot has been provided to apply depth-to-time conversion, as the inverse of sutotz. Because we know the values of the velocity that were used, we must try to figure out the depths Z1, Z2, and Z3, necessary to undo the operation

```
$ suztot < junk1.su  z=Z1,Z2,Z3 v=1500,2000,3000 > junk2.su
$ suxwigb < junk2.su title="time section reconstructed" &
```

Please note, you don't literally type "z=Z1,Z2,Z3" what you want is to find three numbers representing depths to substitute in for Z1, Z2, and Z3. The first value $Z1 = 0$.

You will notice that on the **junk1.su** data, the picture does not start getting distorted until after about depth 105. This gives a clue as to the place where the faster speeds kick in.

You will further notice that the **junk2.su** data does not look very much like the junk.su data. The first thing that you should notice is that the original *junk.su* data only goes to about .24 seconds, but the junk2.su data goes to more than .5 seconds.

It is a good idea to use **surange** to see if the header values have been changed by the processing. The original data shows

```
$ surange < junk.su
32 traces:
tracl    1 32 (1 - 32)
tracr    1 32 (1 - 32)
offset   400
ns       64
dt       4000
```

whereas the depth converted data has a greater number of samples

```
$ surange < junk1.su
32 traces:
tracl    1 32 (1 - 32)
tracr    1 32 (1 - 32)
trid     30
offset   400
ns       126                <------ ns has increased!!!
dt       4000
d1       3.000000
```

and finally, the depth-to-time converted data

```
$ surange < junk2.su
32 traces:
tracl    1 32 (1 - 32)
tracr    1 32 (1 - 32)
offset   400
ns       63                  <-------- ns is now 63
dt       4000
```

shows ns=63, rather than the original ns=64 samples.

5.4.1 How time-depth and depth-time conversion works

The way that this works is simple. Each sample of the data is a function of time. We have velocities to use for each time value. If the velocity is constant, then the process of time to depth conversion is more of a relabeling process than a calculation. However, for situations where the velocity varies as we go to later times in the data, we have to deal with the fact that the sample spacing of the time-to-depth shifted data changes as the velocity changes. Indeed, constant or piecewise-constant profiles rarely accurately represent wavespeed variation in the real earth, so there can be considerable change in the vertical location of the samples.

The depth is calculated for each sample, but because we want the output to be uniformly sampled, we have to interpolate the missing depth values. This interpolation may be done many ways, but in this program it is done by fitting a sinc function (*sinc interpolation*) to the data points. (Look up **sinc** interpolation in a textbook on signal processing.) The bandwith of this sinc function is the the band from 0 to the Nyquist frequency of the data. When resampling to a greater number of samples, the Nyquist frequency of the output is greater than the Nyquist frequency of the input, so there is no possibility of aliasing. However, if we subsample data, the potential for aliasing exists.

To repeat the test, we should be setting **nt=64** to force the number of samples to be the same on both input and output

```
$ suplane > junk.su
$ suttoz < junk.su t=0.0,.15,.2 v=1500,2000,3000 > junk1.su
$ suztot < junk1.su nt=64 z=Z1,Z2,Z3 v=1500,2000,3000 > junk2.su
$ suxwigb < junk.su title="test pattern" &
$ suxwigb < junk1.su title="depth section" &
$ suxwigb < junk2.su title="reconstructed time section" &
```

The time-to-depth may be improved by truncating the additional values

```
$ suwind itmax=64 < junk1.su | suxwigb title="time to depth" &
```

where **suwind** has been used to pass only the first 64 samples of each trace.

The short answer is that while the time-to-depth and depth-to-time conversions are ostensibly simply piecewise linear operations in this simple example, there is the potential for errors that can be introduced by the interpolation process. These errors may make process of stretching only partially invertible.

5.4.2 How to calculate the depths Z1, Z2, and Z3

The main problem is deciding how the $v(t)$ and $v(z)$ functions are to be interpolated. As is seen in Fig 5.3 constant step models often seen in cartoon well log diagrams do not accurately depict the complexity of actual well logs.

The simplest approximation to a real well log is a piecewise continuous curve, with **piecewise linear** being the simplest example of such a curve. That is, we assume a functional form of $v(t) = mt + b$ for velocity as a function of time. Here, m is the slope of the linear trend given by the ratio of the change in velocity divided by the change in time, and b would be the beginning velocity of the trend.

For the example of $t = 0.0, .15, .2 \qquad v = 1500, 2000, 3000$ in the region from $t = 0.0$ to $t = .15$ the velocity profile would be given by $v_1(t) = (500/.15)t + 1500$. Here the velocity starts at $1500m/s$ at the surface, and increases to $2000m/s$ at time $.15s$. In the second region, which begins at $t = .15s$ to $t = .2s$, the velocity profile is given by $v_2(t) = (1000/.05)(t - .15) + 2000$. Here the velocity changes by $1000m/s$ in at time increases from $.15s$ to $.2s$.

Calculating the values of the depths $Z1, Z2, Z3$, we see trivially that $Z1 = 0$ and that by integrating the two equations above yields $Z2 = 131.25$ and $Z3 = 193.75$, respectively.

To explain why this is so, suppose there was only one layer with constant velocity v_0, that "turned on" at time $t = 0$. Then we could find the depth z corresponding to each time t by noting that $z = (1/2)v_0t$ (distance = rate times time). The extra factor of $(1/2)$ appears because t is a *two-way traveltime*. If we had a $v(t)$ medium, where the value of velocity was a function of time, then we could consider this as being a collection of discrete layers, and the depth would be the sum of the thicknesses of the discrete layers

$$z = (1/2)\sum_{k=0}^{n} v_k(t_{k+1} - t_k), \tag{5.4.1}$$

where v_k is the velocity in the k-th layer.

If the respective v_k were smoothly varying velocity functions $v_k(t)$ and $\delta t_k = (t_{k+1} - t_k)$ would become dt and thickness of the k-th layer would become

$$z_k = (1/2)\int_{t_k}^{t_{k+1}} v_k(t)dt. \tag{5.4.2}$$

5.5 Sonar and Radar, bad header values and incomplete information

Most likely, depth conversion for sonar and radar requires simply knowing the speed of sound in water in the former case, and the speed of light, in the latter. This sounds

simple, and if we had all of the information in a neat and consistent form, it would be.

The first complication comes from the fact that SU is a *seismic* package. When non-seismic data are used in a seismic package, often the time sampling interval must be scaled to store the data. The reason for this is that the creators of the SEG-Y data format chose the time sampling interval **dt** not to be a floating point number, but rather as an unsigned short integer. They did this to save space in the header, requiring only 2 bytes for a short integer. On a 32 bit machine, the size of the largest value that an unsigned short can take on is 32767. Thus, scaling is necessary.

Usually, these scale factors are multiples of some power of 10. Try doing depth conversion on the sonar and radar data, using values you know for the speed of sound

```
$ suttoz v=SPEED_OF_SOUND_IN_WATER < sonar.su | suximage perc=99 &
```

and the speed of light

```
$ suttoz v=SPEED_OF_LIGHT < radar.su | suximage perc=99 &
```

respectively.

The speed of light is $2.998 \times 10^8 m/s$. The speed of sound in water is $1500 m/s$. Likely, the correct values to use in each case will be off by some multiplier that is a power of 10, owing to the fact that the natural frequencies available for radar and sonar are not in the same band as those used for seismic data.

If we type

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

we see that the time sampling interval, dt, is expressed in microseconds. Seismic frequencies range from a few Hz, to maybe 200 hz, but likely are not up into the kilohertz range, unless some special survey is being conducted. Sonar frequencies likely range ten's of kilohertz to hundreds of kilohertz. Radar operates in the megahertz range. So, it is common for the user to fake the units on the time sampling interval so as to fit the requirements of a seismic code.

5.6 The sonar data

The "sonar.su" file is one of the profiles collected by Dr. Henrique Tono of Duke University in a special laboratory setting.

According to a personal communication by Dr. Tono, the "geologic setting" of the sonar data is thus

> "The deposits and images were produced at the Saint Anthony Falls Lab of the University of Minnesota. Here, experimental stratigraphy is produced

> under precisely controlled conditions of subsidence, base level, and sediment supply. By superimposing optical images of the sectioned deposits on seismic images, we can directly observe the ability of seismic profiling to distinguish different geological features.
>
> The experimental basin is 5 m by 5 m (25 m2) and 0.61 m deep. Sediment and water were mixed in a funnel and fed into the basin at one corner. This produced an approximately radially-symmetrical fluvial system, which averaged 2.50 m from source to shoreline. The edges of the basin were artificially roughened in order to direct the channels away from the walls. The "ocean level" was maintained through a variable-discharge siphon located in the opposite corner of the basin. Though we imposed a gradual base-level rise, in order to simulate subsidence, the shoreline maintained a constant position through the experiment."

Dr. Tono goes on to describe the experimental layout:

> "The outgoing pulse is generated with a Prototype JRS DPR300 (Pulser/Receiver), which drives a 900-volt square pulse into the transducer. It is set to a pulse/receive frequency of 100 Hz, with an input gain of 30 dB in echo mode. The high pass filter is set at 20 KHz, and the low pass filter at 10 MHz. A Gage-Applied Compuscope 1602 digitizer computer card (16 Bit, 2 Channel card with acquisition memory of 1 Million samples) is used to perform the A/D conversion, and the data is displayed on a computer screen by means of GageScope 3.50. It is digitally recorded on the computer hard disk. A sample rate of 2.5 MS/s is chosen (Nyquist frequency=1.25 MHz). It is then re-formatted to SEG-Y and processed with Seismic Unix.
>
> The data were acquired with a $5mm$ shotpoint and station interval (zero offset), and $1cm$ separation between lines."

In the directory /data/cwpscratch/Data1 you will find a number of JPEG format files depicting the experimental setting described by Dr. Tono.

The file "dsc01324.su" is an SU format file version of the image DSC01324.JPG, cropped to remove parts of the image that are not the cross section. This is not exactly the cross section of the data sonar.su, but it gives the idea. Rarely, are we able to slice into the actual model in this fashion.

5.7 Homework Problem - #2 - Time-to-depth conversion of the sonar.su and the radar.su data. Due Thursday 10 September 2015 and Tuesday 15 September 2015, for the respective sections

Find the necessary velocities to permit the correct time-to-depth conversion of the **sonar.su** and **radar.su** data. You will need to figure out the appropriate units, because it is not possible for these non-seismic datasets to have an accurate representation of the time sampling interval represented in the trace header field **dt**. Make sure that you give a justification explaining why your choice of the appropriate power of 10 scaling factor is likely the correct one. Remember that the depth scale on you output data should make sense.

5.8 Concluding Remarks

When receiving software, either that is given to us, or that which we purchase, it is important to try to figure out what assumptions are built into the package. One way to do this is to apply the software to test data.

As applied scientists and engineers, we are often in in situations where we are forced to use a tool that is not quite right for the job. It is not uncommon for laboratory experimentalists or ground penetrating radar practitioners to use seismic processing software to do part of the analysis of their (non-seismic) data. We must be careful to keep the problem simple and expect only what we deserve from the data.

When receiving data, it is important to know everything that you can possibly know about the data, such as the spacing of the traces, the time sampling interval, any processing that has been applied, and any redifinition of header values.

5.8.1 The sonar - seismic analogy

When explaining seismic imaging to non-geophysicists, it is tempting to say that seismic imaging is "sort of like sonar". However, sonar is a rough-surface-scattering based imaging method, whereas seismic imaging is a "specular" or mirror-reflection-scattering imaging. In rough-surface scattering, the image may, indeed, be formed by "straight down and straight back" reflections. In seismic, this is rarely the case. We must take offset between the source and receiver into account. In seismic methods, there are also rough-surface contributions. These are the diffractions that we look for on stacked data.

In the early days of seismic prospecting (c. 1930s) there were practitioners of the seismic method who thought that seismic was the same as sonar and thus expected that seismic datasets should be "data images." Such phenomena as "bowties" provide a clue that reflection seismic is not the same as sonar.

Chapter 6

Zero-offset (aka poststack) migration

The first reflection seismic experiment as applied to petroleum exploration was conducted by physicist John Clarence Karcher in Oklahoma in 1921 (Schriever) in conjunction with Marland Oils, which would later become CONOCO. Oklahoma was the center of the US oil industry at that time.

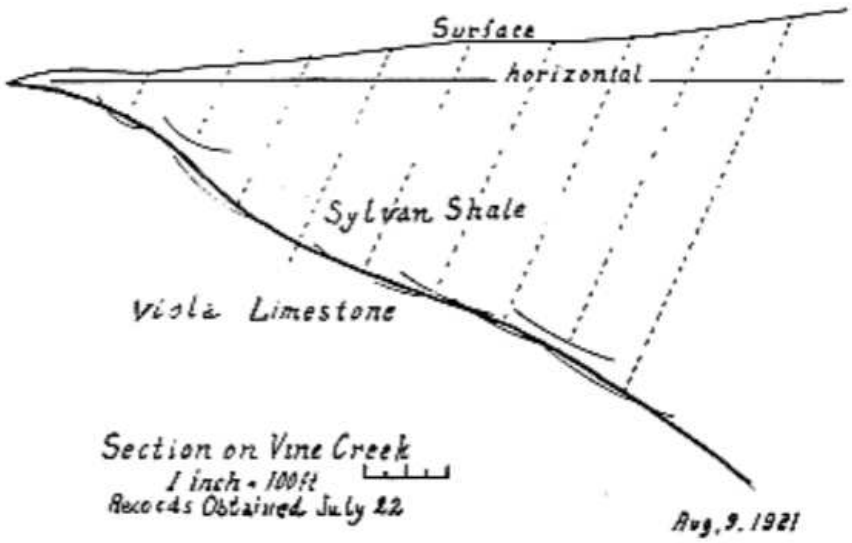
While it is clear from reading documents from that era that the expectations of some practicioners of reflection seismic methods were that the results should be similar to sonar, it is clear from a figure in Karcher's report from 1921, that he and others were aware of the geometry of reflection, including a notion of migration, see Figure 6.1.

By the 1930s most geophysicists were well aware of the geometrical issues at the heart of proper seismic interpretation. With the formation of the Society of Exploration Geophysicists in 1930, followed by the first issue of the Society's journal *Geophysics* the proper usage of seismic data for geologic interpretation became known to the geophysical community.

In Figure 6.2 we see the classical "bowtie" feature seen over a syncline. To the early interpreter of seismic data, this diagram would not have constituted an image of the subsurface, but rather a source of geometrical data (such as dip) pertaining the subsurface reflector.

Another notion that became apparent is that parts of the data on the seismic traces is displaced from its "correct" position by the properties of wave propagation. Assuming that all reflections are normal incidence for this zero-offset geometry, it is clear that parts of the bowtie originate from higher positions on the sides of the syncline. Thus, the notion of "migrating" those arrivals to their correct location became an important idea for interpretation. Because the seismic data were analog rather than digital, such corrections would naturally be applied graphically.

While graphical migration techniques had been applied since the 1930s, the first notable technical paper describing this technique was published by J. G. (Mendel) Hagedoorn in 1954. This paper is important because Hagedoorn's description of the migration process inspired early digital computer implementations of migration.



Depth of the Viola limestone at Vines Branch was measured with reflection seismograph on August 9, 1921—world's first reflection seismograph geologic section.

Figure 6.1: Geometry of Karcher's prospect, note semicircular arcs indicating that Karcher understood the relation of surfaces of constant traveltime to what is seen on a seismogram.

6.1 Migration as reverse time propagation.

One way of looking at migration is as a reverse time propagation. The idea may be visualized by running the output from a forward modeling demo in reverse time. Do the following, noting that you need to replace "yourusername" with your actual username on the system, so that the items are copied to your personal scratch area

```
$ cp /data/cwpscratch/Data1/syncline.unif2 /gpfc/yourusername/Temp1
$ cp /data/cwpscratch/Data1/XSyncline /gpfc/yourusername/Temp1
$ more XSyncline
$ more syncline.unif2
```

Now

```
$ cd /gpfc/yourusername/Temp1
```

If you type:

```
$ more syncline.unif2
   0         0
4000         0
   1    -99999
   0       1000.
```

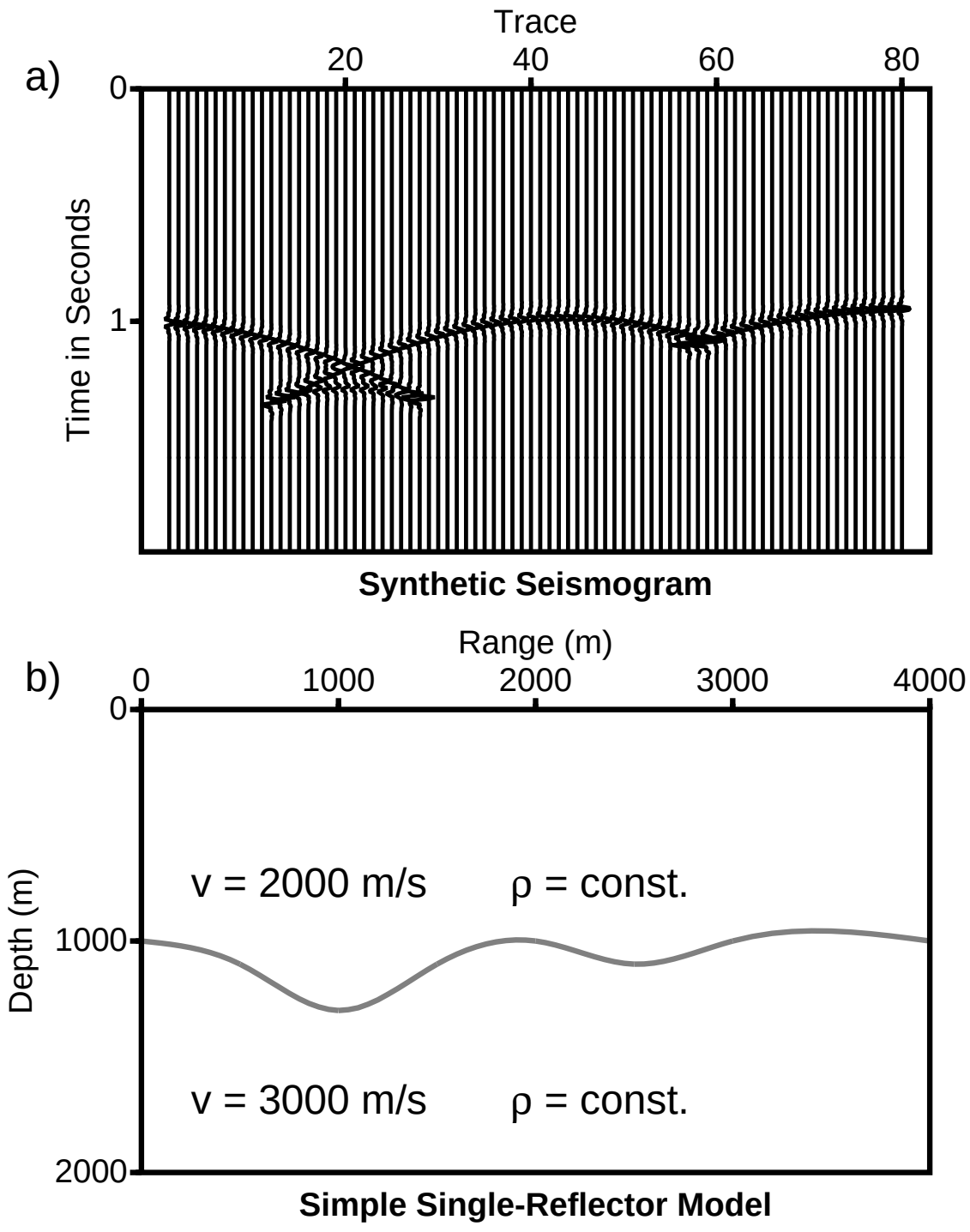


Figure 6.2: a) Synthetic Zero offset data. b) Simple earth model.

```
 500.      1100.
 1000.     1300.
 2000.     1000.
 2700.     1100.
 3200.     1000.
 4000.     1050.
   1.   -99999
```

you will see the input data for a wavespeed profile building program called **unif2**. The contents of this file define two boundaries in a velocity model. The data for the two boundaries is separated by the values

```
   1.   -99999
```

The values in the column on the left are horizontal positions and the values on the right are depths. This model defines the same simple syncline model seen in Fig 6.2. We now look at the contents of the shell script **XSyncline**

```
$ more XSyncline

#! /bin/sh
# Shell script to build velocity profiles with unif2

# input parameters
modelfile=syncline.unif2
velfile=syncline.bin
n1=200
n2=400
d1=10
d2=10

# use unif2 to build the velocity profile
unif2 <$modelfile method=$i ninf=2 nx=$n2 nz=$n1 v00=1000,2000 \
ninf=1 method=spline  > $velfile

# view the velocity profile on the screen
ximage < $velfile  wbox=400 hbox=200 n1=$n1 n2=$n2 d1=$d1 d2=$d2 \
 wbox=800 hbox=400  legend=1 title="Syncline model" label1="depth m" \
label2="distance m " units="m/s" &

# provide input for sufdmod2
xs=1000 zs=10 hsz=10 vsx=1000 verbose=2
vsfile="vseis.su" ssfile="sseis.su" hsfile="hseis.su"
tmax=3.0 mt=10
label1="Depth m"
```

```
label2="Distance m"

# perform finite difference acoustic modeling to generate data
# for a single shot in the
sufdmod2 < $velfile nz=$n1 dz=$d1 nx=$n2 dx=$d2 verbose=1 \
        xs=$xs zs=$zs hsz=$hsz vsx=$vsx hsfile=$hsfile \
        vsfile=$vsfile ssfile=$ssfile verbose=$verbose \
        tmax=$tmax abs=1,1,1,1 mt=$mt |
suxmovie  clip=1.0 \
        title="Acoustic Finite-Differencing" \
        windowtitle="Movie" \
        label1=$label1 label2=$label2 \
        n1=$n1 d1=$d1 f1=$f1 n2=$n2 d2=$d2 f2=$f2 \
        loop=1 sleep=.8 &

exit 0
```

You may run the demo by typing:

```
$ XSyncline
```

The result shows the wavespeed profile for the model. This is similar to the "simple" model that will be discussed later in the these notes. A movie showing snapshots of the wavefield will begin. Watch the wavefront of the energy from the shot expand. You may stop and restart the movie by pressing the far right mouse button. Of interest are the frames at which the first reflections begin. As the movie progresses, you will see the reflected field progress as the reflection point propagates along the reflector surface. Indeed, from viewing this movie, we can see why an integral over the reflector surface, called the "Kirchhoff modeling formula" is a way of modeling the reflected field.

Note that you only see wavefronts, there is nothing like a "ray" to be seen. A ray is the trajectory taken by a point on a wavefront. Second, notice that the "bowtie" forms as the caustic in the propagating wavefield travels to the surface.

The movie will run in a loop. You may stop the movie by pushing the right mouse button. You may reverse the movie by pressing the middle mouse button. Effectively, running the field backward in time is "reverse-time migration." In seismic data, we do not have a record of the down-traveling field. All we have is the record of that part of the reflected field that hits the surface of the earth where there are geophones. The migration process finds the place where the downward travelling field and the reflected field overlay—the reflector surface. One way of looking at migration is that we would like to cross-correlate the down-traveling field with the time-reversed reflected field. The place where these fields correlate is at the reflector surface.

You may also see what the seismic data looks like recorded at the surface of the earth model by viewing the file **hseis.su** via

```
$ suximage < hseis.su perc=99
```

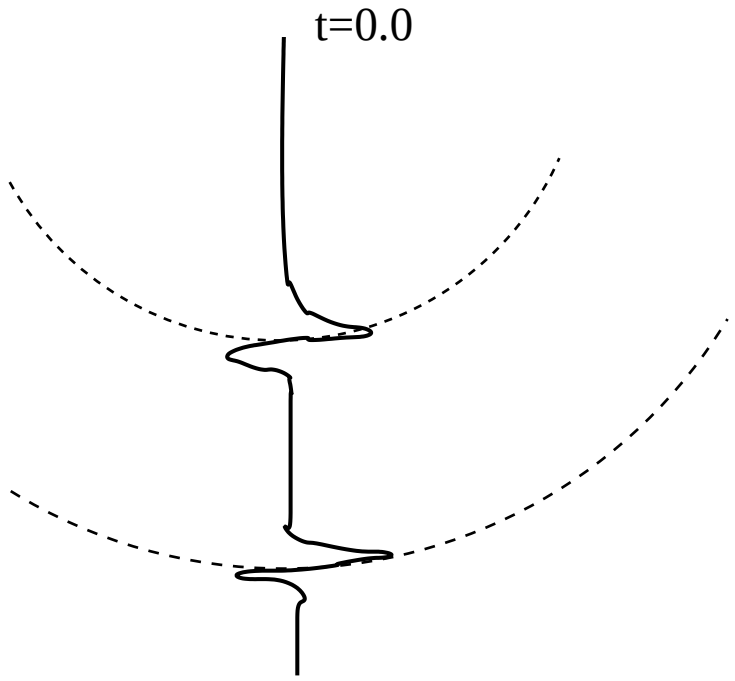


Figure 6.3: The Hagedoorn method applied to the arrivals on a single seismic trace.

to see the direct arrival and the bowtie-shaped reflected arrival.

Finally you may change the depths in the model by editing the file **syncline.unif2**, or change the location of the source to see what varying these quantities changes in the data. You may slow down the movie by increasing the value of the **sleep=** parameter.

6.2 Lab Activity #5 - Hagedoorn's graphical migration

The purpose of this lab example is to migrate the simple data in Figure 6.2a) by Hagadoorn's graphical method. These synthetic data represent the zero-offset reflection seismograms recorded over the undulating reflector model in Figure 6.2b). The wavespeed in the upper medium is assumed to be 2000 m/s, and the data are drawn in such a way that 1.0 s two way time is equivalent to 1000 m of distance. Thus, the time scale translates into a depth scale that is compatible with the horizontal scale.

If we draw a circle centered at time $t = 0$ of a given seismic trace and passing through a given seismic arrival, we have sketched all possible reflection points from which the seismic arrival could have originated. These circles are the same as the incident field seen in the seismic movie. If we recall, seismic migration finds the place where the incident field interacts with the reflected field—the reflector surface.

When similar circles are drawn for every arrival on every trace, the result is a collection of circles whose envelope delineates the reflector. See Fig 6.4 for an idea of what this should look like.

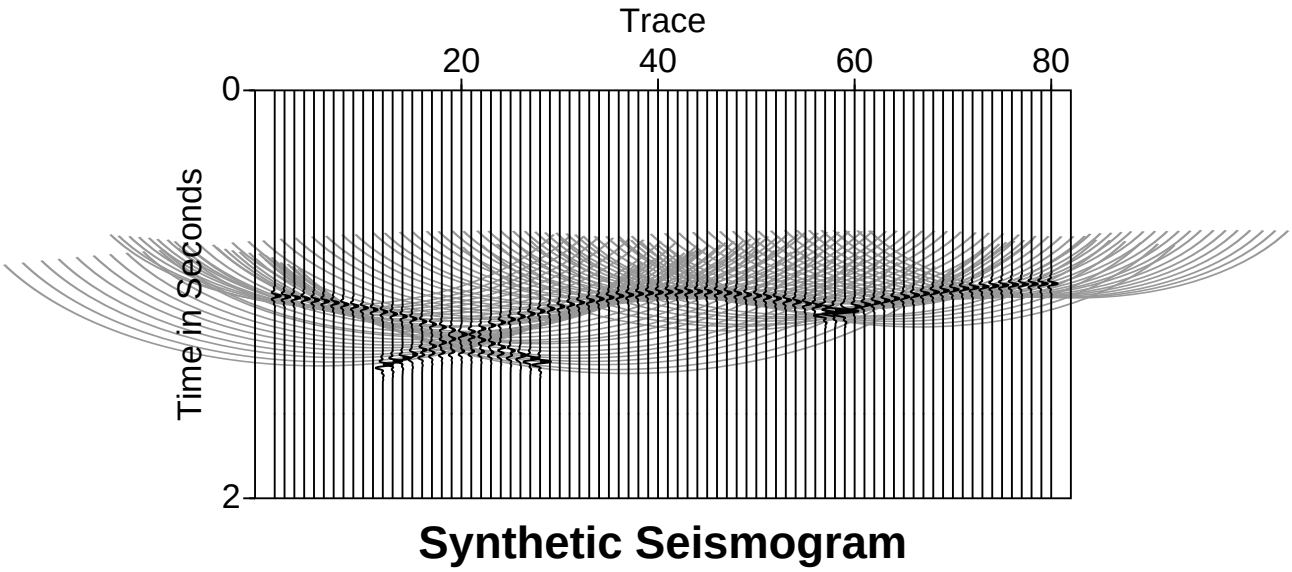


Figure 6.4: Hagedoorn's method applied to the simple data of Fig 6.2. Here circles, each centered at time $t = 0$ on a specific trace, pass through the maximum amplitudes on each arrival on each trace. The circle represents the locus of possible reflection points in (x, z) where the signal in time could have originated.

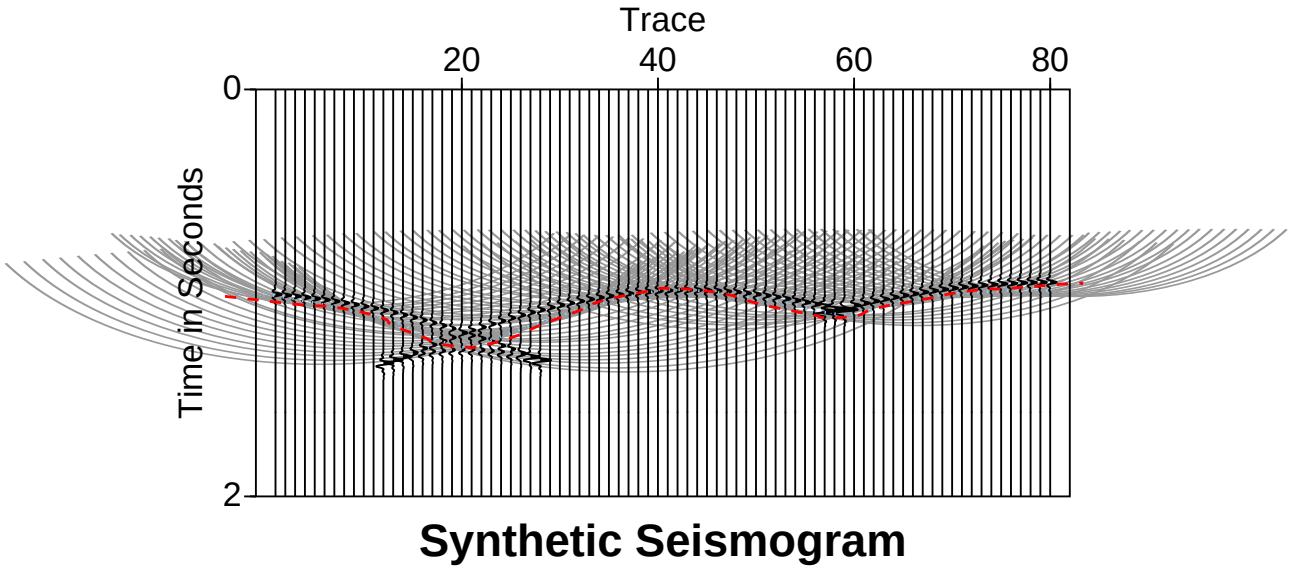


Figure 6.5: The dashed line is the interpreted reflector taken to be the envelope of the circles.

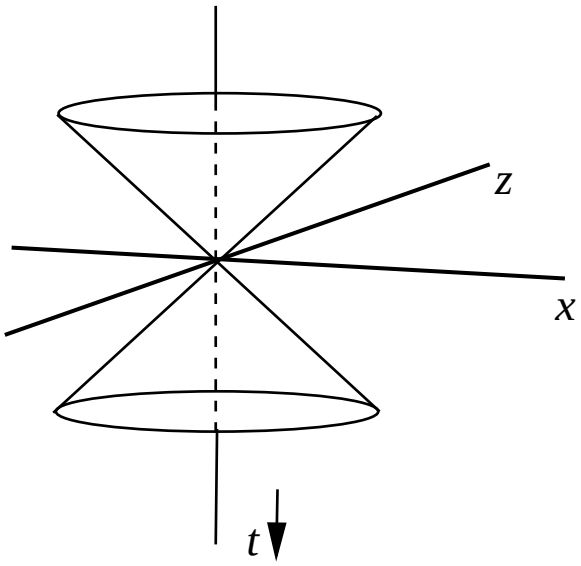


Figure 6.6: The light cone representation of the constant-velocity solution of the 2D wave equation. Every wavefront for both positive and negative time t is found by passing a plane parallel to the (x, z)-plane through the cone at the desired time t. We may want to run time backwards for migration.

Mathematically this method of migration may be thought of as the reconstruction of the reflector by defining the *tangent vectors* of the reflector. What then, are the circles we have drawn? The answer can by found by looking at Figure 6.6. For our 2D constant-wavespeed example, all solutions of the wave equation, which is to say *all wavefronts*, can be found by passing a horizontal plane through the cone in Figure 6.6. Both physical (causal) solutions (the positive t cone) and the nonphysical (anti-causal) solutions (the negative t cone) are depicted. We use the causal cone for modeling, and the anti-causal or reverse-time cone for migration.

To see what a given circle means in Hagadoorn's method, we may look at the reverse time cone in Figure 6.7. We may think of the curve on the $t = 0$-plane as the locus of all possible positions from which the reflection originated, or we may think of this as the wavefront of the backward-propagated wave.

If we were to apply the Hagedoorn method on the computer, we might consider creating for each seismic trace a panel of seismic traces replicating our original seismic arrivals, but on a semicircular pattern. "Spraying" out our seismic data for each trace along the respective Hagedoorn circle would yield one new panel of traces for each seismic trace. Our 80 traces would then become 80 panels of sprayed traces. We would then sum the corresponding traces on each panel. Constructive interference would tend to enhance the region near the reflector, and destructive interference would tend to eliminate everything else, revealing only the reflector. Does this method work? Yes, but it is subject to interference errors, if the data are not densely sampled in space.

Because a point at (ξ, τ) represents an impulse in the (x, t) space, corresponding

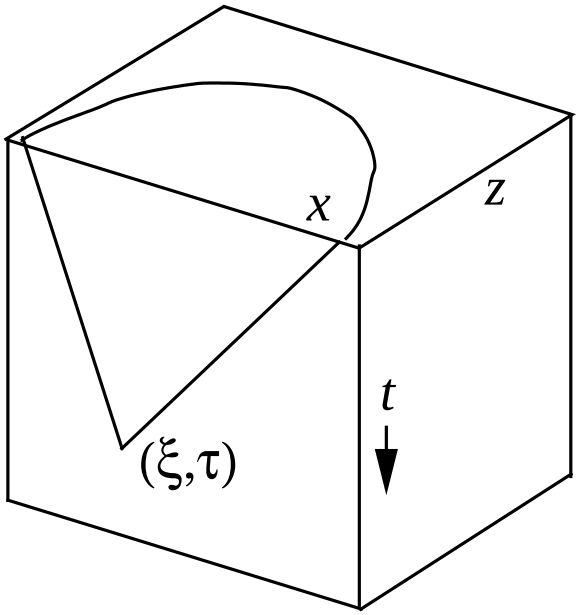


Figure 6.7: The light cone representation for negative times is now embedded in the (x, z, t)-cube. A seismic arrival to be migrated at the coordinates (ξ, τ) is placed at the apex of the cone. The circle that we draw on the seismogram for that point is the set of points obtained by the intersection of the cone with the $t = 0$-plane.

circle drawn in Hagadoorn's method may be thought of as the *impulse response* of the migration operation.

6.3 Migration as a Diffraction stack

Another approach to migration is immediately apparent. If we apply Hagedoorn's method to the diffraction from a point scatterer, then we observe that the scatterer is reconstructed. However, tangent vectors are not defined with regard to a point scatter. Instead, it must be the *ray vector from the source/receiver position to the scatterer* that is being reconstructed. In other words, the *reflected ray vector* is the distinguished vector associated with the imaging point. For a reflector surface, this is the *perpendicularly-reflected ray vector*. See Figure 6.8.

Furthermore, we might ask: why is it necessary to draw Hagedoorn's circles at all? Suppose that we were to sum over all possible diffraction hyperbolae. Then the largest arrivals would exist only where a hyperbola we sum on hits a hyperbola in the data. The sum would then be placed at a point at the apex of the hyperbola passing through our data. This type of migration is referred to as a *diffraction stack*. We sum or "stack" data, but we do this over a diffraction curve. Furthermore the output need not be a depth section, but could be a *time section*.

A useful diagram for understanding the diffraction stack is the light cone diagram in Figure 6.9. A *light cone* is the representation of the surface where solutions of the wave

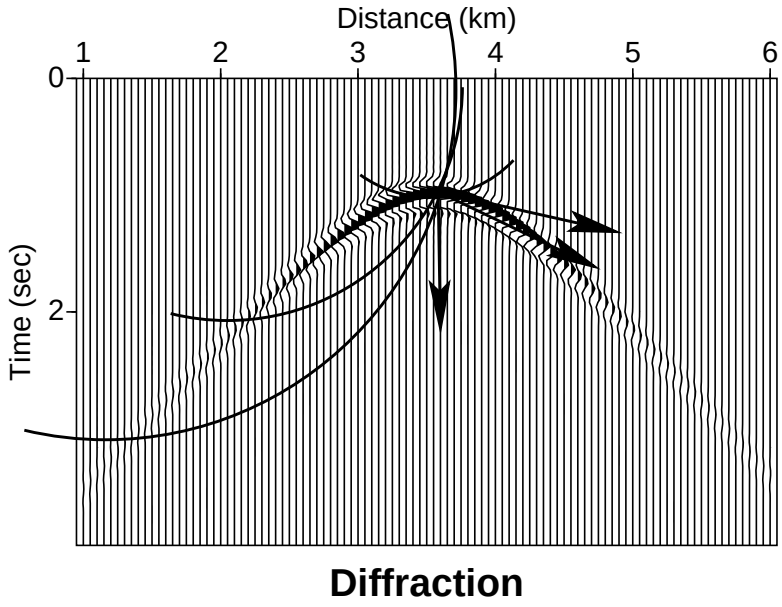


Figure 6.8: Hagedoorn's method of graphical migration applied to the diffraction from a point scatterer. Only a few of the Hagedoorn circles are drawn, here, but the reader should be aware that any Hagedoorn circle through a diffraction event will intersect the apex of the diffraction hyperbola.

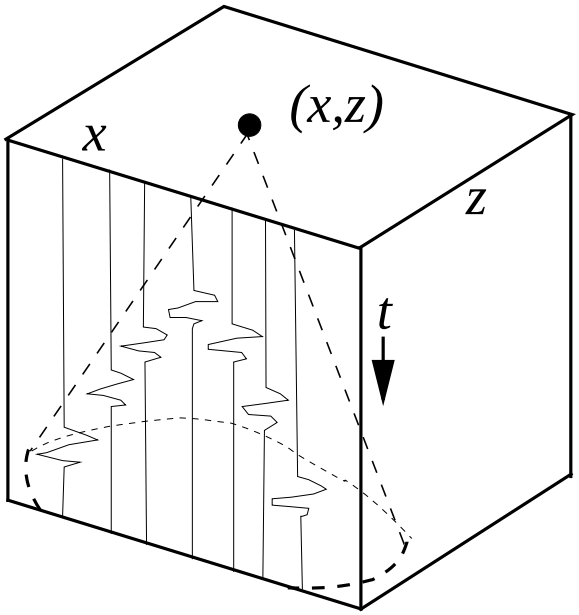


Figure 6.9: The light cone for a point scatterer at (x, z). By classical geometry, a vertical slice through the cone in (x, t) (the $z = 0$ plane where we record our data) is a hyperbola. Time migrations collapse diffraction hyperbolae to their respective apex points. Depth migrations map these apex points into the (x, z) (2D) plane.

equation live. The scatterer is located at the point (x, z). Time increases downward. A horizontal slice through the cone reveals the circular wavefronts that are the circles drawn in Hagedoorn's method. A vertical slice through the cone in (x, t) reveals the hyperbola that is the characteristic shape of a diffraction in a constant wavespeed medium.

6.4 Migration as a mathematical mapping

Another diagram that reveals migration as a type of data transformation or *mapping* may be seen in Figure 6.10. Here, we see that the impulse response of the migration operator is a circular curve in constant wavespeed media.

The diffraction in (x, t) may be thought of as the "impulse response" of the modeling operation that made the data from a point at (x, z). Migration by diffraction stack, therefore, consists of selecting a point (x, z), modeling the diffraction curve in (x, t), and then summing through the data over this curve. Note that this must be done for every output point to make the image.

Figure 6.10 represents more than migration. Going from **a)** to **b)** is Hagedoorn's migration method. Going from **c)** to **d** is the diffraction stack migration method. If however, we reverse directions, mapping from from **b)** to **a)** or from **d)** to **c** then we are *modeling* or doing data-based **de-migration**, which is the inverse of migration. The idea then is that modeling is the forward process and migration is the inverse operation.

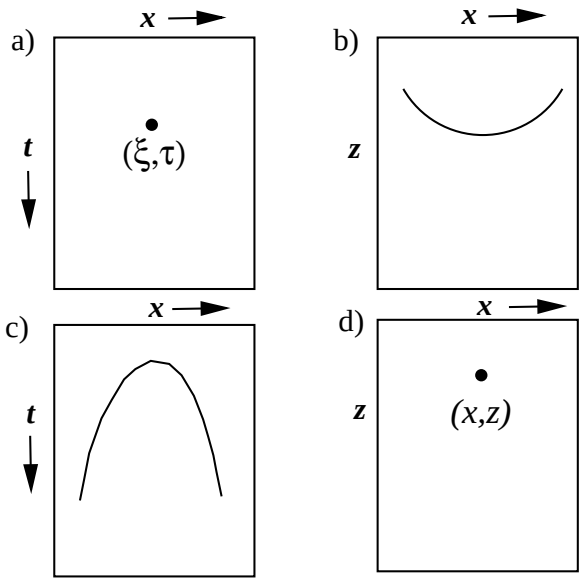


Figure 6.10: Cartoon showing the relationship between types of migration. a) shows a point in (ξ, τ)j, b) the impulse response of the migration operation in (x, z), c) shows a diffraction, d) the diffraction stack as the output point (x, z).

6.5 Concluding Remarks

The notion of the value and motivation of using seismic data has changed through the history of seismic methods. Originally, seismic data were used to find an estimate of perhaps only a single reflector. As the technique developed, the depth to and dip of a specific target reflector was found. Most notably was Frank Rieber's "dip finder." The "dip finder" was a recording system that was effectively an analog computer that delivered an estimate of depth and dip for stronger reflectors. These data were then used for drawing geologic cross-sections. In fact, Frank Rieber's dip finder was doing something similar to a variety of migration called "map migration."

As the petroleum and natural gas industry evolved, so did the importance of the seismic method. The technique started out as an aid in interpretation, becoming later an "imaging technology." Today, seismic migration is viewed by many as the "solution to an inverse problem" wherein recorded seismic data are used as input to solve for the reflectivity of the reflectors, as well as other important material parameters that characterize lithology.

References

Schriever, W. (1952). "REFLECTION SEISMOGRAPH PROSPECTINGHOW IT STARTED." 'GEOPHYSICS", 17(4), 936-942

Bleistein, N., J. K. Cohen, and J. W. Stockwell, Jr. (2001) "Mathematics of multidimensional seismic imaging, migration, and inversion" Springer Verlag, New York.

Chapter 7

Lab Activity #6 - Several types of migration

In this assignment we will apply several types of migration to the simple.su data. These types of migration represent many of those that are commonly used in industry.

7.1 Different types of "velocity"

All seismic migrations require a background wavespeed (velocity) profile. However, we must be very careful when addressing the term "velocity." The actual wavespeeds in the subsurface are called **interval** velocities, and most likely are going to be a function of position.

However, the types of velocities that we often encounter, such as those obtained from velocity analysis are **stacking** velocity, also known as the **NMO** velocity, which is approximately the **RMS (root mean squared)** velocity. Such velocities are often expressed as *velocity as a function of time*.

Having a velocity that is a function of time may seem strange, at first, but imagine that you have a seismic section, with several strong seismic horizons. These strong arrivals must represent relatively large impedence contrasts. If we had a set of well logs to go with the seismic data, then we could identify those rock units with the strong impedence contrasts. We could build a collection of velocities chosen from the well log, and assign the times from the arrival time on the seismic section and thus we would define a $v(t)$. If we have $v(t)$ for specific locations in the rock volume we are investigating we could assemble a $v(t,x)$ or a $v(t,x,y)$ through some sort of interpolation.

7.1.1 Velocity conversion $v_{rms}(t)$ to $v_{int}(t)$

The conversion between these two is given by the Dix equation

$$v_{int} = \left[\frac{(t_2 v_{rms2}^2 - t_1 v_{rms1}^2)}{(t_2 - t_1)} \right]^{1/2}$$

where v_{int} is the interval velocity, t_1 is the traveltime to the first reflector, t_2 is the traveltime to the second reflector, v_{rms1} is the root-mean-squared velocity of the first reflector, and v_{rms2} is the root-mean-squared velocity of the second reflector. The RMS (root mean squared) part comes fromt the fact that we are effectively averaging and taking the square root of squares going from RMS to interval velocity.

Often the output of a migration is horizontal position versus "migrated time." This implies that the input velocity is expressed as $v(t)$, rather than $v(x, z)$. We may have interval or rms velocities as a function of time, e.g. $v_{int}(t)$ or $v_{rms}(t)$. This may seem strange at first, but if we have $v(t)$ profile, then $v(z)$ is obtained by stretching time to depth.

In SU a program called **velconv** performs many simple conversions between interval and rms velocities. Type **velconv** with no options to see the selfdoc of the program

```
$ velconv

 VELCONV - VELocity CONVersion

 velconv <infile >outfile intype= outtype= [optional parameters]

 Required Parameters:
 intype=                 input data type (see valid types below)
 outtype=                output data type (see valid types below)

 Valid types for input and output data are:
 vintt          interval velocity as a function of time
 vrmst          RMS velocity as a function of time
 vintz          velocity as a function of depth
 zt             depth as a function of time
 tz             time as a function of depth

 Optional Parameters:
 nt=all                  number of time samples
 dt=1.0                  time sampling interval
 ft=0.0                  first time
 nz=all                  number of depth samples
 dz=1.0                  depth sampling interval
 fz=0.0                  first depth
nx=all                  number of traces

 Example:  "intype=vintz outtype=vrmst" converts an interval velocity
           function of depth to an RMS velocity function of time.

 Notes:  nt, dt, and ft are used only for input and output functions
```

```
        of time; you need specify these only for vintt, vrmst, orzt.
        Likewise, nz, dz, and fz are used only for input and output
        functions of depth.

The input and output data formats are C-style binary floats.
```

7.2 Stolt or (f, k)-migration

To migrate the simple data in the computer we first begin with Stolt migration. Stolt's method, published in 1978, is a migration by Fourier transform and is often called (f, k) migration. If we consider migration to be a "shifting of data," which is expressed as a signal processing technique, then we may consider that shifting to be done as a filtering process. The data are shifted both temporally and spatially, suggesting that the filter doing the shifting must be a filter that operates in both the frequency and wavenumber domain. If the velocity function is variable (with time) the Stolt method accounts for this by applying a stretch to the data prior to the filtering operation.

To get the shifting correct, Robert Stolt based his "shifting filter" on an integral equation representation of the wave equation, and made use of the *fast Fourier transform* algorithm for speed. To handle variable wavespeed Stolt introduced a "stretch," much as we scaled the time section to appear interchangeable with depth.

7.2.1 Stolt migration of the Simple model data

Here we apply Stolt migration to the **simple.su** data. Copy the dataset into your local working directory via:

```
$ cd  /gpfc/yourusername
$ mkdir Temp2
$ pwd                (you should be in /gpfc/yourusername)
$ cp  /data/cwpscratch/Data2/simple.su  Temp2
```

changing your working directory to Temp2 via:

```
$ cd Temp2
```

You may view the data via:

```
$ suxwigb < simple.su xcur=3 title="Simple"
```

The Stolt migration is performed via the program sustolt. To see the self documentation for sustolt type:

```
$ sustolt
```

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

 sustolt <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)

 Optional Parameters:
 noffmix=1              number of offsets to mix (for unstacked data only)
 tmig=0.0               times corresponding to rms velocities in vmig (s)
 vmig=1500.0            rms velocities corresponding to times in tmig (m/s)
...
```

Among the many parameters, those that are required are cdpmin, cdpmax, dxcdp, vmig and tmig. Note: the type of velocities, vmig, that this program requires are **RMS (or stacking) velocities as a function of time**! To see the range of the cdps in the data, type

```
$ surange < simple.su
80 traces:
tracl    1 80 (1 - 80)
cdp      1 80 (1 - 80)
trid     1
ns       501
dt       4000
```

Thus, cdpmin=1 and cdpmax=80. The data were made with a $40m$ spacing, and with a velocity in the first medium of $2000m/s$. Applying sustolt

```
$ sustolt < simple.su cdpmin=1 cdpmax=80
                   dxcdp=40 vmig=2000 tmig=0.0 > stolt.simple.su
```

We may view the output of Stolt migration via

```
$ suxwigb < stolt.simple.su xcur=3 title="Stolt migration of simple data" &
```

The migrated data look similar in many ways to the graphical migration. These data have been chosen to be too sparsely sampled spatially to be considered "good." This choice was made on purpose to accentuate the artifacts on the data. These artifacts are known by the terms diffraction smiles, migration impulse responses, or endpoint contributions. The fact that the data are sparse makes the migration operator see the individual arrivals as single spikes, resulting in impulse responses on the output. There is insufficient coverage.

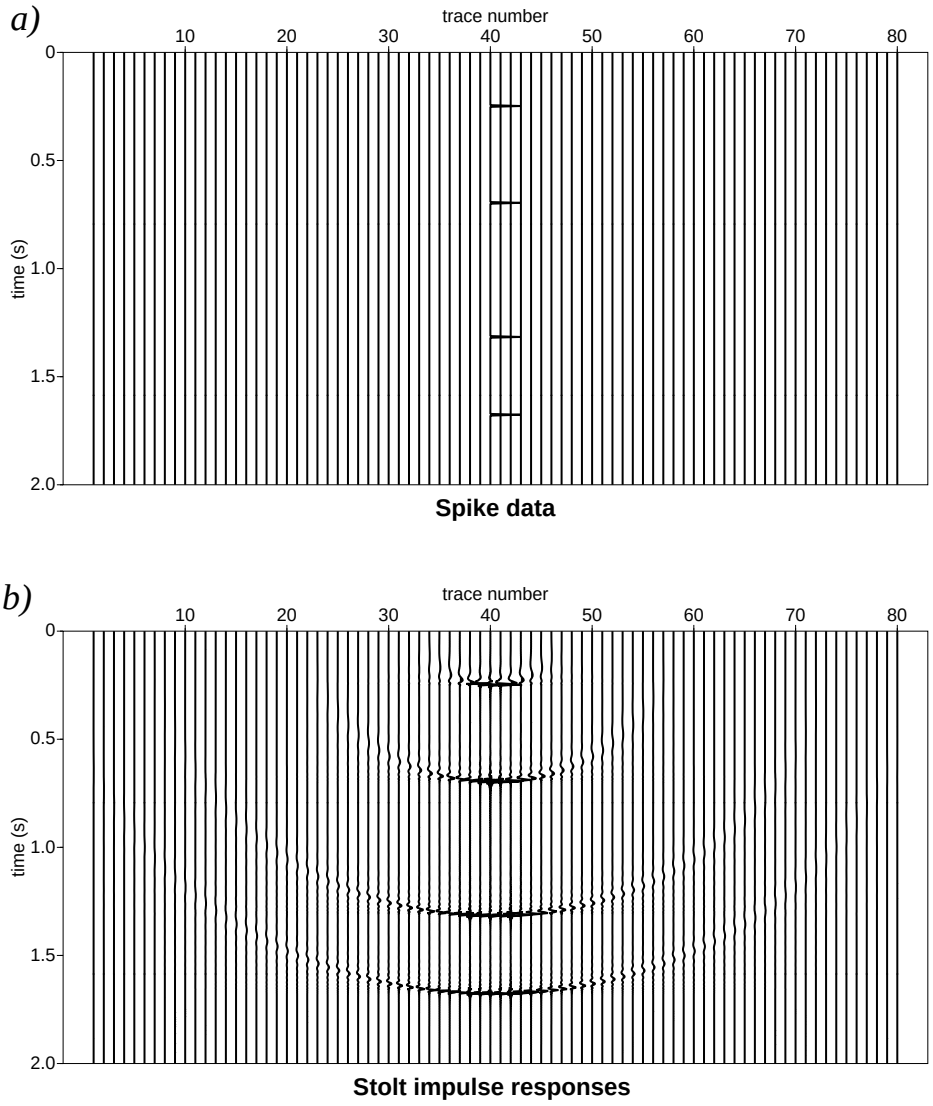


Figure 7.1: a) Spike data, b) the Stolt migration of these spikes. The curves in b) are *impulse responses* of the migration operator, which is what the curves in the Hagadoorn method were approximating. Not only do the curves represent every point in the medium where the impulses could have come from, the amplitudes represent the strength of the signal from that respective location.

Diffraction smiles as impulse responses

An effective way of thinking about many noise artifacts on migrated data is to note that migration operators expect smooth input. If the input is not smooth, such as if there is a noise spike or an abrupt termination of the data, this is similar to putting an impulse into the algorithm. The result is an *impulse response*.

By applying **suspike** we may make data consisting only of 4 nonzero data values (spikes). The data are otherwise the same number of samples (501) and the same number of traces (80) as **simple.su**

```
#! /bin/sh


$ suspike nspk=4 nt=501 ntr=80 dt=.004 ix1=40 it1=63 ix2=40 \
             it2=175 ix3=40 it3=40 it3=330 ix4=40 it4=420  \
                       | sushw key=cdp a=1 b=1 > spike_simple.su
```

and we apply **sustolt** with the same parameters as we used on the **simple.su** data

```
$ sustolt < spike_simple.su vmig=2000 tmig=0.0
                          dxcdp=40 cdpmin=1 cdpmax=80 > stolt.spike.su
```

shown in Figure 7.1

7.3 Gazdag or Phase-shift migration

In 1978 Jeno Gazdag published a type of migration that also makes use of the shift theorem. In this class of method, the model is discretized in such a way that the vertical direction z is considered to be preferred. The data are Fourier transformed, and the migration is applied as a phaseshift, to each wavenumber. The data are then inverse Fourier transformed.

The program **sugazmig**

```
$ sugazmig

 SUGAZMIG - SU version of Jeno GAZDAG's phase-shift migration
             for zero-offset data, with attenuation Q.

 sugazmig <infile >outfile vfile= [optional parameters]

 Optional Parameters:
 dt=from header(dt) or  .004    time sampling interval
 dx=from header(d2) or 1.0      midpoint sampling interval
```

```
 ft=0.0                  first time sample
 ntau=nt(from data)      number of migrated time samples
 dtau=dt(from header)    migrated time sampling interval
 ftau=ft                 first migrated time sample
 tmig=0.0                times corresponding to interval velocities in vmig
 vmig=1500.0    interval velocities corresponding to times in tmig
...
```

Try the Gazdag migration of the **simple.su** data via:

```
$ sugazmig < simple.su dx=40 vmig=2000 tmig=0.0 > gaz.simple.su
```

Note: the velocities, vmig, here are **interval velocities as a function of time**.

You will notice that you could do several Stolt migrations in the time that it takes to do a single Gazdag.

By the application of a slightly different formulation of phase-shift migration, the performance, as well as the accuracy at steeper dips of Gazdag migration can be improved. The result is Dave Hale's **sumigps**

```
$ sumigps

 SUMIGPS - MIGration by Phase Shift with turning rays

 sumigps <stdin >stdout [optional parms]

 Required Parameters:
        None

 Optional Parameters:
 dt=from header(dt) or .004     time sampling interval
 dx=from header(d2) or 1.0      distance between successive cdps
 ffil=0,0,0.5/dt,0.5/dt  trapezoidal window of frequencies to migrate
 tmig=0.0               times corresponding to interval velocities in vmig
 vmig=1500.0            interval velocities corresponding to times in tmig
 vfile=         binary (non-ascii) file containing velocities v(t)
 nxpad=0                number of cdps to pad with zeros before FFT
 ltaper=0               length of linear taper for left and right edges
 verbose=0              =1 for diagnostic print

...
```

Try an improved phase-shift migration:

```
$ sumigps < simple.su dx=40 vmig=2000 tmig=0.0 > ps.simple.su
```

7.4 Claerbout's finite-difference migration

Another "backpropagation" approach was taken by Jon Claerbout in 1970 via finite-difference solution of a one-way approximate wave equation. This is a reverse-time finite difference migration, but it is not the exact wave equation. The "15 degree" part refers to the angle of a cone about vertical, within which the traveltime behavior of the migration is sufficiently similar to that of the standard wave equation as to yield an acceptable result. Other approximations for increasingly larger angles have been created. An implementation of a version of this finite-difference migration that gives the user a choice of different "angles of accuracy" is **sumigfd**

```
 SUMIGFD - 45-90 degree Finite difference migration for zero-offset data.

   sumigfd <infile >outfile vfile= [optional parameters]

 Required Parameters:
 nz=             number of depth sapmles
 dz=             depth sampling interval
 vfile= name of file containing velocities
                 (see Notes below concerning format of this file)

 Optional Parameters:
 dt=from header(dt) or .004    time sampling interval
 dx=from header(d2) or 1.0      midpoint sampling interval
 dip=45,65,79,80,87,89,90       Maximum angle of dip reflector
...
```

Try:

```
$ cp /data/cwpscratch/Data2/vel.fdmig.simple /gpfc/yourusername/Temp2
$ cd /gpfc/yourusername/Temp2
$ sumigfd < simple.su dx=40 dz=12 nz=150 vfile=vel.fdmig.simple  > fd.simple.su
```

Note: the velocity file (vfile) expects **interval velocities as a function of** (x, z), where x is taken as the fast dimension in this file.

7.5 Ristow and Ruhl's Fourier finite-difference migration

A hybridization between phase-shift migration and finite-difference migration known as "Fourier finite difference" was published in 1994 by D. Ristow and T. Ruhl. This type of migration is implemented in **sumigffd**

```
 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.

 sumigffd <infile >outfile vfile= [optional parameters]

 Required Parameters:
 nz=               number of depth sapmles
 dz=               depth sampling interval
 vfile=         name of file containing velocities

 Optional Parameters:
 dt=from header(dt) or .004    time sampling interval
 dx=from header(d2) or 1.0     midpoint sampling interval
 ft=0.0                 first time sample
 fz=0.0                 first depth sample
...
```

Try:

```
$ cp /data/cwpscratch/Data2/vel.fdmig.simple /gpfc/yourusername/Temp2
$ cd /gpfc/yourusername/Temp2
$ sumigffd < simple.su dx=40 dz=12 nz=150
                         vfile=vel.fdmig.simple > ffd.simple.su
```

7.6 Stoffa's split-step migration

Another algorithm, known as the "split-step" algorithm, developed by P. Stoffa, et al in 1990 is an extension of this idea with **sumigsplit**

```
 SUMIGSPLIT - Split-step depth migration for zero-offset data.

 sumigsplit <infile >outfile vfile= [optional parameters]

 Required Parameters:
 nz=                  number of depth sapmles
 dz=                  depth sampling interval
 vfile=               name of file containing velocities

 Optional Parameters:
 dt=from header(dt) or .004    time sampling interval
```

```
 dx=from header(d2) or 1.0     midpoint sampling interval
 ft=0.0                        first time sample
 fz=                           first depth sample
...
```

Try:

```
$ cp /data/cwpscratch/Data2/vel.fdmig.simple /gpfc/yourusername/Temp2
$ cd /gpfc/yourusername/Temp2
$ sumigsplit < simple.su dx=40 dz=12 nz=150
                             vfile=vel.fdmig.simple > split.simple.su
```

7.7 Gazdag's Phase-shift Plus Interpolation migration

A problem with the original Gazdag phaseshift migration is that it did not handle lateral velocity variation well. An approach called "phase shift plus interpolation" (PSPI) was developed Jeno Gazdag in 1984 that partially alleviates this problem. In SU this is implemented as **sumigpspi**

```
 SUMIGPSPI - Gazdag's phase-shift plus interpolation migration
            for zero-offset data, which can handle the lateral
            velocity variation.

 sumigpspi <infile >outfile vfile= [optional parameters]

 Required Parameters:
 nz=             number of depth sapmles
 dz=             depth sampling interval
 vfile= name of file containing velocities
                 (Please see Notes below concerning the format of vfile)

 Optional Parameters:
 dt=from header(dt) or .004    time sampling interval
 dx=from header(d2) or 1.0     midpoint sampling interval
...
```

Try:

```
$ sumigpspi < simple.su dx=40 dz=12 nz=150
                         vfile=vel.fdmig.simple > pspi.simple.su
```

All of these programs are similar in structure, with only the interpolation algorithm being different. Each of these algorithms is easily extended to prestack application (with amplitudes not being preserved).

Note: All of these programs expect an input velocity in terms of **interval velocities as a function of** (x, z). Correspondingly, the output files of all of these programs are *depth sections*, which, if everything has been done correctly, is a representation of a cross section through the earth.

7.8 Lab Activity #7 - Shell scripts

By now everyone is tired of typing the same commandlines, repetetively. Furthermore, it is easy to forget exactly what was typed for further testing. To remedy this problem we will now capture these commandlines into a program called a "shell script." The shell script is one of the most powerful aspects of Unix and Unix-like operating systems.

For example, we can capture all of the migrations above into a single script for comparison. Begin by typing

```
cd Temp2
```

and opening a file called **Migtest** using your favorite editor. The contents of **Migtest** should be:

```
#! /bin/sh

set -x

# Stolt
sustolt < simple.su cdpmin=1 cdpmax=80  \
                    dxcdp=40 vmig=2000 tmig=0.0 > stolt.simple.su
# gazdag
sugazmig < simple.su dx=40 vmig=2000 tmig=0.0 > gaz.simple.su

# phase shift
sumigps < simple.su dx=40 vmig=2000 tmig=0.0 > ps.simple.su

# finite difference
sumigfd < simple.su dx=40 dz=12 nz=150  \
            vfile=vel.fdmig.simple  > fd.simple.su

# split step
sumigsplit < simple.su dx=40 dz=12 nz=150 \
               vfile=vel.fdmig.simple > split.simple.su
```

```
# phase shift plus interpolation
sumigpspi < simple.su dx=40 dz=12 nz=150 \
               vfile=vel.fdmig.simple > pspi.simple.su

exit 0
```

The top line indicates that the Bourne shell (sh) is being used to run the commands that follow. The "set -x" tells the Bourne shell interpretor to echo each command as it is being run. The backslashes (
) indicate are line continuation symbols and should have no spaces following them.

When you have finished typing in the contents of the **Migtest**, then save the file. You will need to change the permissions of this file to give the script execute permission. This is done via

```
$ chmod u+x Migtest
```

You may now run the shell script by simply typing:

```
$ Migtest
```

If you get a "Migtest: command not found" error but the permissions are correct, you likely need to have "." on your path. You can change your path, or type equivalently

```
$ ./Migtest
```

to run **Migtest**. Each command within the shell script is run in succession.

We may consider writing a shell script called **ViewMig** with the contents

```
#! /bin/sh

# Stolt
suxwigb < stolt.simple.su title="Stolt" wbox=800 hbox=200 d2=40 \
xbox=200 ybox=550 &

# gazdag
suxwigb < gaz.simple.su title="Gazdag" wbox=800 hbox=200 d2=40 \
xbox=200 ybox=550 &

# phase shift
suxwigb < ps.simple.su title="Phase Shift" wbox=800 \
           hbox=200 d2=40  xbox=200 ybox=450 &

# finite difference
suxwigb < fd.simple.su title="Finite Difference" \
                  wbox=800 hbox=200 d2=40 xbox=150 ybox=350 &
```

```
# Fourier finite difference
suxwigb < ffd.simple.su title="Fourier Finite Difference" \
                  wbox=800 hbox=200 d2=40 xbox=150 ybox=350 &

# split step
suxwigb < split.simple.su title="Split step" \
              wbox=800 hbox=200 d2=40 xbox=150 ybox=250 &

# phase shift plus interpolation
suxwigb < pspi.simple.su title="PSPI" \
              wbox=800 hbox=200 d2=40  xbox=0 ybox=0 &

exit 0
```

As before, save **ViewMig**, and change the mode via

```
chmod u+x ViewMig
```

to make the file executable. Run the command by typing **ViewMig** on the commandline:

```
$ ViewMig
```

You may vary the parameters to change the appearance of the plot. The idea of ViewMig is to give side by side comparisons of the various migration types.

7.9 Homework #3 - Due 17 Sept 2015 (Thursday session) and 22 Sept 2015 (Tuesday Session).

Rewrite the shell script **Migtest** (combining **ViewMig** so that it saves plots as PostScript output (i.e. use **supswigb** or **supsimage**. Compare the output of the different migration methods. Take care to recognize that you may need vastly different values for **hbox=** and **wbox=** or **height=** and **width=** by checking the self documentation for **supsimage**, **psimage**, **supswigb**, and **pswigb**. Do not set **xbox=** and **ybox=**.

Take particular note of the appearance of the output, any noise or artifacts you see. Include comparison figures, commands, and commentary. Again, submit no more than 3 pages maximum as a PDF file.

7.9.1 Hints

PostScript is a graphics language created by Adobe systems which is the forerunner to PDF. PostScript is still widely used and has not really been replaced by PDF.

On most systems there is a tool for viewing PostScript format files. On Linux systems one such tool is **gs** which is the *GhostScript* interpreter. GhostScript is a powerful graphics data conversion program. Another PostScript viewer **gv** which is *GhostView*. You may be able to view your .eps files with **gs**

```
$ gs filename.eps
```

either the plot will open on the screen, or a new icon will appear on the bar of icons on the left side of your screen. You should also find that if you are using OpenOffice (or LibreOffice) Writer that you should be able to drag and drop PostScript files into your OpenOffice (or LibreOffice) Writer document, directly.

When using any graphics program you need to be aware of its options. In SU, make sure that you type:

```
$ supsimage
$ psimage
$ supswigb
$ pswigb
$ supswigp
$ pswigp
```

so that you can see all of the various options that these programs may expect. Note that the dimensions of the plots are in inches, and that some programs expect plot dimentions as **wbox= hbox=** whereas others expect the dimensions to be given as **width= height=**. For example. Note that the color schemes are completel different in the PostScript generating programs, from thei X-windows graphics program counterparts.

7.10 Lab Activity #8 - Kirchhoff Migration of Zero-offset data

In 1978, Bill Schneider published his Kirchhoff migration method. The program that implements this in SU is called sukdmig2d.

There are two shell scripts in /data/cwpscratch/Data2 that you will need to be able to run this on the simple.su data. Go to your home directory, make a temporary directory called **Temp2**, and copy

```
$ cd /gpfc/yourusername
$ mkdir Temp2
$ cp /data/cwpscratch/Data2/Rayt2d.simple Temp2
$ cp /data/cwpscratch/Data2/Kdmig2d.simple Temp2
$ cp /data/cwpscratch/Data2/simple.su Temp2
$ cp /data/cwpscratch/Data2/vel.kdmig.simple Temp2
$ cd Temp2
```

Furthermore, you will need to make sure that the headers are correct on simple.su. You should see:

```
$ surange < simple.su
80 traces:
tracl    1 80 (1 - 80)
cdp      1 80 (1 - 80)
trid     1
sx       0 3160 (0 - 3160)
gx       0 3160 (0 - 3160)
ns       501
dt       4000
```

If not (for example, if the **sx** and **gx** fields are not set), then do the following

```
$ mv simple.su simple.orig.su
$ sushw < simple.orig.su key=sx,gx a=0,0 b=40,40 > simple.su
```

(If the source and geophone positions are not set, then the **sukdmig2d** program will think that all of the input data are at the position sx=0, gx=0.)

The method of migration is called "Kirchhoff" because the technique is based on an integral equation called the *Kirchhoff modeling formula*, which is a high-frequency representation of the wavefield emanating from a reflector in the form of an integral over the reflector. Some people like to think of the Kirchhoff integral as describing an "exploding reflector" model of reflectivity. In some sense, the *Kirchhoff migration formula* is an approximate inverse of the Kirchhoff modeling formula, meaning that we are in an approximate sense solving for the reflectivity in the subsurface.

The migration integral equation may be implemented as such, or as a sum (an integral) over diffractions (the diffraction stack). In either case, the Kirchhoff migration formula may also be called a "Kirchhoff-WKBJ migration," where the WKBJ (Wentzel, Kramers, Brilliouin, Jeffreys) denotes that ray theoretic quantities are being used. Most importantly traveltimes and some estimate of ray-theoretic amplitudes need to be computed for the background wavespeed profile to do the migration. If you recall, we might view migration as moving data up or down along a light cone. Approximating the wavefield by ray tubes is one way of doing that.

The shell script **Rayt2d.simple** runs the program **rayt2d**, which generates the necessary traveltime tables. The shell script **Kdmig2d.simple** runs **sukdmig2d** on the simple.su data. Read each shell script by typing:

```
$ more Rayt2d.simple

#! /bin/sh

rayt2d vfile=vel.kdmig.simple \
dt=.004 nt=501 fa=-80 na=80 \
fz=0 nz=501 dz=4 \
fx=0 nx=80 dx=40 ek=0 \
```

```
nxs=80 fxs=0 dxs=40 ms=1 \
tfile=tfile.simple

xmovie < tfile.simple clip=3 n1=501 n2=80 loop=1 title="%g traveltime table" &
```

You may type 'q' to get out of the **more** program.

The program **rayt2d** generates a traveltime table containing the traveltime from each source to each receiver for every point in the model. The movie shown by **xmovie** shows these traveltimes as shades of gray. The idea of running the movie over the traveltimes is to see if there are any inconsistencies in the collection of traveltimes.

```
$ more Kdmig2d.simple

#! /bin/sh

sukdmig2d infile=simple.su outfile=kdmig.simple.su ttfile=tfile.simple \
ft=0 fzt=0 nzt=501 dzt=4.00  angmax=80.0 \
fxt=0 nxt=80 dxt=40 fs=0 ns=80 ds=40 \
dxm=40 v0=2000 noff=1 off0=0 doff=0
```

You may type 'q' to get out of the **more** program.

The program **sukdmig2d** performs the Kirchhoff migration drawing on the traveltimes created by **rayt2d** that are in the file **tfile.simple**. Indeed, if a person had a better traveltime table generator, then this program could use those traveltimes instead.

To apply the Kirchhoff migration to the dataset **simple.su** we first type:

```
$ Rayt2d.simple
```

to generate the traveltime tables for **sukdmig2d**. You will notice that a little window showing a movie will appear. You may grab the lower corner of this window by clicking and dragging with your mouse to stretch the image to a larger size. This movie shows the map of traveltimes to each point in the subsurface from the common source-receiver position. As this is only a constant-background model, we would expect that the curves of constant traveltime (wavefronts) are circles. You should be able to detect this in the change of the grayscale. If the curves do not appear to be perfect circles, this is due the aspect ratio of the plot. You may stretch the plot so that it has the correct aspect ratio.

To perform the the Kirchhoff migration we type

```
$ Kdmig2d.simple
```

the resulting migrated data is the file **kdmig.simple.su** which you may view via

```
$ suximage < kdmig.simple.su  &
```

7.11 Spatial aliasing

You may have noticed that the output from migrating the **simple.su** test pattern, no matter what migration method is used, contains artifacts. Because all of the various migration methods tried have more or less the same pattern of artifacts, we are led to suspect that this is caused by the **simple.su** dataset, rather than the migration routines themselves.

We can study the problem by migrating a different version of the test pattern called **interp.su**

```
$ cd /gpfc/yourusername/Temp2
$ cp /data/cwpscratch/Data2/interp.su  .
$ suxwigb < interp.su xcur=3 title="interp data"

$ surange < interp.su
$ surange < simple.su
```

You will notice that **interp.su** appears to show the same data as the original **simple.su** data, but **surange** shows that there are 159 traces, instead of the 80 traces that were in **simple.su**. The **interp.su** data were made by using the program **suinterp** interpolate the traces in **simple.su**. The interpolation was done via the command

```
$ suinterp < simple.su |
      sushw key=tracl,cdp a=1,1 b=1,1 > interp.su
```

The **sushw** program fixes the header values so that the trace numbers are accurately represented.

If we run **sustolt** to migrate the **interp.su** data and compare this with performing Stolt migration on the original **simple.su** data

```
$ sustolt < interp.su cdpmin=1 cdpmax=159 dxcdp=20 vmig=2000 tmig=0 |
                     suxwigb xcur=3  title="interpolated"
$ sustolt < simple.su cdpmin=1 cdpmax=80 dxcdp=40  vmig=2000 tmig=0 |
                     suxwigb xcur=3  title="simple data"
```

then we see that the interpolated data yield a much better image. Please note that **cdpmax=159** and because the data are interpolated, the spacing between traces is **dxcdp=20**, which is half of the value used for the previous tests because the interpolation process put a new trace between every existing trace.

You may rerun the **Migtest** shell script changing **simple.su** to **interp.su**, taking care to change the input parameters to correctly reflect that the number of traces is 159 and that the spacing between them is cut in half to a value of 20.

7.11.1 Interpreting the result

We have several ways of interpreting the presence of artifacts in the migrated **simple.su** data.

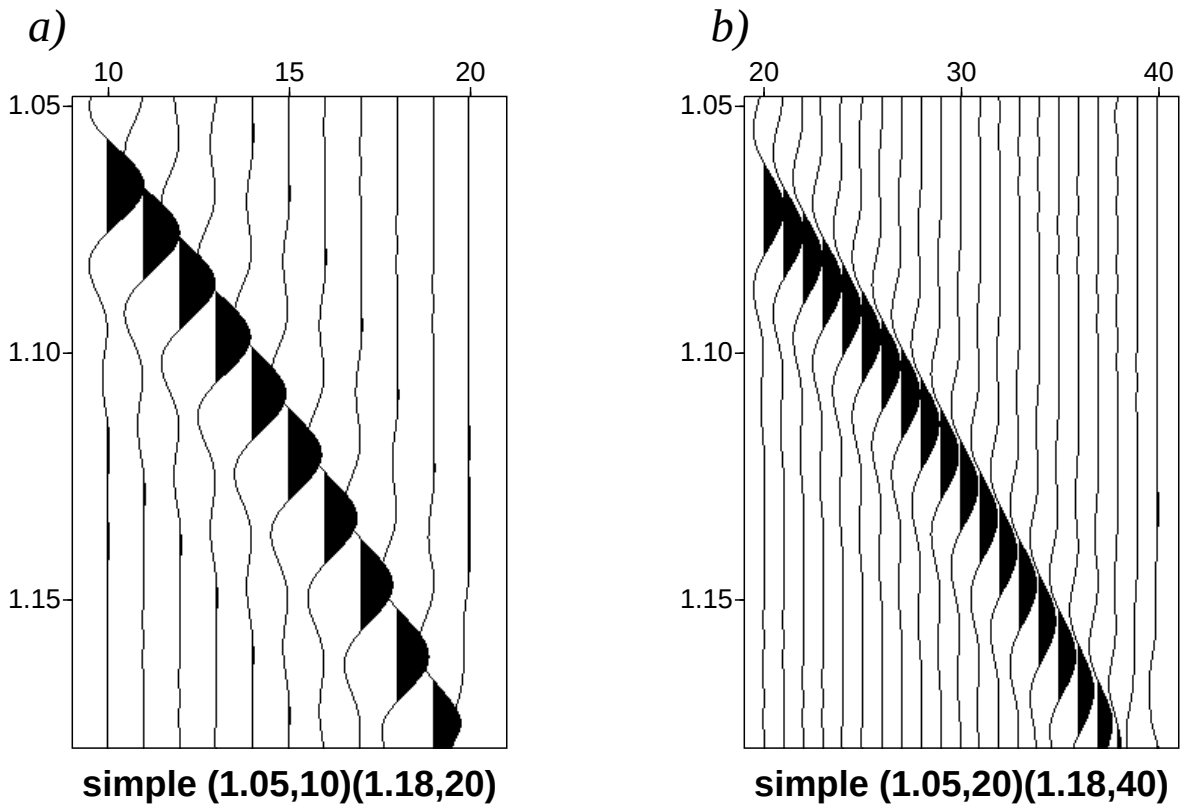


Figure 7.2: a) The **simple.su** data b) The same data trace-interpolated, the **interp.su** data. You can recognize spatial aliasing in a), by noticing that the peak of the waveform on a given trace does not line up with the main lobe of the neighboring traces. The data in b) are the same data as in a), but with twice as many traces covering the same spatial range. Each peak aligns with part of the main lobe of the waveform on the neighboring trace, so there is no spatial aliasing.

First of all, we may consider the spacing between the arrivals on successive traces to be so separated that each arrival acts more like a single point input, so the arcs shaped artifacts represent *impulse responses*.

Second, we may view the artifacts as representing terminations in the integral that is implicit in the inverse Fourier transform, that is the actual mathematical operation of (f, k) migration. So these terminations give rise to *endpoint contributions* where the jumps in the data act like discrete limits of integration.

The third interpretation is that the input arrivals are *spatially aliased* so that the Fourier transform process thinks that certain high spatial frequencies in the data, are really low spatial frequency information, and are putting this information in the wrong place. It is this last interpretation that we would like to investigate further.

7.11.2 Recognizing spatial aliasing of data in the space-time domain

If we view these two input datasets in detail

```
$ suxwigb < simple.su xcur=3 title="simple" interp=1 &
$ suxwigb < interp.su xcur=3 title="interp" interp=1 &
```

by clicking and dragging the rubberbandbox, we can view these datasets in detail. If we zoom in on traces 10 through 20, and time values 1.05s to 1.15s in the **simple.su** data as in Fig 7.2a). In the **interp.su** data, these correspond to the traces 20 through 40 as in Fig 7.2 b). The **interp=1** allows the wiggle traces to be displayed smoothly at any scale of zooming.

The spatial aliasing is evident in the **simple.su** data, because peak of the waveform on a given trace does not align with the main lobe of the waveform on the neighboring traces. Effectively the spatial variability of the data are undersampled in **simple.su** because the trace spacing is to coarse. In real seismic data, great pains are taken to have receiver spacing sufficiently fine to prevent spatial aliasing. However, there are situations, particularly in the case of out-of-plane noise events, that noise can be spatially aliased. Furthermore, we may have missing traces that cause artifacts in migration.

7.11.3 Recognizing spatial aliasing in the (f,k) domain

The term *spatial aliasing* implies that there is a spatial subsampling which occurs, implying that there is a wrapping of data in the wavenumber domain. We can see an example of this in the comparison of (f, k) transforms of *simple.su* data, with the *interp.su* data in Figure 7.3. In Figure 7.3*a)*and *b)* the **simple.su** and **interp.su** data are shown in the (f, k) domain. The **interp.su** data, show a typical (f, k) domain representation of seismic data.

As with aliasing in the frequency domain, we see that as we reach the maximum wavenumber—the equivalent of the Nyquist frequency in the (ω, k) domain, the spectrum is still nonzero. Typically we like to have data vanish smoothly at the Nyquist frequency

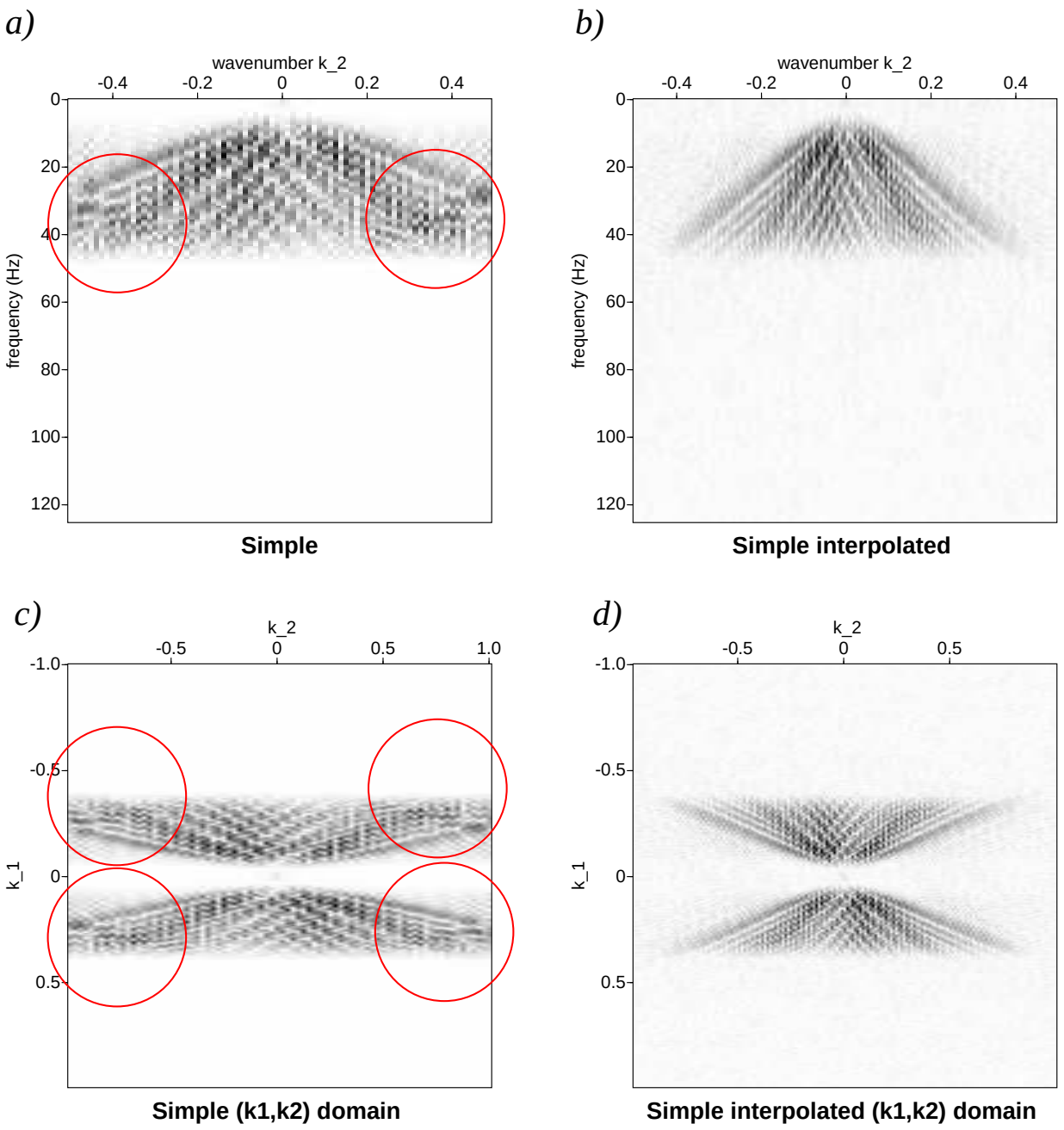


Figure 7.3: a) Simple data in the (f, k) domain, b) Interpolated simple data in the (f, k) domain, c) Simple data represented in the (k_z, k_x) domain, d) Interpolated simple data in the (k_z, k_x) domain. The *simple.su* data are truncated in the frequency domain, with the aliased portions folded over to lower wavenumbers. The interpolated data are not folded.

(or wavenumber) to ensure the absence of aliasing without introducing ringing in the data.

Meaning of the wavenumber domain

But what does the wavenumber domain mean? As we will see in the next section, there is a relationship between the magnitude of the $\boldsymbol{k}$ vector and the quantity ω/c in data owing to the wave equation. For zero offset reflection seismic data, we actually have $|k| = 2|\omega|/c$, were the factor of 2 comes from the fact that we are dealing with 2-way traveltimes. Thus for our seismic datasets, we would expect the range of $|k|$ values to be

$$\frac{2|\omega_{min}|}{c} \leq |k| \leq \frac{2|\omega_{max}|}{c}$$

where ω_{min} and ω_{max} are the minimum and maximum frequencies in the data.

We freely trade time t for depth x_3 so it should not be a shock that we may consider the data, not in the (f, k) domain, but rather in the 2D wavenumber domain (k_1, k_2), where k_1 is the vertical wavenumber and k_2 is the horizontal wavenumber. Figures 7.3*c)*and *d)* show the corresponding images that we obtain by making these assumptions, using the program **suspeck1k2** to calculate the 2D spatial transform amplitude spectrum. The spectrum is symmetric because the data are real valued. The Fourier transform of a real valued function is always symmetric.

The fan-like shape results because we may identify $\boldsymbol{k}$ vectors with "ray vectors." The angualar range in the k-domain represents that angular range of rays which illuminated the reflector in the simple model.

7.11.4 Remedies for spatial aliasing

Fundamentally, spatial aliasing can only be completely avoided if data are sampled sufficiently finely in space to accurately represent all spatial frequencies (alternately wavenumbers) in the data. As we have seen above, simply having receivers more closely spaced significantly reduced the spatial aliasing in our test example. While collecting the data with fine enough spatial sampling, in the first place, is the best remedy, we may not alwatys have adequate spacing for all frequencies in the data.

Mathematically we can see how *frequency bandwidth* corresponds to *spatial coverage*. If we write the wave equation

$$\left[\nabla^2 - \frac{1}{c^2(\boldsymbol{x})}\frac{\partial^2}{\partial t^2}\right] u(\boldsymbol{x}, t) = 0$$

and assume a solution of the form

$$u(x, t) \sim A(\boldsymbol{x}) e^{i(\boldsymbol{k}\cdot\boldsymbol{x} - \omega t)},$$

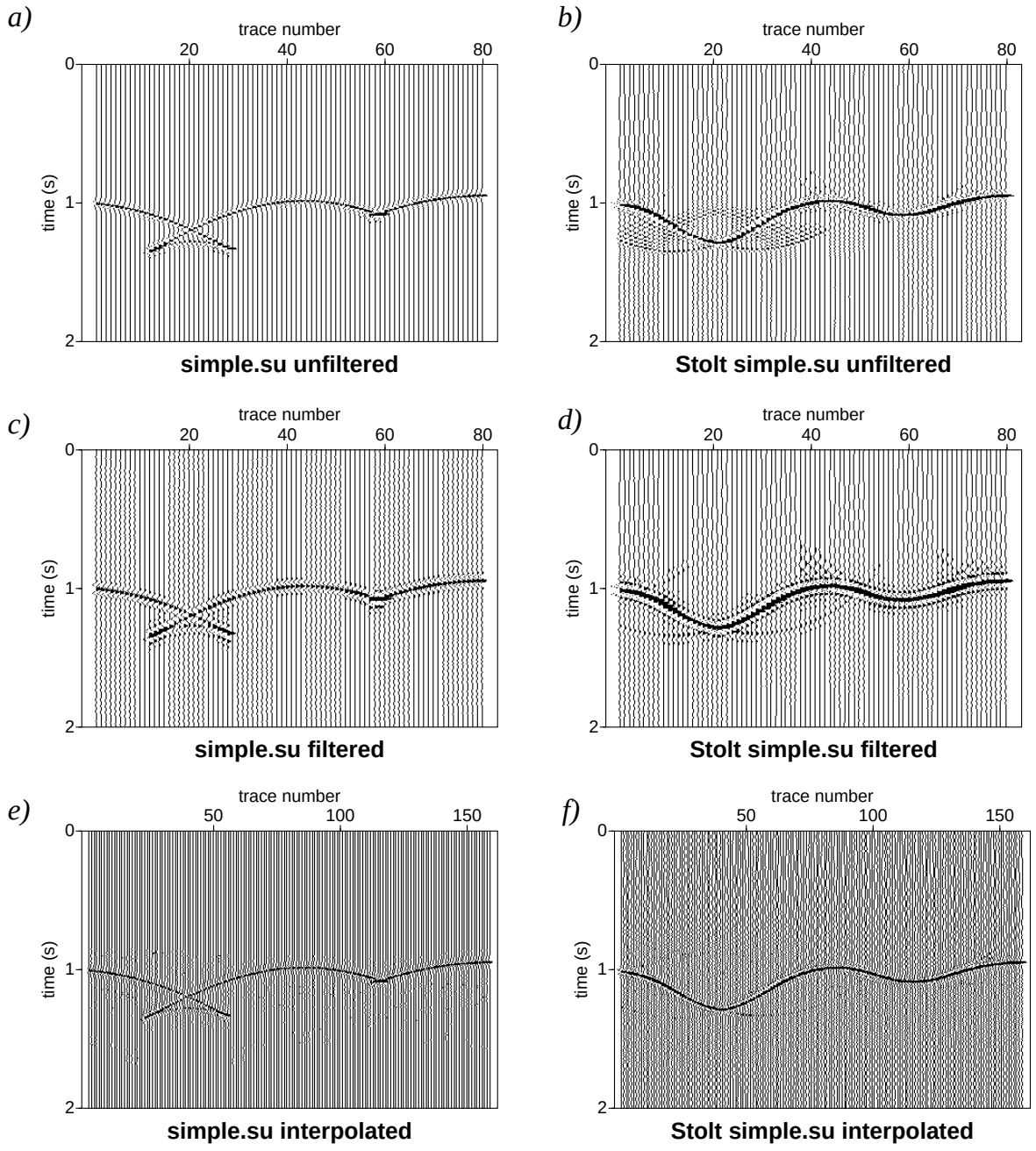


Figure 7.4: a) **simple.su** data unfiltered, b) **simple.su** data filtered with a 5,10,20,25 Hz trapezoidal filter, c) Stolt migration of unfiltered data, d) Stolt migration of filtered data, e) interpolated data, f) Stolt migration of interpolated data. Clearly, the most satisfying result is obtained by migrating the interpolated data.

that is, we assume a space and time oscillating solution, then we obtain

$$\begin{aligned}\left[\nabla^2 - \frac{1}{c^2(\boldsymbol{x})}\frac{\partial^2}{\partial t^2}\right]\left(A(\boldsymbol{x})e^{i(\boldsymbol{k}\cdot\boldsymbol{x}-\omega t)}\right) &= \left[(i\boldsymbol{k})^2 - \frac{(i\omega)^2}{c^2(\boldsymbol{x})}\right]A(\boldsymbol{x})e^{i(\boldsymbol{k}\cdot\boldsymbol{x}-\omega t)} \\ &\quad +2e^{i(\boldsymbol{k}\cdot\boldsymbol{x}-\omega t)}\left[\left(i\boldsymbol{k} - \frac{(-i\omega)}{c^2(\boldsymbol{x})}\right)\cdot\nabla A(x) + \nabla^2 A(\boldsymbol{x})\right]e^{i(\boldsymbol{k}\cdot\boldsymbol{x}-\omega t)} \\ &= 0.\end{aligned}$$

Canceling common terms, and saving only the term with the highest powers in ω and $\boldsymbol{k}$ we have

$$-(\boldsymbol{k})^2 + \frac{\omega^2}{c^2(\boldsymbol{x})} = 0$$

implying that

$$|\boldsymbol{k}| = \frac{|\omega|}{c(\boldsymbol{x})}.$$

Thus when $|\omega|$ and $|\boldsymbol{k}|$ are "large", we have a relationship between $|\boldsymbol{k}|$ and $|\omega|/c(\boldsymbol{x})$ that tells us that reducing the bandwidth in ω will also reduce the bandwidth in $\boldsymbol{k}$.

Thus, *frequency filtering* can be used to combat *spatial* aliasing, provided that all frequencies are not spatially aliased in Fig 7.4 there is a comparison with such a frequency filtered case. The simple data were filtered via the program **sufilter**

```
sufilter < simple.su f=0,5,20,25  > simple_filtered.su
```

to limit the bandwith. Stolt migration was applied to both the original unfiltered version of **simple.su** and the frequency-filtered version. The frequency filtered version shows much less of the effect of spatial aliasing, but also has lower resolution, which is to say that the reflector is fatter. This does not seem to be a problem on an image that has only one reflector, but if there were a bunch closely spaced reflectors, we could easily lose resolution of these reflectors with frequency filtering.

Dip filtering

The third approach to spatial aliasing is filtering in the (f, k) domain. Becuase we want to preserve as much of the frequency content of the data as possible, the filter that we want to use should preserve the magnitude of the wavenumber vectors in the data, but restrict the angles so that we suppress aliasing. Such a filter is called a *dip filter*.

We may experiment with **sudipfilt** by trial and error

```
$ sudipfilt dt=1 dx=1 < simple.su slopes=-2,-1,0,1,2 amps=0,1,1,1,0 |
    sustolt cdpmin=1 cdpmax=80 dxcdp=40 vmig=2000 tmig=0  |
       suxwigb xcur=3 title="migration after dipfilter" d2=40 &

$ sudipfilt dt=1 dx=1 < simple.su slopes=-4,-3,0,3,4 amps=0,1,1,1,0 |
    sustolt cdpmin=1 cdpmax=80 dxcdp=40 vmig=2000 tmig=0  |
```

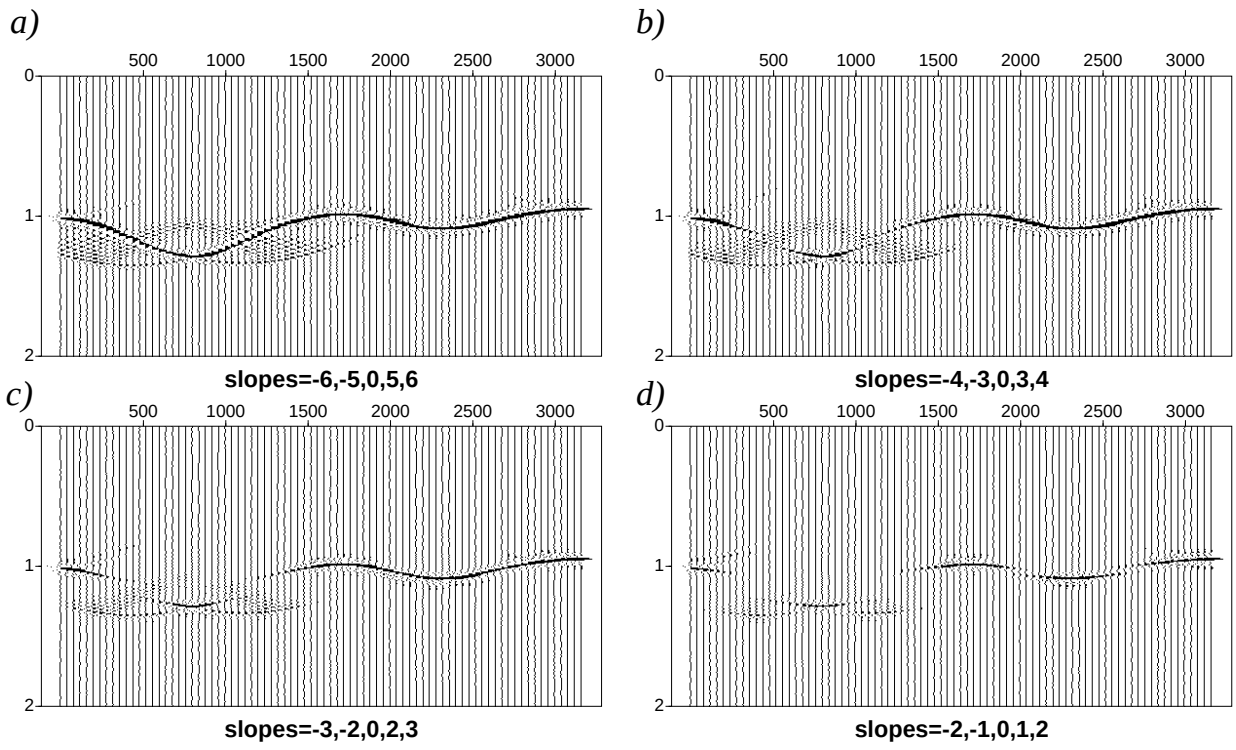


Figure 7.5: The results of a suit of Stolt migrations with different dip filters applied.

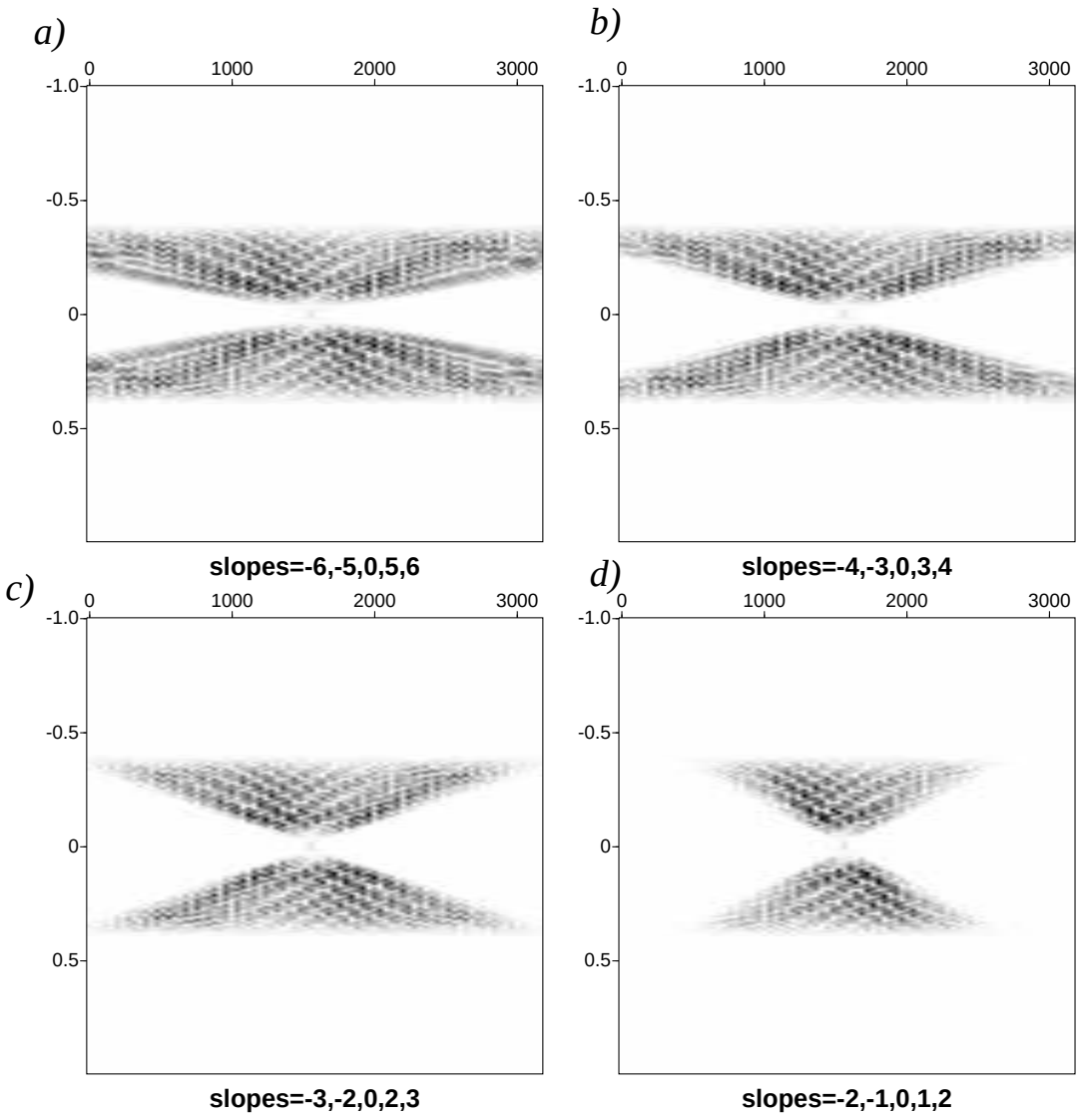


Figure 7.6: The $(k1, k2)$ domain plots of the **simple.su** data with the respective dip filters applied in the Stolt migrations of Figure 7.5

```
       suxwigb xcur=3 title="migration after dipfilter" d2=40 &

$ sudipfilt dt=1 dx=1 < simple.su slopes=-5,-4,0,4,5 amps=0,1,1,1,0 |
    sustolt cdpmin=1 cdpmax=80 dxcdp=40 vmig=2000 tmig=0  |
       suxwigb xcur=3 title="migration after dipfilter" d2=40 &

$ sudipfilt dt=1 dx=1 < simple.su slopes=-6,-5,0,5,6 amps=0,1,1,1,0 |
    sustolt cdpmin=1 cdpmax=80 dxcdp=40 vmig=2000 tmig=0  |
       suxwigb xcur=3 title="migration after dipfilter" d2=40 &
```

Dip filtering is less satisfying in this case, because where it works well to eliminate the spatial aliasing, it also eliminates the dips in the image which constitute the most steeply dipping parts of the structure. The effect in the (k_1, k_2) domain may be seen in Figure 7.6. The resulting migrations of the dip filtered versions of may be seen in Figure 7.5. Where the spatial aliased noise is suppressed the best, the steeply dipping parts of the syncline are not imaged at all. Again, we see that trace interpolation is the best option for suppressing spatial aliasing.

7.12 Concluding Remarks

Investigators in oil companies implemented variations of Hagedoorn's graphical migration, using notions from the theory of wave propagation and methods from signal processing theory. May different types of "migration" were thus created. By accident some of these methods were "amplitude preserving," meaning that reflectivity information is preserved in the image produced by such a migration.

Such "true amplitude" or amplitude preserving migrations became important when issues of reservoir characterization by seismic methods became important. The first of these reservoir characterization methods, first discovered in the 1970s was called the "bright-spot" method, which allowed the identification gas sands in the Gulf of Mexico by their high-amplitude reactivities. In reality, all that was done to see the bright spots was for seismic data processors to stop normalizing away the amplitude differences in their migrated images. This change marked the beginning of seismic migration as a parameter estimation tool.

Chapter 8

Zero-offset $v(t)$ and $v(x,z)$ migration of real data, Lab Activity #9

Now that you have had an introduction to a few zero-offset migration codes, we will apply some of these migrations to a dataset consisting of real data. As before, make a temporary directory in a scratch directory area. Call this one **Temp3**.

```
$ cd /gpfc/yourusername
$ mkdir Temp3
$ cd Temp3
$ cp /data/cwpscratch/Data3/Stoltmig .
$ cp /data/cwpscratch/Data3/PSmig .
$ cp /data/cwpscratch/Data3/seismic3.su .
$ cp /data/cwpscratch/Data3/Suttoz.stolt .
$ cp /data/cwpscratch/Data3/Suttoz.psmig .
```

These data were collected over a structure in the North Sea, known as the Viking Graben. The data were shot by Mobil Corporation back in the 1980s. This is a *stacked section*. The data **seismic3.su** have been gained, have had multiple suppression applied via **supef** (Wiener spiking and Wiener prediction error filter deconvolution), have been NMO corrected (velocity analysis performed with **suvelan**, and normal moveout corrected via **sunmo**), dip-moveout corrected with **sudmofk**, and have finally been stacked using **sustack**. The pre-processing is not very good. No migration has been applied.

First we check the header values on the data with **surange**

```
$ surange < seismic3.su
2142 traces:
tracl    1 2142 (1 - 2142)
tracr    1 120120 (1 - 120120)
fldr     3 1003 (3 - 1003)
tracf    1 120 (1 - 120)
ep       101 1112 (101 - 1112)
```

```
cdp      1 2142 (1 - 2142)
cdpt     1 120 (1 - 120)
trid     1
nhs      1 60 (1 - 1)
gelev    -10
selev    -6
scalel   1
scalco   1
sx       3237 28512 (3237 - 28512)
gx       0 28250 (0 - 28250)
counit   3
mute     48
ns       1500
dt       4000
```

It is convenient to copy and paste this screen into a file called "Notes" for future reference. We see that there are 2142 traces in the section, that the time sampling interval $dt = 4000$, which is to say $4ms$ sampling, and $ns = 1500$ samples per trace. We know from the headers on the original data that the spacing between midpoints, which is to say the spacing between CMPs, is $12.5m$.

You may view the data via:

```
suximage < seismic3.su perc=99
```

or

```
suximage < seismic3.su perc=99 verbose=1
```

The **verbose=1** option will show you the actual clip values that **perc=99** is giving you. You may see what the actual values of **bclip** (the numerical value of the color black) and **wclip** (the numerical value of the color white)

```
bclip=0.569718 wclip=-0.583124
```

You may then input clip values. For example

```
suximage < seismic3.su clip=.2 verbose=1
```

boosts the lower amplitudes. Try different values **clip=** to see what this does.

8.1 Stolt and Phaseshift $v(t)$ migrations

The easiest migrations are the Stolt and Phaseshift migrations. It is common to do a "quick look" at data by applying these types of migrations algorithms. Because a velocity analysis has been applied to these data, we have a collection of stacking velocities as a function of time, which we will make use of as an estimate of migration velocity.

If you type

```
$ sustolt
$ sumigps
```

and look carefully at the parameters for these programs, you will see that **sustolt** requires **RMS velocities** as a function of time, whereas **sumigps** requires **interval** velocities. If you look in the shell script **Stoltmig**, you will see

```
#! /bin/sh

sustolt < seismic3.su cdpmin=1 cdpmax=2142  \
tmig=0.0,1.0,2.5,3.45,4.36,5.1,5.45,5.95 \
vmig=1500,2000,3160,3210,3360,3408,3600,3650 \
dxcdp=12.5 > stolt.seis.su

exit 0
```

Here the pairs **vmig=** are **RMS** velocities (the velocities that come from velocity analysis—stacking velocities) for each time value in **tmig=**. These are only preliminary velocities taken from a single velocity analysis on the data.

If you look in the shell script **PSmig**, you will see

```
$ more PSmig


#! /bin/sh

sumigps < seismic3.su  \
tmig=0.0,1.0,2.5,3.45,4.36,5.1,5.45,5.95 \
vmig=1500,2000,3738.45,3338,3876.32,3678.11,5706.7,4156.17 \
dx=12.5  > ps.seis.su

exit 0
```

Here the pairs **vmig=** are **interval** velocities (the actual seismic wavespeeds) for each time value in **tmig=**. The conversion of RMS to interval velocities is made through the application of the Dix equation. In SU, the program **velconv** is useful for a number types of velocities, including rms to interval velocities. The program **suintvel** is useful for converting a few stacking velocities to interval velocities.

For example, you can run **suintvel** to convert the stacking velocities as a function of time, used in the **Stoltmig** script into the interval velocities used in the **PSmig** script via

```
$ suintvel t0=0.0,1.0,2.5,3.45,4.36,5.1,5.45,5.95
                vs=1500,2000,3160,3210,3360,3408,3600,3650
```

The resulting velocities given by the value of **v=** below

```
h=0,1000,2803.84,1585.55,1763.72,1360.9,998.672,1039.04
v=1500,2000,3738.45,3338,3876.32,3678.11,5706.7,4156.17
```

The values of **h=** are the depths corresponding to the times in the **t0=** field. These could be used for model building, but are not used in migration. Again, these values are only preliminary.

If you look inside the shell script **Suttoz.stolt**

```
$ more Suttoz.stolt
```

you will see that the same velocities are used for depth conversion from the Stolt migrated data (which is in time) to make a Stolt-migrated depth section.

Neither of these sets of velocities should be viewed as "exact"—-these are only preliminary estimates. Notice, for example, that there is no lateral variation in these velocities. These are only $v(t)$, which implies a $v(z)$, rather than a $v(x, z)$, profile. Yet, a cursory examination of the data shows a profile that dips to the right, indicating that there is likely substantial lateral variation of the velocities in the subsurface.

You may run the **Stoltmig** shell script by typing

```
$ Stoltmig
```

you will see that the output file is **stolt.seis.su**. You may create a depth section version of **stolt.seis.su** by typing:

```
$ Suttoz.stolt
```

The resulting output is the file **stolt.depth.seis.su**.

You may now plot all three of these files **seismic3.su**, **stolt.seis.su**, and the depth section version **stolt.depth.seis.su** via

```
$ suximage < seismic3.su clip=.2 title="Stacked data" &
$ suximage < stolt.seis.su clip=.2 title="Stolt time section" &
$ suximage < stolt.depth.seis.su clip=.2 title="Stolt depth section" &
```

8.1.1 Questions for discussion

Compare the original data to the Stolt time migration. Compare how the diffractions that appear in the stacked data are changed in the migrated sections. Are there artifacts? What does the shape of the artifacts tell you about the migration wavespeed?

Look for geologic structures in the time migrated data. Compare these to the time section. Compare the depth section to the time migrated section. Do you see all of the data in the depth section? Look for geologic structures in the depth section.

Where is the water bottom? Do you see an unconformity? Do you see any faults? Artifacts? Horst and graben structures? Any suspicious horizons that might be multiples?

Is the migration velocity correct? Too high? Too low?

8.1.2 Phase Shift migration

We may now run the *phase shift migration* demo script **PSmig** by typing

```
$ PSmig
```

You will first notice first that it takes a bit longer to run the phase shift program than it did with Stolt. The advantage to phase shift migration is that propagates the field locally down in depth, so may handle the local variation of the background wavespeed better. The output of this program is a time section. You may convert this to a depth section by typing

```
$ Suttoz.psmig
```

As with the Stolt migration example, you may view the results via

```
$ suximage < seismic3.su clip=.2 title="Stacked data" &
$ suximage < ps.seis.su clip=.2 title="PS time section" &
$ suximage < ps.depth.seis.su clip=.2 title="PS depth section" &
```

8.1.3 Questions for discussion

You may want to compare these images with the Stolt migrations. Ask the same questions as before. Is the image better in any way? Are there more artifacts, or fewer artifacts? Has your opinion about the background wavespeed changed?

8.2 Lab Activity #10: FD, FFD, PSPI, Split step, Gaussian Beam $v(x, z)$ migrations

To create better images, having a migration that uses a background velocity profile that varies both in horizontal and vertical position $v(x, z)$ is required.

In the directory /data/cwpscratch/Data4 we have several shell scripts that run 5 different types of migration. These are finite difference, Fourier finite difference, phaseshift plus interpolation, split step, and Gaussian Beam (GB).

All of these were discussed in previous sections, with the exception of GB. The "GB" stands for "Gaussian beam" and refers to the beam-like approximate wavefield. This choice of "Green's function" is often found to be an improvement on standard ray methods. Gaussian beam migration was invented by geophysicist Ross Hill in about 1990. Typically, people in the industry usually make a distinction between ordinary Kirchhoff migration and Gaussian Beam migration, however, Gaussian Beam migration is an application of the Kirchhoff migration technique using Green's functions that are beams with an amplitude that varies like a bell shaped, or Gaussian function in a cross section of the beam.

We are going to migrate the same data, **seismic3.su** so copy the shell scripts **Migtest.fd**, **Migtest.ffd**, **Migtest.split**, **Migtest.pspi**, and **Migtest.gb** from /cwpscratch/Data4/ to your Temp3 directory by

```
$ cd /gpfc/yourusername
$ mkdir Temp4
$ cp /data/cwpscratch/Data4/seismic3.su Temp4
$ cp /data/cwpscratch/Data4/Migtest.fd Temp4
$ cp /data/cwpscratch/Data4/Migtest.ffd Temp4
$ cp /data/cwpscratch/Data4/Migtest.split Temp4
$ cp /data/cwpscratch/Data4/Migtest.pspi Temp4
$ cp /data/cwpscratch/Data4/Migtest.gb Temp4
$ cp /data/cwpscratch/Data4/newvelxz.bin Temp4
$ cp /data/cwpscratch/Data4/newvelzx.bin Temp4
```

where these last two files are background wavespeed profiles for You may view the background wavespeed profile by typing

```
$ ximage < newvelzx.bin n1=1500 legend=1
$ ximage < newvelxz.bin n1=2142 legend=1
```

By now, you should recognize that the model is 1500 samples in depth and 2142 samples in the horizontal direction. The file **newvelzx.bin** and **newvelxz.bin** are merely transposed versions of each other. The reason both files are needed is that the first 4 migrations to be tested read in the wavespeed profile in constant depth slices, whereas the Gaussian Beam migration reads the data in vertical slices. This is an oddity of coding, nothing more.

It will take quite a bit of time to run all of these. In a classroom environment, have different students run different scripts, so that all scripts are run in a reasonable amount of time. In each case, simply type the name of the shell script to execute

```
$ Migtest.fd
$ Migtest.ffd
$ Migtest.split
$ Migtest.pspi
$ Migtest.gb
```

Better yet, if you want to get timing information, you may use the Unix **time** function.

```
$ time Migtest.fd
$ time Migtest.ffd
$ time Migtest.split
$ time Migtest.pspi
$ time Migtest.gb
```

Compare your results. The notion of *cost* must be apparent by now. Some algorithms are more expensive in computer time than others. If we merely want to have quick looks, then you cannot beat Stolt migration. However, as more image quality is desire, and more realistic models of the background wavespeed is desired, then the more expensive migrations become more attractive.

8.3 Homework Assignment #4 Due 24 Sept 2015 Thursday session, 28 Sept 2015 Tuesday group - Migration comparisons

Run three of the above migration shell scripts and report on the computational cost and quality of the migrations. Show commands run, make properly labeled plots, and write brief commentary again, a maximum of 3 pages, in PDF format and email to your instructor. Which migration gives the best image? Which migration is the fastest?

8.4 Concluding Remarks

It may have occurred to you to ask why so many different migration algorithms exist. In part, this is cultural. Within company and academic environments, different theories of migration were explored. Part of this is owing to the business culture. If one company is supplying a trademarked service, can other companies compete, unless they invent their own "better" technique? Part of this depends on what we want from the data. Are amplitudes important, or is it all image quality alone? Which is more important, speed or accuracy? Time is money, but then an image that fails to direct the driller to the correct target may be more costly in the long run.

Phase shift and finite difference algorithms are more expensive, but may offer better images, for the reason that the process may be thought of as **wavefront continuation**. Maintaining the continuity of the backward propagating wavefront, should, in theory make a more physically correct imaging process. Kirchhoff migrations are less computationally expensive, but these depend on shooting rays or ray tubes. Rays are sensitive to small variations within a model. If the background wavespeed profile is not sufficiently smooth (i.e. twice differentiable) then inconsistencies in the calculated ray field may result with small variations introducing larger errors. Thus, any cost savings realized may be done at the expense of improper reconstruction or propagation of wavefronts.

Stolt migration relies on neither ray tracing nor wavefront continuation, but on the assumption that a stretching and filtering process in the (f, k) domain can accurately undo the effects of wave propagation on the wavefield, at the expense of never having the correct background wavespeed. Stolt migration is *fast*.

Finally, the beginning student learns that "diffractions are bad." However, if we see a lot of diffractions, then this means the processing from noise suppression to velocity analysis to NMO and stack were done well enough to make the diffractions not be obliterated. In this sense, "seeing a lot of diffractions after stack is good."

Chapter 9

Data before stack

So far, all of our data sets have been zero-offset synthetic data and *poststack* datasets. We now seek to investigate the world of *prestack* data. While we will find that prestack migrations are beyond the capability of the machines in the lab, except for very small examples, we will find that there are a host of prestack processes that we can apply. Indeed, most of the "processing" in seismic data processing is on prestack data. We will find that we will be able to see far more in our dataset than we saw in the examples of the previous chapter. Indeed, students have the experience of making better images than published images on our dataset!

As we proceed, students also may notice that we are having more fun, because this is more like real industry processing, unlike the first few chapters, where our data were largely test patterns.

9.1 Lab Activity #11 - Reading and Viewing Seismic Data

For the lab examples that follow make a temporary directory in your working area called **Temp5** area and type

```
$ cd /gpfc/yourusername
$ mkdir Temp5
```

As was discussed at the beginning of these notes, one of the popular seismic data exchange formats is the SEG Y data format. Data may be in this format on tape media, but today it is equally common for SEG Y data to be files stored on other media, such as CD, DVD, or USB disk drive and memory stick devices. These latter media are preferable for easy transport.

Our test dataset is stored as an SEG Y file called **seismic.segy** in /data/cwpscratch/Data5 as a data file in the SU format. This file is big, about 800 megabytes. Make sure that you are working in an area on your system capable of storing several gigabytes. You may copy this file to your working area via

```
$ cd /gpfc/yourusername
$ cp /data/cwpscratch/Data5/seismic.segy Temp5
```

9.1.1 Reading the data

The data are in SEGY format, and need to be read into the SU data data format. This is done via:

```
$ cd Temp5
$ segyread tape=seismic.segy verbose=1 | segyclean > seismic.su
```

Reading seismic data, particularly tapes, is more of an art than a science. If you ever request data from someone, make sure that you get the data in a format that you can read. Sometimes it is best to have them send you a small test dataset, before committing to a larger set.

Seismic processing software, whether commercial or open source, has the property that there is an internal or working data format that usually differs greatly from the external or "data exchange" formats that data usually are transferred in. In addition, there are field data recording formats that differ still from the data exchange formats. The SU data format is based on the SEG Y format, but is not the same. So, we must convert our data from the data exchange format to the SU format before we can work on the data.

9.2 Getting to know our data - trace header values

Once we have converted the dataset to SU format there are many ways to begin learning about the data. For example we might want to merely view the size of the dataset with **ls -l**

```
$ ls -l  seismic.su
```

will tell us the size of the data. In this case, it also is about 800 megabytes. We may use **surange** to determine the header settings, in particular to see if they are correct

```
$ surange < seismic.su

120120 traces:
tracl    1 120120 (1 - 120120)
tracr    1 120120 (1 - 120120)
fldr     3 1003 (3 - 1003)
tracf    1 120 (1 - 120)
ep       101 1112 (101 - 1112)
cdp      1 2142 (1 - 2142)
cdpt     1 120 (1 - 120)
trid     1
```

```
nhs      1
offset   -3237 -262 (-3237 - -262)
gelev    -10
selev    -6
scalel   1
scalco   1
sx       3237 28512 (3237 - 28512)
gx       0 28250 (0 - 28250)
counit   3
mute     48
ns       1500
dt       4000
```

Because this takes awhile to run, once you have obtained the output from **surange**, open a file name "Notes" with your favorite editor, and copy and paste the screen output from **surange** into Notes. We can see such information as the range of header values, see if the numbers make sense. That sort of thing.

We need to know the total number of traces, the total number of shots, the number of receivers per gather, the time sampling interval, the number of samples. These are all things which we will need for further processing

9.2.1 Setting geometry

One of the most time consuming and difficult, and yet, one of the most important steps in reading seismic data sets occurs in this step of the process. This is called *setting geometry*. The process is one of converting field observation parameters recorded in the field observers' logs into trace header values. The process itself is often time consuming, if everything is correct in the logs, but typically there are errors in observers' logs that complicate this process. It can take as long as a month to set the geometry in a 3D dataset!

In SU, the tools for setting geometry are

```
$ suaddhead
$ sushw
$ suchw
$ sudumptrace
$ suedit
$ suxedit
$ suutm
```

We may make use of **sushw** and **suchw** later in the notes. For the most part, we will assume that our dataset has had geometry set properly.

9.3 Getting to know our data - Viewing the data

If we know that our data have the trace headers set correctly the next part of working with the data is to view subsections of the data to see if there are missing traces, zero traces, and bad traces. We are interested in the quality and reproduceability of the data across the section. We are also interested evaluating whether there is noise that may need to be suppressed.

9.3.1 Windowing Seismic Data

It is always a good idea to look at some small part of the data to see if you *have* data. For example is not uncommon to want to look a the first N traces. For example:

```
$ suwind   key=tracl count=1000 < seismic.su |
                                  suximage perc=99 &
```

gives a quick look at the data. We can see gathers of some variety. To see what kind of gathers we have (shot versus CMP), the header values will help us. Typing the following:

```
$ sugethw sx gx offset ep cdp < seismic.su | more
```

```
    sx=3237         gx=0        offset=-3237        ep=101          cdp=1

    sx=3237         gx=25       offset=-3212        ep=101          cdp=2

    sx=3237         gx=50       offset=-3187        ep=101          cdp=3

    sx=3237         gx=75       offset=-3162        ep=101          cdp=4

    sx=3237         gx=100      offset=-3137        ep=101          cdp=5

    sx=3237         gx=125      offset=-3112        ep=101          cdp=6

    sx=3237         gx=150      offset=-3087        ep=101          cdp=7

    sx=3237         gx=175      offset=-3062        ep=101          cdp=8

    sx=3237         gx=200      offset=-3037        ep=101          cdp=9

    sx=3237         gx=225      offset=-3012        ep=101          cdp=10

    sx=3237         gx=250      offset=-2987        ep=101          cdp=11

    sx=3237         gx=275      offset=-2962        ep=101          cdp=12
```

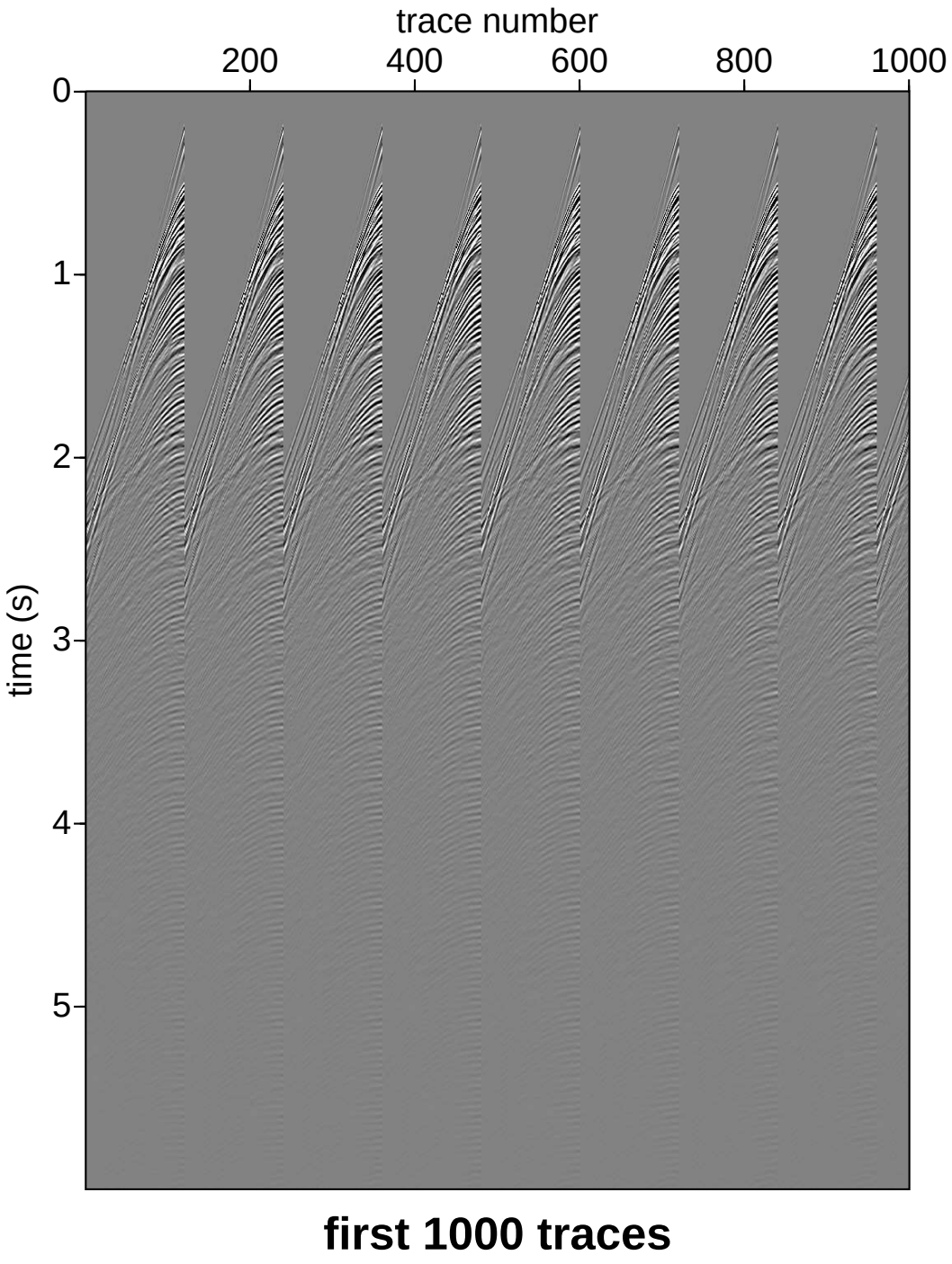


Figure 9.1: The first 1000 traces in the data.

....

shows the values of several important header fields. We can eventually figure out that these are **shot gathers** by noting which fields change the most slowly. In this case, the source position **sx** and the energy point number **ep** are the slowest changing. Shot gathers are also called **common shot gathers**, **shot records**, and **common source gathers**. These terms are used interchangeably.

It is a good idea to get to know your data by flipping through it, much as you would flip through a coffee table picture book. We can view all of the shot records by using **suxmovie**.

```
$ suwind count=12000 skip=0 < seismic.su |
                    suxmovie  n2=1200 loop=1 perc=99 title="Frame %g" &
```

It takes a few minutes, but eventually a window will come up, which you can re-size by dragging on the lower right corner. This window will show a movie of 10 gathers at a time, with the frame number being shown in the title. You can stop the movie at a frame by pressing the far right mouse button. You may see the successive 12000 trace blocks by setting **skip=12000**, **skip=24000**, and so forth.

Events with differing moveouts

Stop the movie at any frame, and zoom into view features of the shot gathers. Some features to look for are *multiples*. These are repetitions in the data in time caused by reverberations in the water column. *Pegleg multiples* may appear to be arrivals with hyperbolic moveout that show a long time moveout in a gather. Whereas *reflections* will have less moveout, indicating higher velocity, but also be hyperbolic in shape. Reflections that have an hyperbola that peaks away from the shot position will indicate a dipping bed.

Direct arrivals will tend to have a linear moveout, as will *ground roll* on land data. that appears to roll over within the section.

9.4 Getting to know your data - Bad or missing shots, traces, or receivers

In real data there may be bad traces or missing traces. Some shots may be bad, or there may be consistent or systematic errors in the data.

9.4.1 Viewing a specific Shot gather

Close the movie window, and capture a particular shot gather for study. For example, we will capture the shot gather at **ep=200**, but any will do. This is done via **suwind**

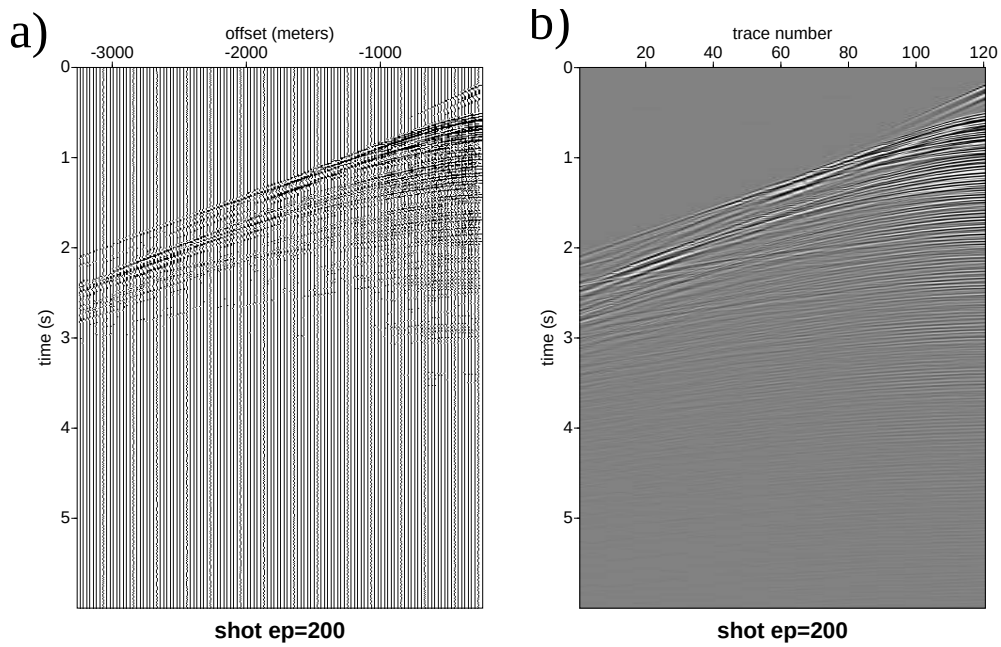


Figure 9.2: a) Shot 200 as wiggle traces b) as an image plot.

```
$ suwind < seismic.su key=ep
                  min=200 max=200 > shot.ep=200.su
```

This will take a few minutes. Once you have this shot gather, you may view the data both as an image plot and as shot gather

```
$ suximage < shot.ep=200.su perc=99 title="shot ep=200"  &
$ suxwigb < shot.ep=200.su perc=99 title="shot ep=200"  &
```

The view of the data is not particularly good, because we have not applied a gain to the data to take into account the amplitude decay with distance traveled.

9.4.2 Charting source and receiver positions

We may view a chart of the source-receiver positions with **suchart**. This is done via

```
suchart < seismic.su |
           xgraph n=120120 linewidth=0
               label1="sx" label2="gx" marksize=2
                        mark=8 title="sx gx chart" &
```

If you zoom in on the plot, missing shots are revealed. Another popular type of chart called the "stacking chart" is discussed below. The **suchart** program is useful as a quality control tool, because any errors in the headers, or inconsistencies in the data are immediately revealed by the plot of header values.

9.5 Geometrical spreading aka divergence correction

The amplitudes of seismic waves experience a reduction in amplitude that is a function of the distance r that the wave travels. There are two sources of this amplitude reduction. The first is due to *geometrical spreading*. The wave energy remains constant, but as the wavefront expands, the energy density reduces as a function of the increasing area of the wavefront.

9.5.1 Some theory of seismic amplitudes

For constant velocity solutions to the constant-velocity scalar wave equation

$$\left[\nabla^2 - \frac{1}{c^2}\frac{\partial^2}{\partial t^2}\right] U(x, y, z, t, x_0, y_0, z_0) = -W(t)\delta(x - x_0)\delta(y - y_0)\delta(z - z_0)$$

look like

$$U(x, y, z, t, x_0, y_0, z_0) = \left(\frac{1}{4\pi r}\right) W(t - r/c),$$

where (x, y, z) is the coordinates of a point on the wavefront, (x_0, y_0, z_0) are the coordinates of the source, t is traveltime, c is the (constant) wavespeed, we assume that the source starts at time $t = 0$, and

$$r = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2}$$

is the radial distance from the source point to a point on the wavefront. $W(t - r/c)$ is a waveform traveling at speed c which arrives at time r/c.[1]

In the real earth, the wavespeed varies with position, so the wavefront surface will not be a simple spherical shell, it will be a complicated function. There may be focusings and defocussings due to the velocity variations that cause the wave amplitude to decay by a function that is more complicated than a simple $1/r$. In reality the *divergence correction* problem is a problem of modeling wave amplitudes as a function of the velocity model.

Anelastic attenuation

The second cause of wave amplitude reduction is due to anelastic attenuation. Rock may be thought of as a kind of spring that undergoes distortion in a cyclical fashion as a wave travels through it. Owing to frictional as well as due to more complicated effects involving the motion of fluids in rock pore spaces, some of the seismic wave energy is converted to heat. This failure of the material to behave exactly elastically is is called

[1] The reader may have expected an "inverse square law" from experiences in undergraduate physics classes, rather than a $1/r$ law. Energy density does diminish according to an inverse square law, but because seismic wave energy is proportional to the square of seismic amplitudes, the $1/r$ amplitude loss is consistent with an inverse square law of energy spreading.

"anelastic attenuation." There is an additional loss of energy due to scattering, which is called "scattering attenuation."

The effect of anelastic and scattering attenuation is to reduce wave amplitudes exponentially as a function of the number of cycles that the wave has traveled—distance in wavelegths, and is usually expressed in terms of a "quality factor" Q. For example in the frequency domain a decaying solution may be written as

$$u(x, y, z, \omega) = \left(\frac{1}{4\pi r}\right) w(\omega) e^{-\frac{\omega r}{cQ}}$$

and $w(\omega)$ is the frequency domain representation of the waveform represented by $W(t - r/c)$ above.

To correct for geometric spreading and attenuative amplitude loss, we may apply an amplitude correction, known as a **gain** to the data. There are many gaining strategies, we will discuss a couple of the more common ones.

9.5.2 Lab Activity #12 Gaining the data

One way to do this is to multiply the data by a power of time. This is done via **sugain**

```
$ sugain < shot.ep=200.su tpow=1 > gain.tpow=1.ep=200.su
$ sugain < shot.ep=200.su tpow=2 > gain.tpow=2.ep=200.su
```

and the effect is to multiply each amplitude on the trace by a factor of t^{tpow}.

A simple way of looking at this is that for an average constant velocity of c, the two-way traveltime is $t = 2r/c$, where r is the distance the wave has traveled to the reflector. Hence $1/t \propto 1/r$, and thus $1/r$ geometrical spreading balanced by multiplying data by t.

This does not seem to be quite enough owing to the fact that the wavespeed generally increases with depth, and the presence of anelastic attenuation. Commonly a factor of t^2 is a better choice, but this may be too much. It may be that the value of the power of t will be a number $1 <$ **tpow** < 2. There are other effects. There may be a general boosting of some amplitudes in such a way that larger amplitudes are boosted more than smaller amplitudes. Thus, a global power may be applied. One example would be to apply a square root function to the data which would be expressed in **sugain** by the choice of **gpow=.5**. Finally, there may be isolated amplitudes that are just too large. These amplitudes may be trucated by clipping the data, thus the **qclip=.95** in the application of **sugain** above.

In Jon Claerbout's *Imaging the Earth*, there is a discussion of a sophisticated application of gaining that would be applied in **sugain** via a choice of parameters that translates into the options: **tpow=2 gpow=.5 qclip=.95**. The latter **qclip=** refers to clipping on *quantiles*. What is a "quantile?"

This is so useful, that we have a special parameter for this called **jon=** wherein **jon=1** applies this combination of parameters in **sugain**. **Caveat: Througout this set of notes jon=1 is used because it is convenient, not because it is optimal! It is up to the you to experiment with sugain to find the optimal gaining.** It may be that you want to run sugain separately with

```
$ sugain < shot.ep=200.su tpow=2 gpow=.5 qclip=.95 | suxwigb
```

and try changing the values of **tpow**, **gpow**, and **qclip** to see the effects of varying these parameters.

Try to find an optimal setting for the panel. One way of testing your choice is to view your data with **suxgraph**

```
$ sugain < shot.ep=200.su tpow=1.0  | suxgraph
$ sugain < shot.ep=200.su tpow=1.5  | suxgraph
$ sugain < shot.ep=200.su tpow=2.0  | suxgraph
$ sugain < shot.ep=200.su tpow=2.5  | suxgraph
$ sugain < shot.ep=200.su tpow=1 gpow=.5  | suxgraph
$ sugain < shot.ep=200.su tpow=1.5 gpow=.5  | suxgraph
$ sugain < shot.ep=200.su tpow=2 gpow=.5  | suxgraph
$ sugain < shot.ep=200.su tpow=1 gpow=.5  qclip=.99 | suxgraph
$ sugain < shot.ep=200.su tpow=1 gpow=.5  qclip=.99
           | suxgraph label1="time (s)" label2="amplitude"

... and so forth
```

When we plot our data in this fashion, with **suxgraph** we are overlaying all of the traces in the gather. The resulting plot shows a crude estimate of the "envelope" of the waves in the gather by plotting all of the traces on top of one another. The envelope is a mathematical surface containing the amplitudes but ignoring the oscillation. If we are successful in removing the decay in amplitude with time, then the amplitude of the envelope will be more or less constant with time.

The idea is to see which combination of the parameters corrects the geometrical spreading and attenuative decay with time so that the amplitudes in the trace are roughly the same for most of the length of the trace. If there are noise spikes or other over corrected spike events, we use the **qclip=** to suppress those.

Note that you probably will not need to set **perc=** on the resulting image if the gaining is correct.

9.5.3 Statistical gaining

Another commonly used method is *automatic gain control* (AGC). The notion of having some automatic or programmed amplification factor dates from the days of analog recording. Instruments have a maximum range of amplitudes that they can record, known as the *dynamic range* of the instrument. It was common that the dynamic range of seismic recording instruments could not accomodate the full range of seismic amplitudes. Adjust the instrument so that it could record all of the early larger amplitudes without clipping, and the smaller and later amplitudes would be lost. Set the instrument for the smaller amplitudes, and the recording system would be overwhelmed by the larger amplitudes at the earlier times in the data. The solution was to use a different gains in differing time windows in the data.

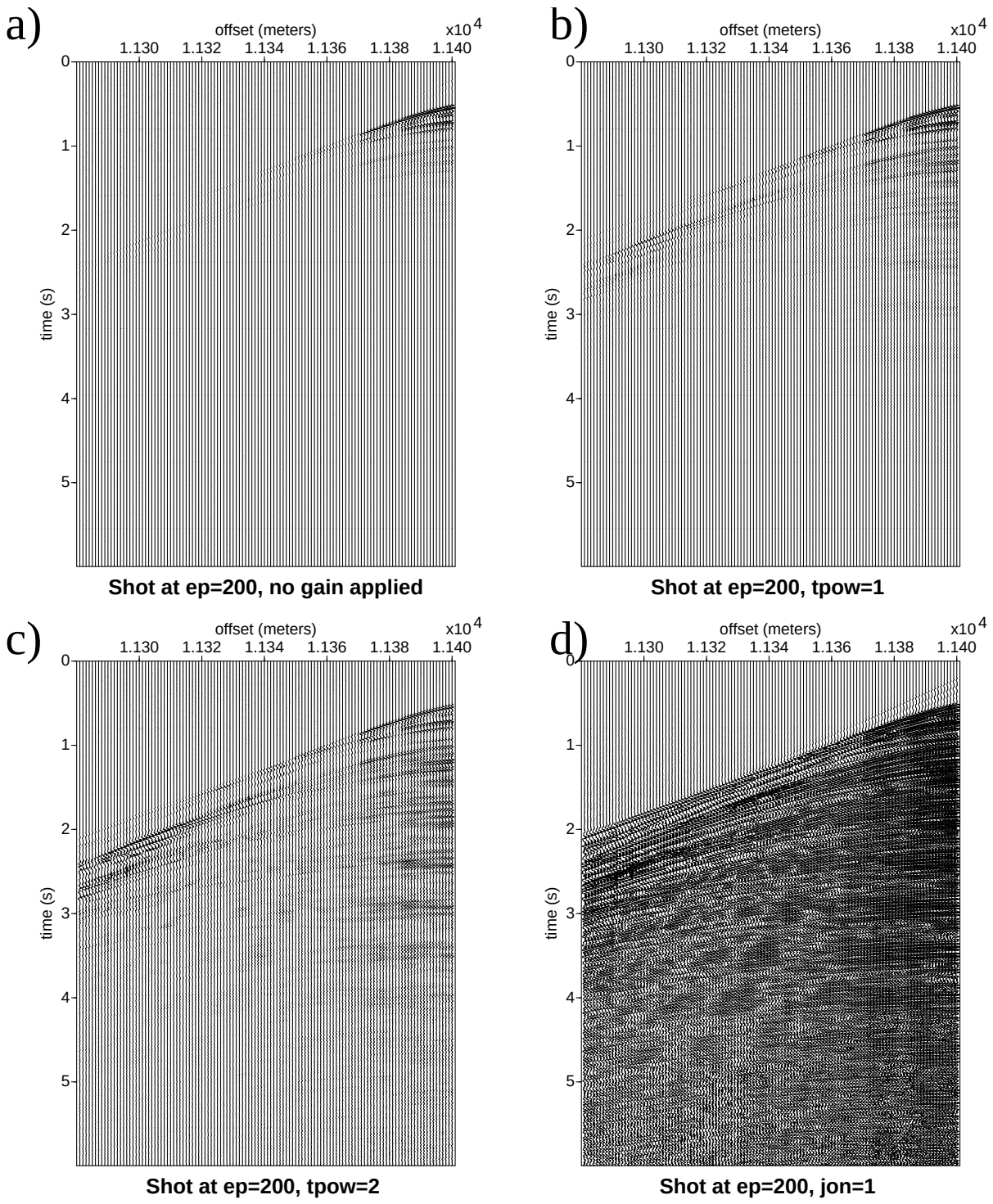


Figure 9.3: Gaining tests a) no gain applied, b) **tpow=1** c) **tpow=2**, d) **jon=1**. Note that in the text we often use **jon=1** because it is convenient, not because it is optimal. It is up to you to find better values of the gaining parameters. Once you have found those, you should continue using those.

AGC takes the rms amplitude of a seismic trace in a succession of windows on each seismic trace, sums over the RMS value of trace amplitude for each window, and normalizes the data within the respective window the by dividing by the sum.

AGC is roughly data driven, but it is somewhat dangerous to use, in that the AGC function can lose small or large amplitudes, and can introduce artifacts that have more to do with the window size you use, and less with the real amplitude decay in the data.

There are varying opinions as when to and when not to use AGC.

9.5.4 Model based divergence correction

There is a more sophisticated approach to gaining data, which is to model the actual geometrical spreading amplitudes by solving the wave equation for the amplitudes using a velocity model, and then normalizing the data based on these calculated amplitude values.

In SU the programs

```
$ sudivcor
$ sudipdivcor
```

In modern migration programs, it may be that we don't want to gain the data, but that the gaining is part of the inverse process that is being applied to the data. The correction for geometrical spreading then, is *built in* to the migration process.

9.6 Getting to know our data - Different Sorting Geometries

We need not live with our data in the form of shot gathers. By now the reader is aware of the CMP–NMO–Stack procedure. The data are recorded as shot gathers, and are resorted to CMP gathers. We may sort data in terms of offset to make *common offset gathers], or by receivers to make* receiver gathers or by any other parameter that we might want.

9.6.1 Lab Activity #13 Common-offset gathers

In the early days of seismic prospecting, it was not unusual for surveys to be conducted of a source receiver geometry consisting of single shot-geophone pairs, collected at a common (constant) offset between source and receiver. This natural because *common-offset gathers* provide data that kind of look like an image of the subsurface—the *data image* discussed in earlier chapters.

We view common-offset gathers today as an important data sorting geometry. From the "Notes" file, we see that the minimum offset in the data is -262 m and the maximum offset is -3237 m. We can save 3 common offset sections via

```
$ suwind < seismic.su key=offset
           min=-262 max=-262 | sugain  jon=1
                                      > gain.jon=1.offset=-262.su

$ suwind < seismic.su key=offset
           min=-1012 max=-1012 | sugain  jon=1
                                       > gain.jon=1.offset=-1012.su

$ suwind < seismic.su key=offset
           min=-3237 max=-3237 | sugain  jon=1
                                       > gain.jon=1.offset=-3237.su
```

Note that we have used **jon=1** for convenience. You should use your own values of **tpow=**, **gpow=**, and **qclip=**. View and compare these respective *near, intermediate, and far offset sections*. Note the presence of multiples, and the time resolution on each section as well as the time of the first arrival representing the water bottom. Indeed, some operations such as prestack Stolt migration (**sustolt** and FK and TX Dip moveout (**sudmofk** and **sudmotx** require that the input data be resorted into common-offset gathers. This is done in SU via:

```
susort offset gx < seismic.su > seismic.co.su
sugain jon=1 < seismic.co.su > gain.jon=1.co.su
```

Again, the choice of **jon=1** for the gaining is used here for convenience. You should use your own values of **tpow=**, **gpow**, and **qclip** instead.

File naming convention

Note that the file names are chosen to reflect the processing steps applied to the data in the file **gain.jon=1.co.su** indicates that the file contains "common offset gathers that have been gained with parameter **jon=1**. The convention is not unique but is convenient as it is easy to forget what processes have been applied to a file.

9.6.2 Lab Activity #14 CMP (CDP) Gathers

In 1950 geophysicist Harry Mayne patented the Common Depth Point Stacking method of seismic data enhancement. The idea is simple, sort the data into gathers whose source and receiver geometry all have the same midpoint in each gather. Correct for the normal moveout (NMO) and sum (stack). The result should be less noisy equivalent zero-offset trace. To sort the data, we use **susort**, a cleverly written program that makes use of the powerful sorting capability built into the Unix operating system.

9.6.3 Sort and gain

Rather than make a bunch of intermediate temporary files, we run the process of gaining and sorting successively in a pipe. Our processing flow for gaining the full dataset is

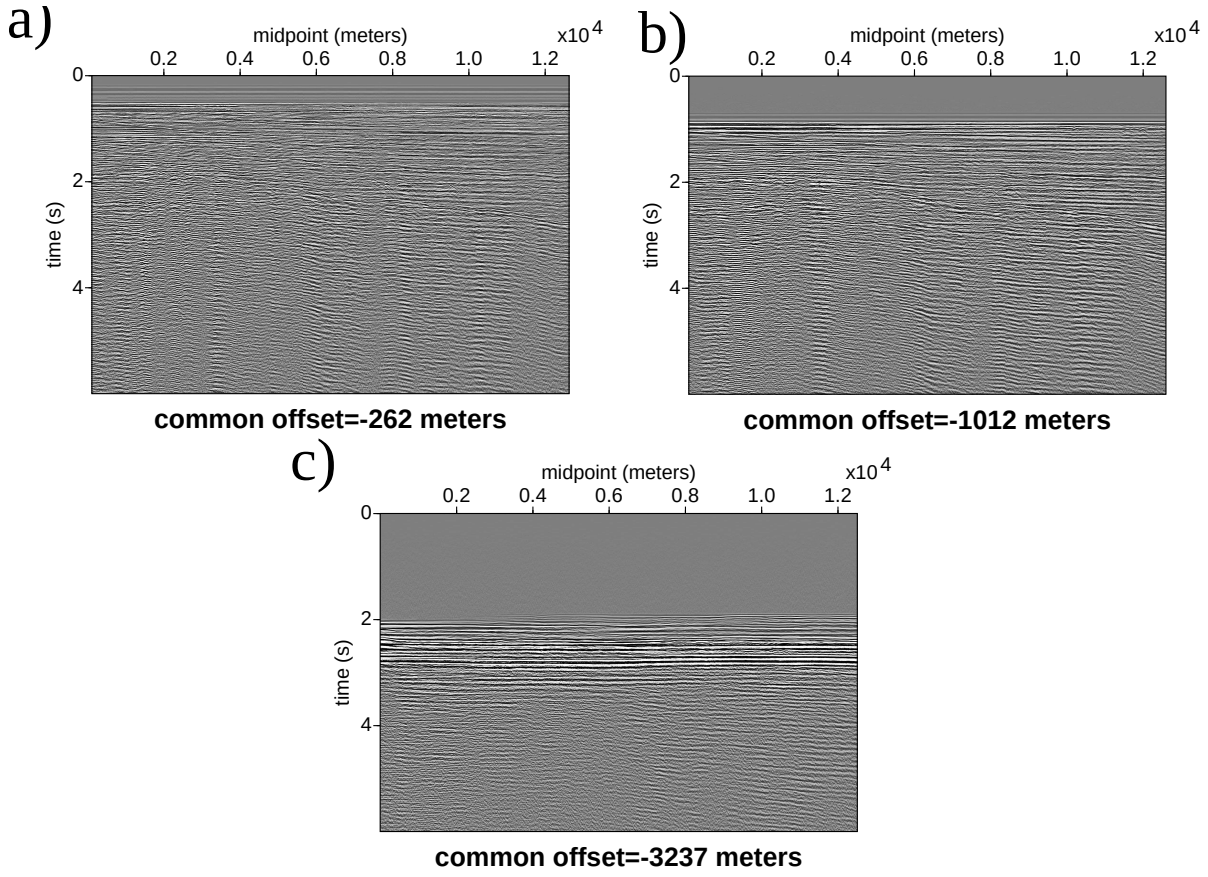


Figure 9.4: Common Offset Sections a) offset=-262 meters. b) offset=-1012 meters. c) offset=-3237 meters. Gaining is done via **...** —**sugain jon=1** — ...
for convenience. A better gaining of the data is possible.

```
$  susort cdp offset < seismic.su  > seis.cdp.su
$  sugain jon=1 < seis.cdp.su > gain.jon=1.cdp.su
```

though here **jon=1** should be replaced with the best values for the gaining parameters that you can find.

File naming convention

Note again that the file names are chosen to reflect the processing steps applied to the data in the file **gain.jon=1.cdp.su** indicates that the file contains "common depthpoint gathers that have been gained with parameter **jon=1**." The convention is not unique but is convenient as it is easy to forget what processes have been applied to a given data file.

If you are experimenting with gains, sort first, because this is a much more expensive operation

```
$ susort cdp offset < seismic.su  > seismic.cdp.su
```

and then do the gain on the sorted data **seismic.cdp.su**.

```
$ sugain  YOUR GAIN PARAMETERS HERE < seismic.cdp.su > gain.PARAMETERS.cdp.su
```

You will note that we always sort from file to file. (We have found that some systems the sorting will fail if **susort** is used with a pipe —.) Again note the naming convention.

9.6.4 Viewing the headers

We now have the data gained, and sorted into CMP gathers. (We use the old term "CDP" here because this term is the one SU uses to designate the CMP header field in the SEG Y header.)

As before, we can view some header fields in the data

```
$ sugethw < gain.jon=1.cdp.su sx gx offset ep cdp | more

    sx=3237         gx=0        offset=-3237        ep=101         cdp=1

    sx=3237         gx=25       offset=-3212        ep=101         cdp=2

    sx=3262         gx=25       offset=-3237        ep=102         cdp=3

    sx=3237         gx=50       offset=-3187        ep=101         cdp=3

    sx=3262         gx=50       offset=-3212        ep=102         cdp=4

    sx=3237         gx=75       offset=-3162        ep=101         cdp=4

    sx=3287         gx=50       offset=-3237        ep=103         cdp=5
```

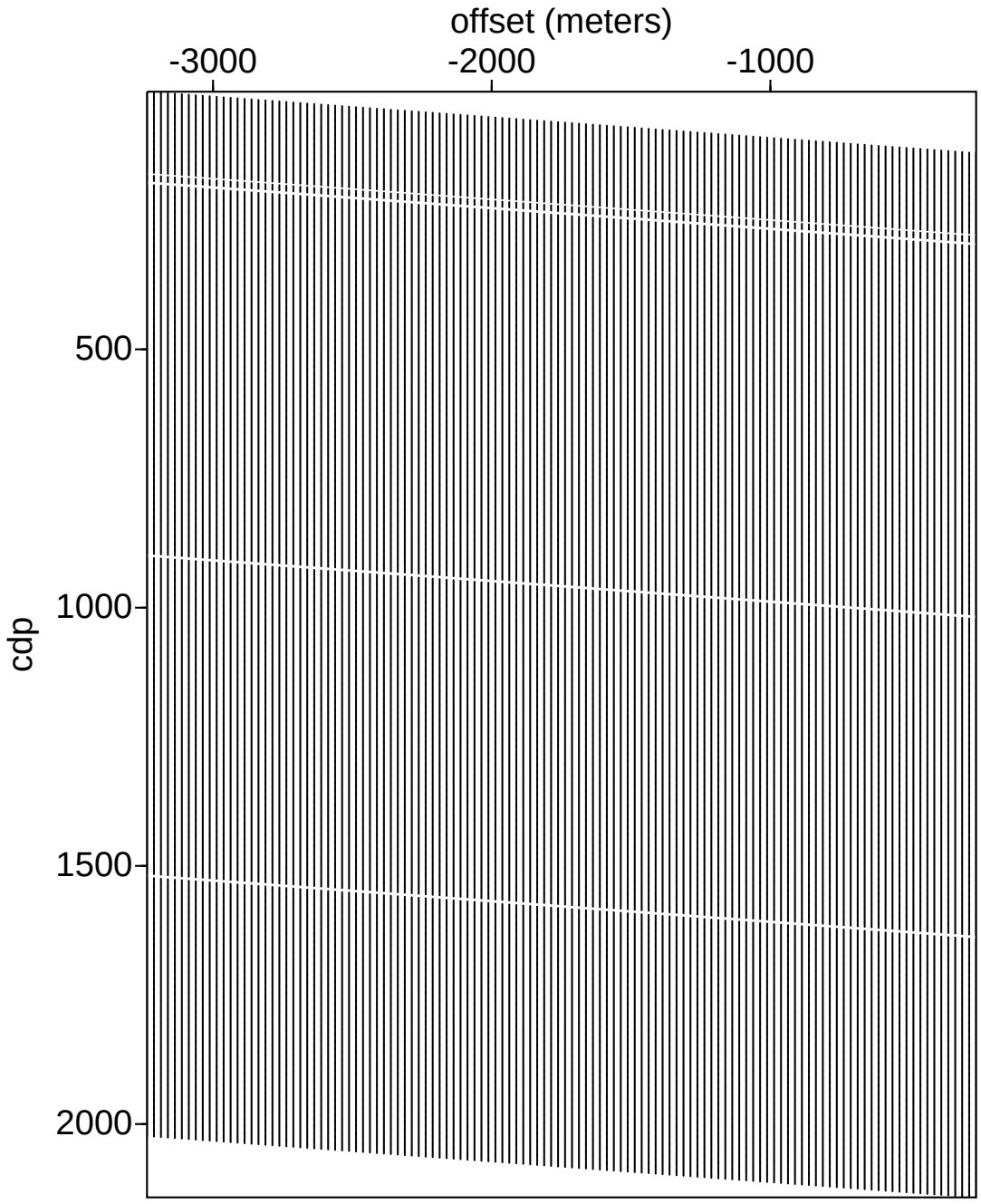


Figure 9.5: A stacking chart is merely a plot of the header CDP field versus the offset field. Note white stripes indicating missing shots.

```
    sx=3262          gx=75        offset=-3187         ep=102          cdp=5

    sx=3237          gx=100       offset=-3137         ep=101          cdp=5

    sx=3287          gx=75        offset=-3212         ep=103          cdp=6

    sx=3262          gx=100       offset=-3162         ep=102          cdp=6

    sx=3237          gx=125       offset=-3112         ep=101          cdp=6

...
skipping
...

   sx=3637          gx=825       offset=-2812         ep=117          cdp=50

    sx=3612          gx=850       offset=-2762         ep=116          cdp=50

    sx=3587          gx=875       offset=-2712         ep=115          cdp=50

    sx=3562          gx=900       offset=-2662         ep=114          cdp=50

    sx=3537          gx=925       offset=-2612         ep=113          cdp=50

    sx=3512          gx=950       offset=-2562         ep=112          cdp=50

    sx=3487          gx=975       offset=-2512         ep=111          cdp=50

    sx=3462          gx=1000      offset=-2462         ep=110          cdp=50

    sx=3437          gx=1025      offset=-2412         ep=109          cdp=50

    sx=3412          gx=1050      offset=-2362         ep=108          cdp=50

    sx=3387          gx=1075      offset=-2312         ep=107          cdp=50
```

We notice a few characteristics of the data from the header fields. First, it seems that the **cdp** field is changing rapidly, but eventually, we see that we have more traces with the same **cdp** value. What is happening is that the data do not have full fold on the beginning of the dataset. We eventually have enough fold to have full coverage in CMP

```
...
    sx=6037          gx=3800      offset=-2237         ep=213          cdp=265

    sx=6012          gx=3825      offset=-2187         ep=212          cdp=265
```

```
    sx=5987         gx=3850     offset=-2137        ep=211          cdp=265

    sx=5962         gx=3875     offset=-2087        ep=210          cdp=265

    sx=5937         gx=3900     offset=-2037        ep=209          cdp=265

    sx=5912         gx=3925     offset=-1987        ep=208          cdp=265

    sx=5887         gx=3950     offset=-1937        ep=207          cdp=265
```

...

9.6.5 Stacking Chart

As we did in Section 9.4.2 we can use **suchart** to view the header fields graphically

```
suchart < seismic.su key1=cdp key2=offset |
            xgraph n=120120 linewidth=0
               label1="cdp" label2="offset" marksize=2 mark=8
```

This is effectively a *stacking chart*, in that it shows the available geophone positions for each CMP. When data are stacked, they are summed along lines of constant CMP on this chart.

If we zoom in on this plot, then missing data becomes apparent as gaps in the plot of header values. The missing shots are distributed over several CMPs and thus the effect of the missing data is minimized, but not eliminated.

We do not have full fold on the 120 CMPs on the ends of the data. but, you can see that on low CDP number end of the plot, it is mainly far offset data that are stacked, whereas on the high CMP number side of the data it is near offset data that are stacked. This accounts for the strange appearance of the edges of the data that we see in stacked sections.

9.6.6 Capturing a Single CMP gather

Around **cdp=265** we are near the **ep=200** portion of the data. We can capture this CMP gather with **suwind**

```
$ suwind < gain.jon=1.cdp.su key=cdp
             count=120  min=265 max=265 > gain.jon=1.cdp=265.su
```

which may be viewed as wiggle traces

```
$ suxwigb < gain.jon=1.cdp=265.su
```

For a better view, we may plot the wiggle traces in true offset, reading the offset from the header field

```
$ suxwigb < gain.jon=1.cdp=265.su key=offset &
```

revealing that there are missing traces due to the missing shots in the data.

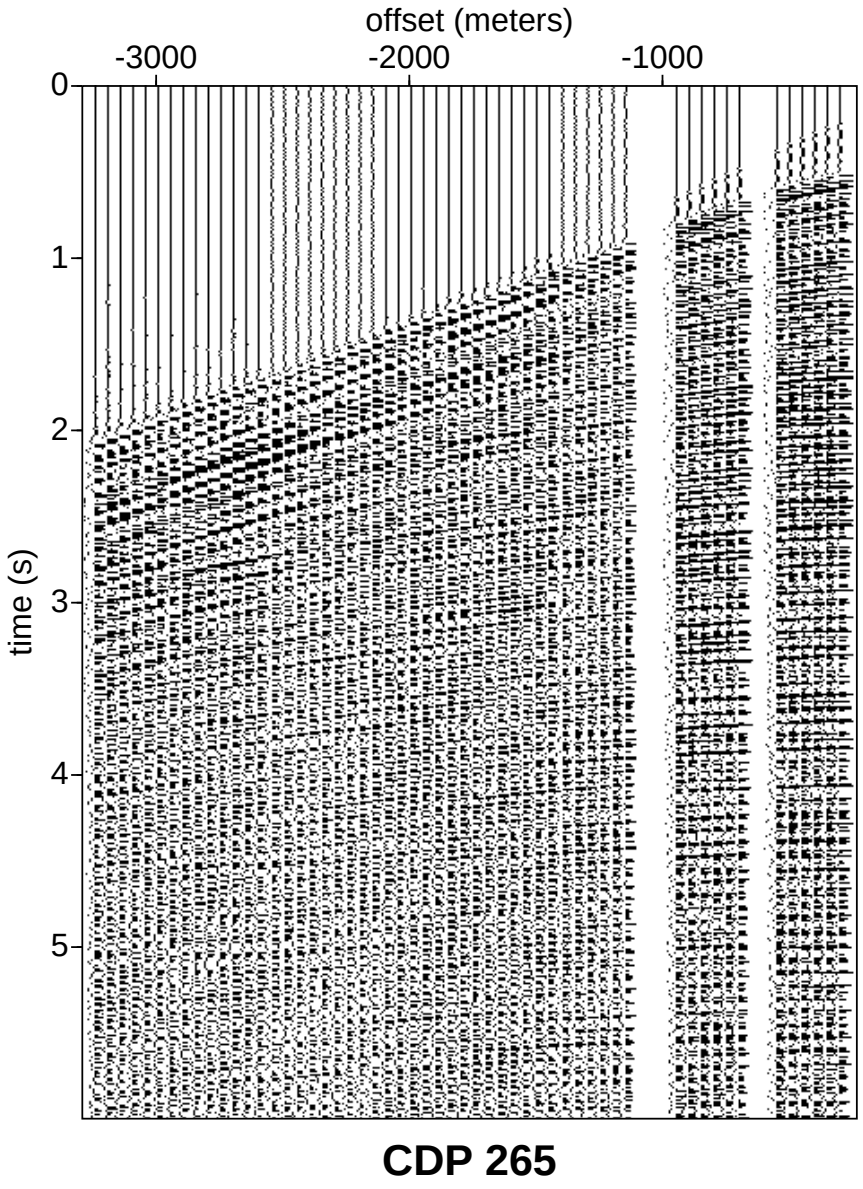


Figure 9.6: CMP 265 of the gained data.

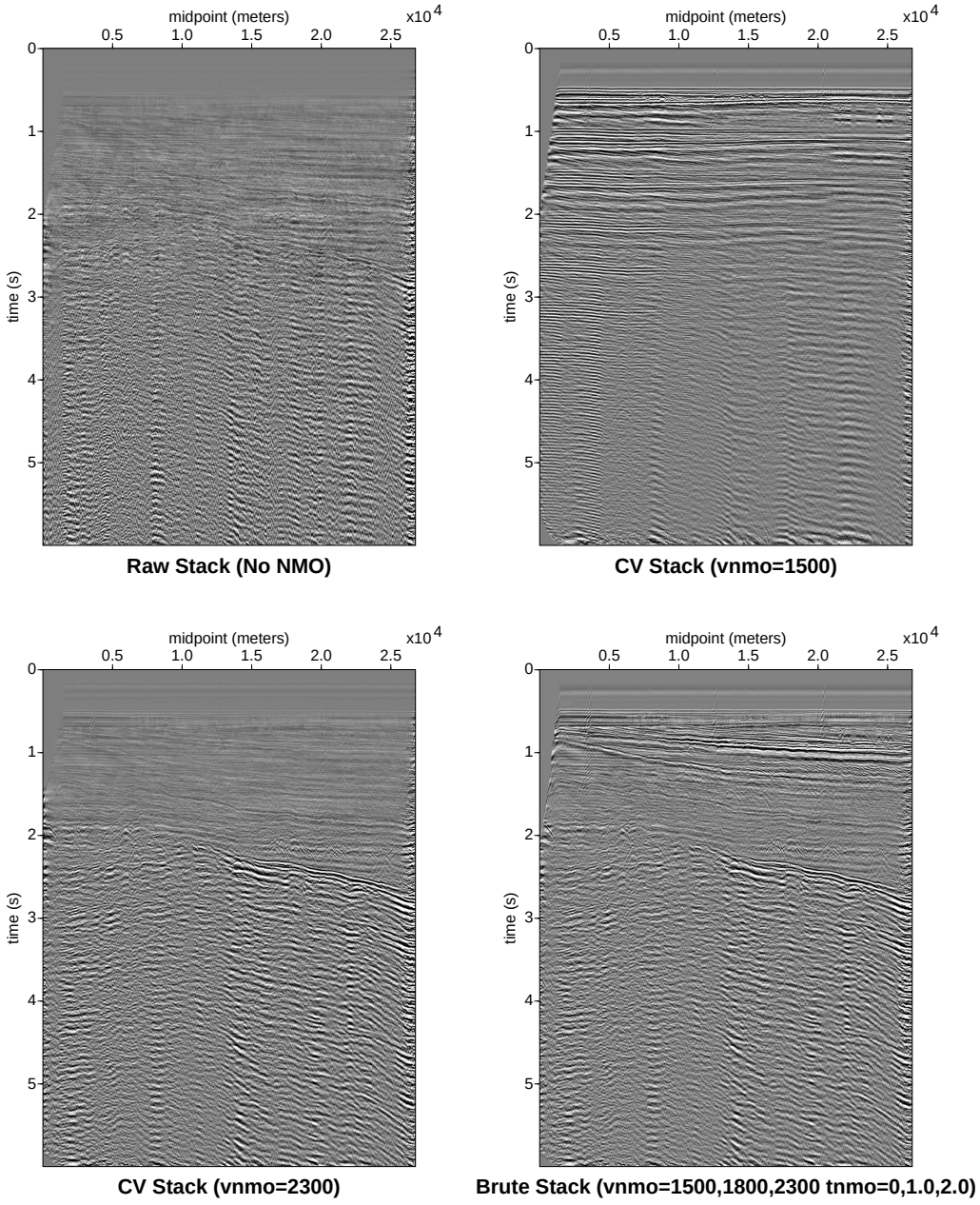


Figure 9.7: a) "Raw" stack: no NMO correction, b) CV Stack vnmo=1500, c) CV Stack vnmo=2300 d) Brute Stack vnmo=1500,1800,2300 tnmo=0.0,1.0,3.0

9.7 Quality control through raw, CV, and brute stacks

The term "quality control" or QC is the industry name for what we have been calling "getting to know your data." The most widely used method of QC is to perform a serice of *CV* stacks and maybe a *brute stack* of the data.

9.7.1 Lab Activity #15 - "Raw" Stacks, CV Stacks, and Brute Stacks

Another common "quick look" technique is the construction of *brute stacks*. As the name suggests, a brute stack is a stack of CMP data with only an approximate NMO correction. Typically some form of brute stack is used as a field quality control technique.

A Raw stack

For example, we may use **sustack** to stack the CMP gathers with *no* NMO correction. Because industry uses the term "brute stack" with the assumption that some rough NMO correction as a function of depth is applied, we use the term "raw stack" for the case of a no-NMO stack of the data. For example try

```
$ sustack < gain.jon=1.cdp.su  | suximage perc=99 title="Raw Stack" &
```

This will take a few seconds to come up on the screen. Remarkably, we can see the hint of our structure even in a stack with no NMO correction. So, yes, we have data.

Constant Velocity (CV) stacks: guessed stacking velocities

We can answer other questions, for example we might want to know more about the multiples? We can NMO correct the data to the water speed of 1500 m/s to view the multiples type

```
$ sunmo vnmo=1500 < gain.jon=1.cdp.su |
      sustack | suximage perc=99 title="CV stack vnmo=1500" &
```

We look for repetition in the data. Multiples consist not only of bounces from the first arrival in the water column, but multiple bounces of many arrivals. The water surface is a big mirror, and not only are there multiple reverberations of the first arrival, but multiple reverberations of all other arrivals, such that potentially the whole seismic section is repeated, with multiples of early arrivals overwriting the later arrivals. This isn't simply a an addition of multiples to the data, but rather multiples become a secondary source, and the multiples are *convolved* with the data.

If we choose a number that corresponds to the moveout time for later arrivals, for example **vnmo=2300**, this will will tend to enhance later reflections, though it is apparent that reverberations dominate the image.

```
$ sunmo vnmo=2300 < gain.jon=1.cdp.su  |
      sustack | suximage perc=99 title="CV stack vnmo=2300 "&
```

Putting it together—a Brute Stack

Suppose that we guess a profile with NMO corrected using vnmo=1500,1800,2300 set at the at the times tnmo=0.0,1.0,2.0.

```
$ sunmo vnmo=1500,1800,2300 tnmo=0.0,1.0,2.0 < gain.jon=1.cdp.su  |
      sustack | suximage perc=99 title="Brute Stack vnmo=1500,1800,2300 &
```

This choice of velocities focuses both the water bottom and shallow section, as well as the strong reflector that starts at 2.0 seconds on the left of the section. But note, these values are really just guesses. Even with a more realistic velocity profile it is clear that we need to both suppress multiples, and perform velocity analysis so that we can get a better stack.

9.8 Homework: #5 Due Thursday 1 Oct 2015 and Tues 6 Oct 2015 prior to 9:00AM

Repeat the gaining and raw and brute stack operations of the previous sections (don't show the "raw stack"), experimenting with **sugain** and the values of **tpow=**, **gpow=**, and **qclip=** to find gaining parameters that you believe work better to balance the amplitudes of the data in time. Replace the **jon=1** in the examples below with your parameters.

To repeat the brute stacking operations put in the additional windowing step

```
... | suwind key=offset min=-3237 max=-1000 | ....
```

to pass only the far-offset data. For example

```
$ sunmo vnmo=1500 < gain.jon=1.cdp.su |
    suwind key=offset min=-3237 max=-1000 |
         sustack > stack.far.gain.jon=1.cdp.su
```

Similarly, stack only the near-offset data, for example

```
$ sunmo vnmo=1500 < gain.jon=1.cdp.su |
    suwind key=offset min=-1000 max=-262 |
         sustack > stack.near.gain.jon=1.cdp.su
```

and compare these results. Perform a similar test for each of the NMO velocity cases in the previous section. Do we really want to include the near offset traces when we stack? What is on these traces and why does **nmo=1500** accentuate this part of the dataset?

9.8.1 Are we done with gaining?

The issue of amplitude corrections are complicated. The example in the Homework assignment above is really a preliminary operation. We can see this by asking what processes should still be done to the data, and in fact, some of these operations should be done *before* gaining.

Muting

Some seismic arrivals should be removed before attempting to gain the data. Muting out direct arrival energy is one such item. Muting means to zero out the data in specific ranges of space and time. The SU program

```
$ sumute
```

allows the muting operation to be performed.

What do we mute? We mute direct arrivals and the place where direct arrivals interact with reflections and refracted arrivals at the earliest times of the data. There is also something called a *stretch mute* which is the removal of a low frequency artifact of the NMO correction that is viewed on NMO gathers.

Wavelet shaping

One of the first things that we do to data is to correct for the shape of the wavelet. This is done by *deconvolution*. There are a number of tools that we may use to improve the waveform that involve methods with a variety of assumptions. We discuss some of these in depth in Chapter 11.

Multiple suppression

As our brute stacks show (particularly stacks with **vnmo=1500** that accentuates energy traveling at or near the water speed), our data are dominated by water bottom and peg leg multiples. We need to do gaining to make the amplitudes of the multiples more uniform in order to remove them, but then we must *re-gain* the data.

Our methods of multiple suppression include *predictive deconvolution* and filtering in the tau-p (also known as the Radon or *slant stack* domain. These are topics discussed in later chapters. In the industry, a method called *surface related multiple elimination* (SRME) is very popular. This method models multiples as the autoconvolution of primary reflections, permitting the multiples to be modeled and subtracted out, leaving the data.

Clearly we are not finished, and in fact, we have not really gotten very far yet.

9.9 Concluding Remarks

After Harry Mayne patented the CDP stacking method in 1950 as part of his work at the Petty Geophysical Engineering Company, oil companies were required to pay a licensing

fee to Petty to use the technique commercially. The technique of sorting, NMO correcting, and stacking the data was done with the data in *analog* form. This required highly specialized and multi-milllion dollar magnetic drums and magnetic tape devices. Such operations as NMO and STACK were performed using this unwieldy and tempermental equipment. The ease and simplicity of applying these operations on digital data, as we do in lab assignments, was years away. These operations were costly and technically difficult in the pre-digital era, yet even with only 6 fold stacking, the improvements in data quality were worth it.

Rather than pay Petty the licensing fee, some companies instead did the common depth point sorting part but did not do the stack. Instead of stacking the data, some companies developed high-density plotters such that the traces were effectively plotted one on top of the other producing a "fat" trace, rather than a stacked trace. (We are talking 6 fold data, primarily, so there were a maximum of six traces plotted, side by side, very closely.) Thus, Harry Mayne's patent encouraged the development of high-resolution plotting equipment.

Prior to the mid-1970s, deconvolution followed by the CDP-NMO-STACK sequence was the majority of seismic data processing. Migration was done only on select sections, or not at all. Seismic interpreters extracted dip and depth information from these stacked data.

Chapter 10

Velocity Analysis - Preview of Semblance and noise suppression

Here, we do a dry run of velocity analysis on a single CMP gather and then do a more "production level" approach to velocity analysis in the next chapter. The method of velocity analysis that we will use is called **NMO Semblance analysis**. The idea is to apply the normal moveout correction over a spectrum of velocities and pick the velocities that NMO correct the data to be most coherent. The coherency measurement is an attribute called *semblance*.

Semblance is defined by the following quotient:

$$s(t) = \frac{\left[\sum_{j=0}^{j=n} q(t,j)\right]^2}{\left[\sum_{j=0}^{j=n} nq^2(t,j)\right]}$$

where $s(t)$ is the *semblance trace* and $q(t,j)$ is the j-th sample on the on the input seismic trace. In other words, semblance is the square of the sum divided by n times the sum of the squares of the values on a seismic trace. It should mentioned that there is more than one measurement of coherency that geophysicists employ, but semblance is the most commonly used measure.

Let's capture and gain CDP=265, if you have not done so already. Here I will use **jon=1** but if you have your own gained version, use that instead

```
$ susort cdp offset < seismic.su > seismic.cdp.su
$ sugain jon=1 < seismic.cdp.su > gain.jon=1.cdp.su
$ suwind key=cdp min=265 max=265 < gain.jon=1.cdp.su > gain.jon=1.cdp=265.su
```

Now we are ready to try doing some velocity analysis using the NMO-semblance method. The program that does this is **suvelan**

```
$ suvelan
```

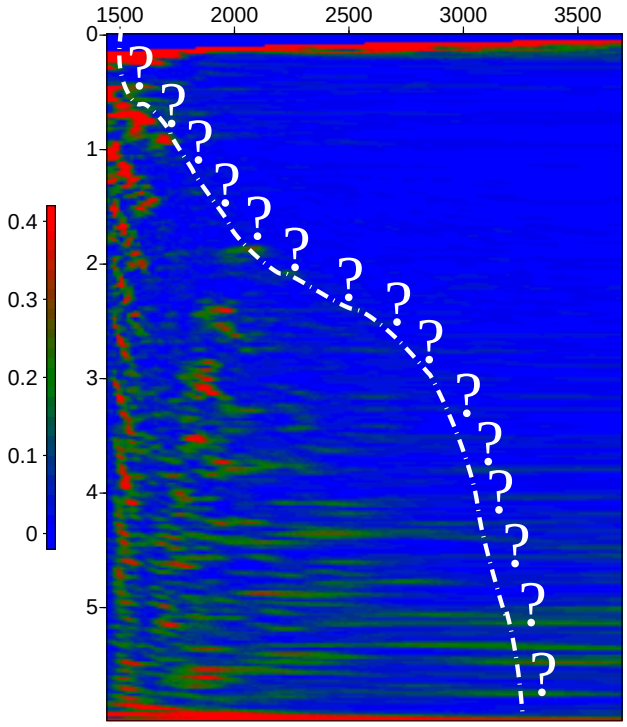


Figure 10.1: Semblance plot of CDP 265. The white dashed line indicates a possible location for the NMO velocity curve. Water-bottom multiples are seen on the left side of the plot. Multiples of strong reflectors shadow the brightest arrivals on the NMO velocity curve.

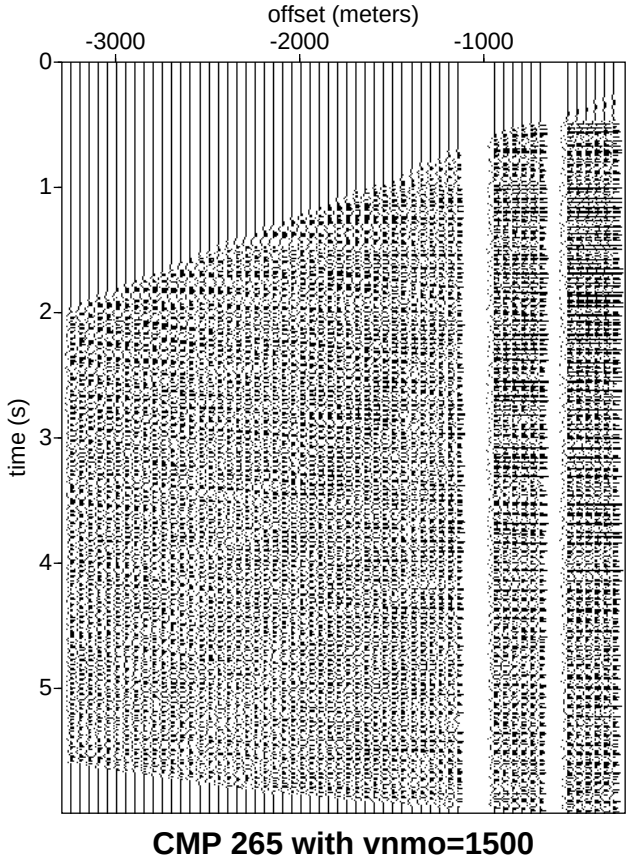


Figure 10.2: CMP 265 NMO corrected with vnmo=1500. Arrivals that we want to keep curve up, wheres multiple energy is horizontal, or curves down.

Here we start with a velocity of **fv=1450** m/s, try different NMO velocities in increments **dv=15** m/s, and choose **nv=150** velocities

```
$ suvelan nv=150 fv=1450 dv=15 < gain.jon=1.cdp=265.su |
   suximage d2=15 f2=1450 verbose=1 cmap=hsv2 legend=1 bclip=.5 &
```

Note that the plotting program is chosen to give a red on blue color map, and the value of **bclip=.5** is chosen to boost the amplitudes on the semblance map.

Running this command on **gain.jon=1.cdp=265.su** we can see evidence of multiples, in that there are repetitions that have something to do with the speed of sound in water. These need to be suppressed before we can proceed further. We see that the multiples have slow moveouts. One set of multiples have a velocity that is approximately the water speed. Another set "shadows" strong reflectors in the subsurface, also tending to the water speed with each repetition.

10.0.1 Creative use of NMO and Inverse NMO

In the sections that follow, we will find that the process of NMO correction may be used as a tool to change the slope of our data. We also will make use of *inverse NMO*. It is possible to make an approximate inverse of the normal moveout correction.

Thus, we will find that we will apply a sequence of *forward NMO* followed by a moveout based filtering technique, followed by an *inverse NMO*. It is important for the reader to realize that these usages are **not** the application of NMO for the final flattening of the data. Even though **sunmo** may appear as part of the processing sequence, the end result is that there is no net NMO correction applied to the data.

10.1 The Radon or (τ - p) Transform

We may exploit the differential moveout in the data between multiples and reflectors by applying an NMO correction to flatten arrivals traveling at the water speed. The Radon or τ - p (τ-p) transform maps the data into the traveltime-slowness domain. The "slowness" may be thought of as the slope of the data in time and offset. In other words it is a quantity with units of *time/distance*. Speed is *distance/time* hence the origin of the term "slowness."

The Radon transform operates by considering each traveltime depth in a seismic dataset and by scanning over the data along curves of different initial slopes or slownesses, referenced between the right and left sides of the data.

Consider for example data made by running **suplane** in the following:

```
$ suplane ntr=120 nt=256 dt=.004 | sushw key=offset a=0
                b=10 | sufilter f=0,5,50,60 > suplanedata.su
$ suximage < suplanedata.su title="suplane data"
                 key=offset label1="time (s)" label2="offset (meters)"
```

where we have 3 intersecting linear arrivals. We would like to separate the data in such a way that will allow us to remove one of the dipping arrivals. We could use *dip or slope* filtering in the (f, k) domain to do this. In this case, filtering in the (f, k) domain might be the best thing, owing to the fact that these are straight lines. However, there is another method called the *Radon or* τ *-p* transform that will do a better job of separating arrivals, particularly if the arrivals are curves with different curvature, such as we see with arrivals having differing moveout.

The Radon transform operates by summing the data over curves that are drawn from time at an intercept offset (**interoff**) which is usually usually either 0 or the smallest offset in the data. The curves are referenced an offset (**offref** that is usually taken as the maximum offset in the data. The data are summed over a fan of these curves taken with differing initial slope and the result plotted as a function of the reference time (the τ) and slope (the p) of each curve.

For example, transforming the **suplanedata.su** made above into the Radon domain is done with **suradon**

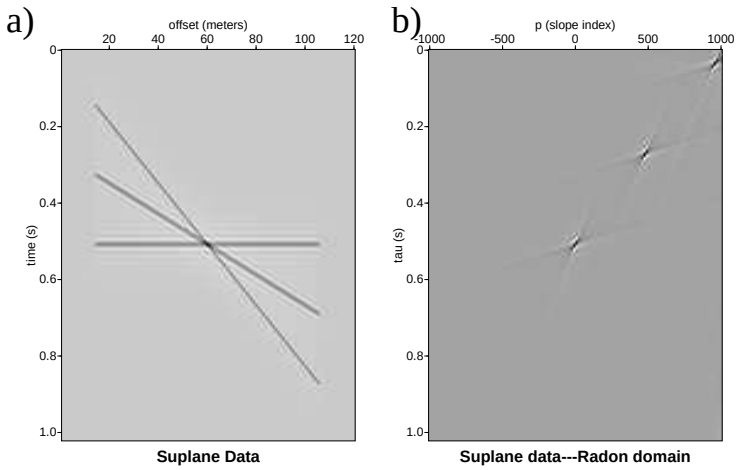


Figure 10.3: a) Suplane data b) its Radon transform. Note that a linear Radon transform has isolated the three dipping lines as three points in the (τ-p) domain. Note that the fact that these lines terminate sharply causes 4 tails on each point in the Radon domain.

```
$ suradon < suplanedata.su igopt=3 ninterp=4 choose=0 depthref=1000
                  interoff=0 offref=1190 pmin=-1000 pmax=1000 > radon.su
$ suximage < radon.su  label1="tau (s)" label2="slope index"
                                title="suplane data Radon transformed"
```

Here it should be noted that **suradon** is a complicated program with a lot of options, so we will approach the problem of using this program cautiously. In this case we use **choose=0** to get a forward Radon transform of the data, **igopt=3** to select a linear Radon transform, meaning that the curves that are summed over are straight lines. The rest of the values make sense if we do

```
$ surange < suplanedata.su
120 traces:
tracl    1 120 (1 - 120)
tracr    1 120 (1 - 120)
offset   0 1190 (0 - 1190)
ns       256
dt       4000
```

which shows that the offset ranges between 0 and 1190 meters. The **pmin** and **pmax** parameters must be chosen to be large enough to encompass all of the slopes (as measured by maximum times) to be seen.

Figure 10.4 shows the result of performing the Radon transform on the data from **suplane**. Because the Radon transform is invertible, one or more of the dipping lines, now represented as points, can be surgically removed, and the inverse transform performed to yield filtered data with one or more of the planes removed. This is different from dip filtering in that there is no frequency domain band-limiting effect.

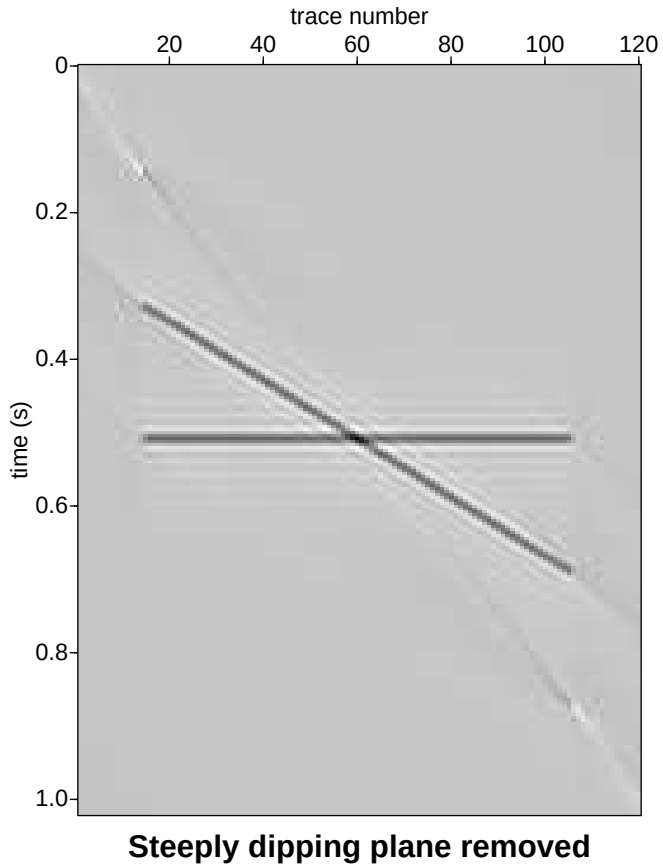


Figure 10.4: The **suplane** test pattern data with the steepest dipping arrival surgically removed in the Radon domain.

Suppose we had applied NMO to the data, so that the most steeply dipping items were the items that we wanted to remove. These steeply dipping arrivals correspond to the arrivals near $p = 1000$ in the Radon transformed plot. We could perform a Radon transform, surgically remove this arrival and then perform an inverse Radon transform to reconstruct the data. Fortunately, the **suradon** program also works as a filter in the Radon domain. If we want to get rid of everything for $p > 600$ then we need merely run the program again, with the choose option. Instead of a Radon transformed dataset, the output this time is the data panel in the time domain with desired items removed. Setting the values of **pmula=600** and **pmulb=600** defines a vertical line in the Radon domain at $p = 600$. The **suradon** program applies a smooth filter to remove contributions to the right of a vertical line to the right of this line

```
$ suradon < suplanedata.su igopt=3 ninterp=4 choose=1 depthref=1000
                  pmula=600 pmulb=600
                  interoff=0 offref=1190 pmin=-1000 pmax=1000 > filtered.su
$ suximage < filtered.su  label1="tau (s)" label2="slope index"
                               title="suplane data filtered in Radon domain"
```

10.1.1 How filtering in the Radon domain differs from $f - k$ filtering

But, so what? We suppressed a dipping arrival. Couldn't we have done that with $f - k$ filtering? For example, the program **sudipfilt** could be used suppress the same arrival. The Radon transform need not be applied only to straight lines, which represent a range of dips in the data. Thus, it is possible to perform the Radon transform by passing more general curves through the data, such as the curves we obtain by performing the NMO correction on CMP gathers. We can attack the differing moveouts between reflections and multiples.

10.1.2 Semblance and Radon for a CDP gather

Our principal sources of multiples come from two sources. The first are simple *water-bottom multiples*, which are reverberations in the water layer. The second are called *peg-leg multiples* (an allusion to the ray path in the water layer resembling a crude artificial leg) which are reflected arrivals that have one or more additional bounces in the water layer.

A crude simulation of our CDP 265 may be made with the **MakeFake** shell script

```
$ cd /gpfc/yourusername
$ mkdir Temp5              (if you have not done so already )
$ cp /data/cwpscratch/Data5/MakeFake  /gpfc/yourusername/Temp5
$ more MakeFake
$ MakeFake
$ ls fake*
```

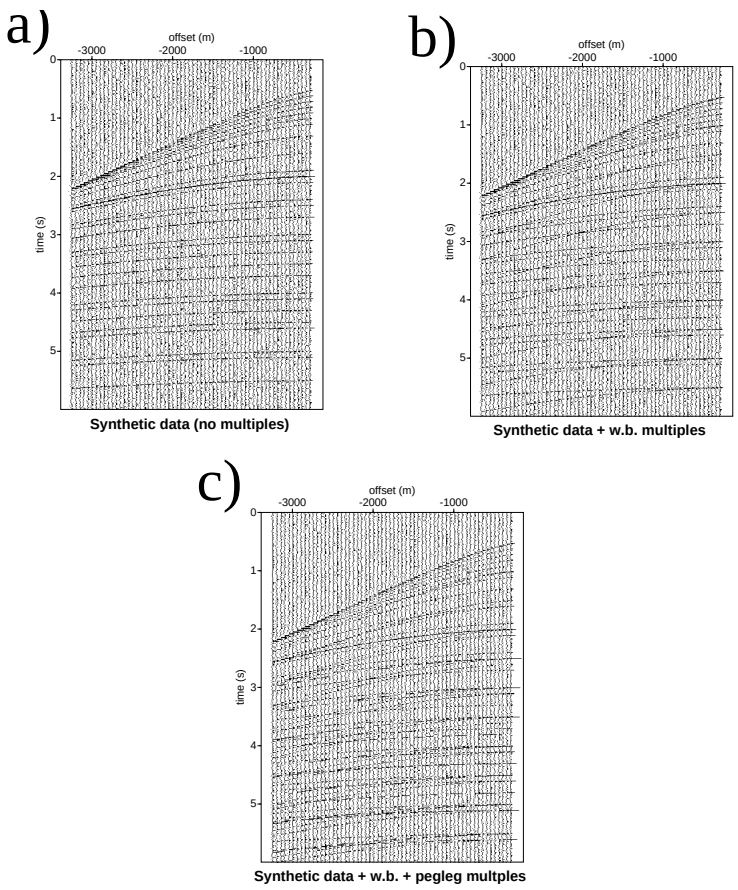


Figure 10.5: a) Synthetic data similar to CDP=265. b) Synthetic data plus simulated water-bottom multiples. c) Synthetic data plus water-bottom multiples, plus select pegleg multiples.

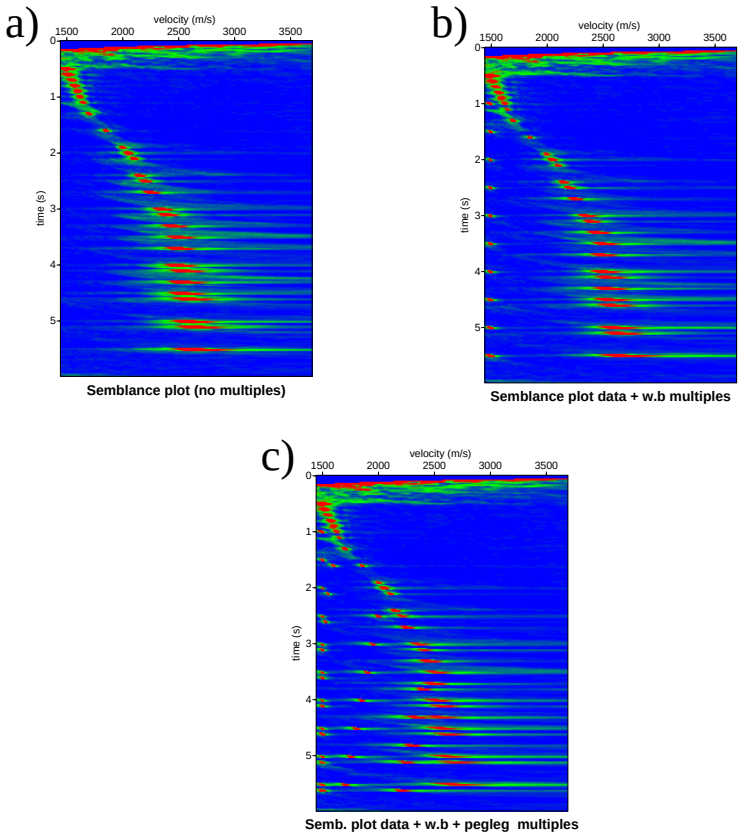


Figure 10.6: a) Synthetic data similar to CDP=265. b) Synthetic data plus simulated water-bottom multiples. c) Synthetic data plus water-bottom multiples, plus select pegleg multiples.

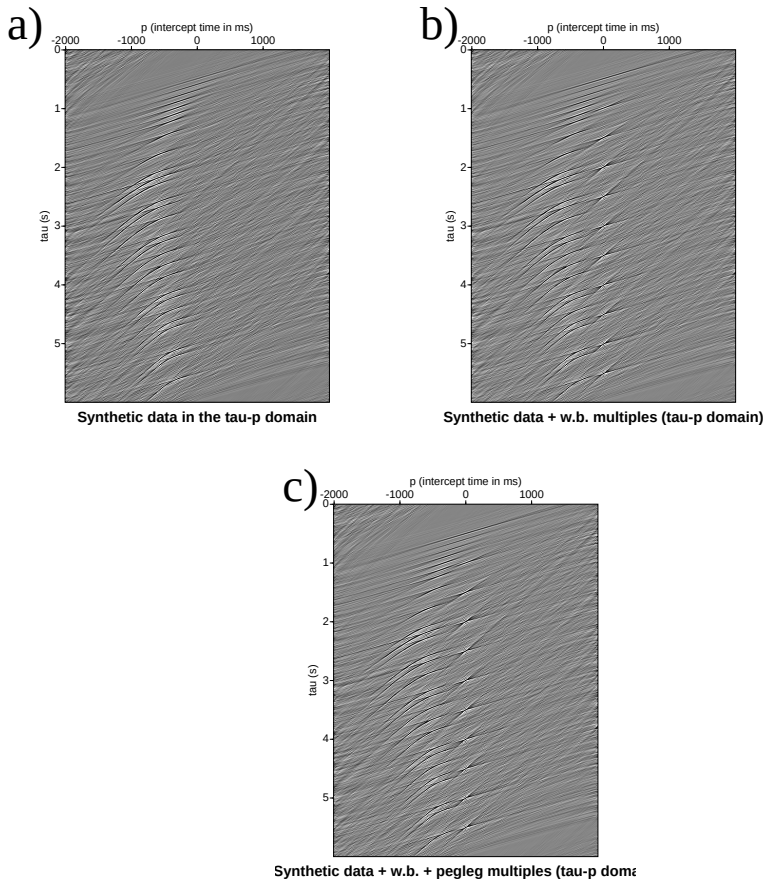


Figure 10.7: a) Synthetic data in the Radon domain b) Synthetic data plus simulated water-bottom multiples in the Radon domain. c) Synthetic data plus water-bottom multiples, plus select pegleg multiples in the Radon domain.

The files that are generated all begin with the word "fake".

```
$ suxwigb  < fake.su perc=99 title="fake data" &
$ suxwigb  < fake+water.su perc=99 title="fake + water bottom multiples" &
$ suxwigb  < fake+water+pegleg.su perc=99
                        title="fake + water + pegleg multiples" &
```

Plots similar to these are shown in Figure 10.5a), b), and c).

Figure 10.5**a)** shows a synthetic data panel similar to CDP 265 in the Viking Graben data without multiples. The second panel Figure 10.5**b)** shows the same data contaminated with simulated water-bottom multiples. Finally, simulated water-bottom multiples plus pegleg multiples from select events are shown in Figure 10.5**c)** .

We can view semblance plots of each simulated datastes via

```
$ suvelan nv=150 fv=1450 dv=15 < fake.su |
   suximage d2=15 f2=1450 verbose=1  title="fake"
                      cmap=hsv2 legend=1 bclip=.5 &

$ suvelan nv=150 fv=1450 dv=15 < fake+water.su |
   suximage d2=15 f2=1450 verbose=1 title="fake+water"
               cmap=hsv2 legend=1 bclip=.5 &

$ suvelan nv=150 fv=1450 dv=15 < fake+water+pegleg.su |
   suximage d2=15 f2=1450 verbose=1 title="fake+water+pegleg"
                    cmap=hsv2 legend=1 bclip=.5 &
```

Plots like these are shown in Figure 10.6**a), b), c)** .

In Figure 10.6 **a)** we see the velocity analysis semblance plot for the synthetic data without multiples. This would be the "perfect" semblance plot. In Figure 10.6 **b)** we see the semblance plot for these same synthetic data contaminated with water-bottom multiples, which are the arrivals at $1500m/s$, the speed of sound in water, arriving at intervals $0.5s$ reflecting the two way traveltime in the water layer. The pegleg multiples are added for select events and the semblance is plotted in Figure 10.6 **c)**. The pegleg multiples are reverberations spaced at $0.5s$, but with decreasing velocity. As we can see, as the pegleg multiples bounce repetitively in the water layer, their velocity approaches the water speed of $1500m/s$.

We may generate corresponding Radon (i.e.τ-p domain) plots of these data via

```
$ sunmo vnmo=1500 < fake.su |
   suradon  offref=-3237 interoff=-262 igopt=2 choose=0
                  pmin=-2000 pmax=2000 dp=8 depthref=1000 |
                      suximage perc=99 title="fake"
                           label1="tau" label2="p"

$ sunmo vnmo=1500 < fake+water.su |
```

```
     suradon  offref=-3237 interoff=-262 igopt=2 choose=0
                 pmin=-2000 pmax=2000 dp=8 depthref=1000 |
                     suximage perc=99 title="fake+water"
                          label1="tau" label2="p"

$ sunmo vnmo=1500 < fake+water+pegleg.su |
    suradon  offref=-3237 interoff=-262 igopt=2 choose=0
                 pmin=-2000 pmax=2000 dp=8 depthref=1000 |
                     suximage perc=99 title="fake+water+pegleg"
                          label1="tau" label2="p"
```

Finally in Figure 10.7 we see the corresponding panels in the Radon (τ-p or *slant stack*) domain. The data have been NMO corrected to flatten arrivals traveling at the water speed. Here everything to the left of **p=0** is data that we want to keep. Water-bottom multiples are flattened to **p=0** and pegleg multiples fall somewhere between the data we want to keep and the water-bottom multiples. A more sophisticated of the NMO correction can be used as a preprocess to make the parts of the data we want to keep fall to the left of **p=0**, while moving items we want to remove to the right of **p=0**.

10.2 Multiple suppression - Lab Activity #17 Radon transform

As we may see in the synthetics in Figure 10.5, multiples tend to have steeper moveouts, which is to say that the multiple energy takes longer time to travel the same distance because a leg of propagation has occurred in the water layer. If we NMO-correct our data to the water speed, or maybe a speed that is slightly higher, this will tend to flatten many of the multiples, but cause events that we want to save to curve up. We perform the NMO correction with **sunmo**

```
$ sunmo vnmo=1500 < gain.jon=1.cdp=265.su | suxwigb
```

You may find that making your wiggle trace plot tall and narrow accentuates the different moveouts. Anything that travels with the speed of water waves is flattened with this choice of NMO velocity. Arrivals that have the moveout of reflections are now curving upward. Anything horizontal, near horizontal, or curving down is something that we want to suppress. We save a copy of this water-speed NMO corrected data as

```
$ sunmo vnmo=1500 < gain.jon=1.cdp=265.su  > junk.su
```

We may now apply **suradon** to transform the data into τ-p domain

```
$ suradon < junk.su  offref=-3237 interoff=-262 igopt=2 choose=0
                 pmin=-2000 pmax=2000 dp=8 depthref=1000 |
                     suximage perc=99 label1="tau" label2="p"
```

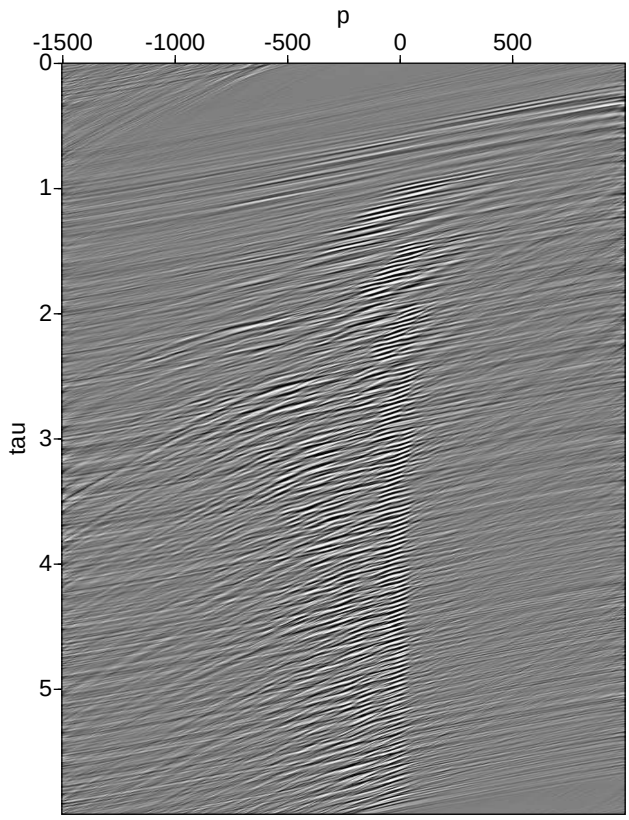


Figure 10.8: CMP 265 NMO corrected with vnmo=1500, displayed in the Radon transform (τ-p) domain. Compare this figure with Figure 10.2. The repetition indicates multiples.

Negative values of p correspond to upward curving events, while **p=0** is anything that is flat. Anything curving down, which is to say, having positive moveout, in other words arrivals that are slower than the water speed, are to the right. The program **suradon** is a sophisticated improvement on the traditional τ-p transform. One improvement is that the p values are given as times in milliseconds on the data, instead of velocities. Also there are several choices of Radon transform that the user may apply. Here in **igopt=2** mode, it sums over hyperbolae rather than mere lines. The p value, then, is the takeoff angle of a hyperbola. Exactly matching a hyperbola would tend to make a dot at a particular τ-p pair on the plot. This is an idealized situation. We don't have the exact hyperbolae. We can control the shape of the hyperbolae with the **depthref** parameter to some degree.

What we get are two regimes of curving arrivals. When we flatten our data with a NMO correction, items with positive p are multiples—these we seek to remove. Items that are flattened or have negative p, are our our data, which we want to keep.

The **suradon** has a second mode useful as filter for multiple suppression. We may select *choose* = 1 to suppress multiples

```
$ suradon < junk.su offref=-3237 interoff=-262 pmin=-2000 pmax=2000
            dp=8 choose=1  igopt=2 pmula=-800 pmulb=47 depthref=1000 |
             sunmo vnmo=1500 invert=1  > junk1.su
```

The values of **pmula=-800** and **pmulb=47** define the beginning and ending p values of the location of the multiples. The program smoothly suppresses items to the right of the line defined by $\tau = 6.0$ and $p = -800$ and $\tau = 0$ and $p = 47$.

The **invert=1** causes **sunmo** to apply "inverse NMO" which is an approximate inversion of the NMO correction to the waterspeed, back to the original data. Some muting occurs (in the right place—a lucky accident) as a result of the inverse NMO.

To see what we obtain from the application of this filter

```
suxwigb < junk1.su title="data after multiple suppression" &
```

To see what we removed from the data, we run suradon in the **choose=2** mode,

```
$ suradon < junk.su offref=-3237 interoff=-262 pmin=-2000 pmax=2000
                  dp=8 choose=2  igopt=2 pmula=-800 pmulb=47 depthref=1000 |
                   sunmo vnmo=1500 invert=1  > junk1.su
```

which gives us an estimate of what was suppressed in the data,

```
suxwigb < junk2.su title="multiples that were suppressed" &
```

A small shell script called "Radon.test" located in /data/cwpscratch/Data5/ puts all of these together

```
#! /bin/sh
```

```
vnmo=
tnmo=
data=
pmula=0
pmulb=0

## view nmo corrected data in the Radon domain
sunmo  vnmo=$vnmo tnmo=$tnmo < $data |
suradon offref=-3237 interoff=-262 pmin=-2000 \
pmax=2000 dp=16 choose=0 igopt=2 \
depthref=1000 | suximage perc=99 &


## nmo->radon-> inverse NMO: for multiple suppression
sunmo  vnmo=$vnmo tnmo=$tnmo < $data |
suradon offref=-3237 interoff=-287 pmin=-2000 \
pmax=2000 dp=8 choose=1 igopt=2 \
pmula=$pmula pmulb=$pmulb \
depthref=1000 |
sunmo vnmo=$vnmo tnmo=$tnmo invert=1 > radon.$data

# view semblance
suvelan < radon.$data dv=15 fv=1450 nv=200 |
suximage d2=15 f2=1450 cmap=hsv2 bclip=.5 &

exit 0
```

10.2.1 Homework assignment #6, Due Thursday 8 Oct 2015 (before 9:00am) and on Tues 13 Oct 2015

- Use "suwind" to capture a single CDP that is different from cdp=265 that we have been studying in class:

  ```
  suwind < gain.YOURPARAMETERS.cdp.su key=cdp
             min=NUMBER max=NUMBER > gain.YOURPARAMETERS.cdp=NUMBER.su
  ```

  (Here for "NUMBER" is any any CDP number between 300 and 2000). View this file with suxwigb, examine the headers with surange (You are working with only one CDP here, remember.)

- Copy the shell script **Test.sh** to your Temp5 directory.

  ```
  cp /data/cwpscratch/Data5/Test.sh /gpfc/yourusername/Temp5
  ```

Modify this file to use your gained version of cdp NUMBER

`gain.YOURPARAMTERS.cdp=NUMBER.su`

file you created in step 1. Run the shell script. Try to find better **tnmo=** and **vnmo=** values than the ones that are in the script to suppress the multiples.

- Show suxwigb (or supswigb) plots of the single CDP before and after multiple suppression, and show the semblance plot of the data after multiple suppression.

Because this is a warmup assignment for more complicated applications later on, here are a few tips and tricks.

1. Make sure you are working with the shell script **Test.sh** located in /data/cwpscratch/Data5 not the shell script **Radon.test** discussed in the previous section. **Test.sh** has some **tnmo= vnmo=** values already set for you, to get you started. Remember that we are using the NMO operator as a tool to separate the moveouts of arrivals that we want to get rid of (the multiples) from the arrivals we want to keep (the reflections). Every application of a forward NMO is followed by an application of filtering in the radon domain, which in turn, is followed by an application of Inverse NMO to undo the original NMO.

2. The goal of velocity analysis is to get the "correct" stacking velocities. In our case, these values will give us two benefits. We can use these **tnmo=** and **vnmo=** for the preprocess for the radon transform domain filtering, and ultimately we will get the NMO velocities to stack the data.

3. The idea is to eliminate the multiples by filtering in the radon domain, and then pick new **vnmo=** and **tnmo=** from your semblance plot.

4. How to pick: You can just zoom in on the semblance plot and read values of the axes. Alternatively, you may pick values by placing the cursor on the place on the plot that you want to pick. Then type the letter "s" and the value of a time velocity pair will be printed in your terminal window. Then edit the shell script to use the new values and then run **Test.sh** again.

5. It is helpful to have a wiggle trace plot of your CDP gather on the screen next to your semblance plot so that you can see what event goes with a given semblance peak. Is it a multiple or is it a real reflector that is making a given semblance peak. Update the **tnmo=** and **vnmo=** pairs and run **Test.sh** again to see if the semblance plot shows fewer multiples. Repeat, until it looks more like a textbook example of a semblance plot.

6. What to pick: If the NMO velocity is perfect at a particular time, were there is a reflector, then the semblance plot will have a peak value at that NMO velocity, and that time.

7. How to know what values to pick: Imagine the geology. The water speed is 1500 m/s at t=0.0, always. The next peak will be at the top of the unconsolidated material in the water bottom, it will be about .5 s and will be only slightly higher than 1500 m/s. We expect that as we go down in depth the velocity will increase until we are in consolidated material, where it will be higher. Speed generally increases with depth, though there may be local velocity reductions (hard to see on semblance). Semblance peaks occur where there are relatively strong reflectors. Eventually, the velocity will tend to increase slowly. Look in Oz Yilmaz book or in Hill and Rueger's notes for an example of "perfect" semblance plots.

8. What if it doesn't work? The only thing left to vary is the location of the filter in the Radon domain. The filter is defined by the values of **pmula=** and **pmulb=** in as used in **suradon**. The filter is defined by a straight line, with endponts time=0 and the value of pmulb, and time= maximum time on the section and the value of pmula. In practice we have found that good values of pmulb are between 180 and 240 and good values of pmula are between 5 and 30. Feel free to experiment.

9. Note that the semblance peaks move to the right or the left a bit depending on where in the radon domain the filter is applied. So, you really need to repick your velocities after you have changed the filtering.

10.2.2 We are not finished with multiple suppression and velocity analysis.

You will notice a that on the near offsets, there are just as many multiples as when we started. This occurs because at near offsets, all arrivals are almost flat, whether they are once-reflected arrives from beds, or if they are multiples. Differential moveout methods are less effective at near offsets.

10.3 Muting revisited

If you plot the multiple suppressed gather from the Homework assignment, you will notice that values have been zeroed for short times and far offsets. Thus, there are some "time,offset" pairs that are zeroed out by **sunmo** and are thus missing after the cascade of processes that finishes with an inverse NMO.

10.3.1 The stretch mute

When we apply the normal moveout (NMO) correction, there is a radical distortion of the data called the *NMO stretch*. To remedy the NMO stretch, an editing of the data called the *stretch mute* is applied. You have already applied the stretch mute, but you were unaware of this.

If you look at the self-doc for **sunmo**

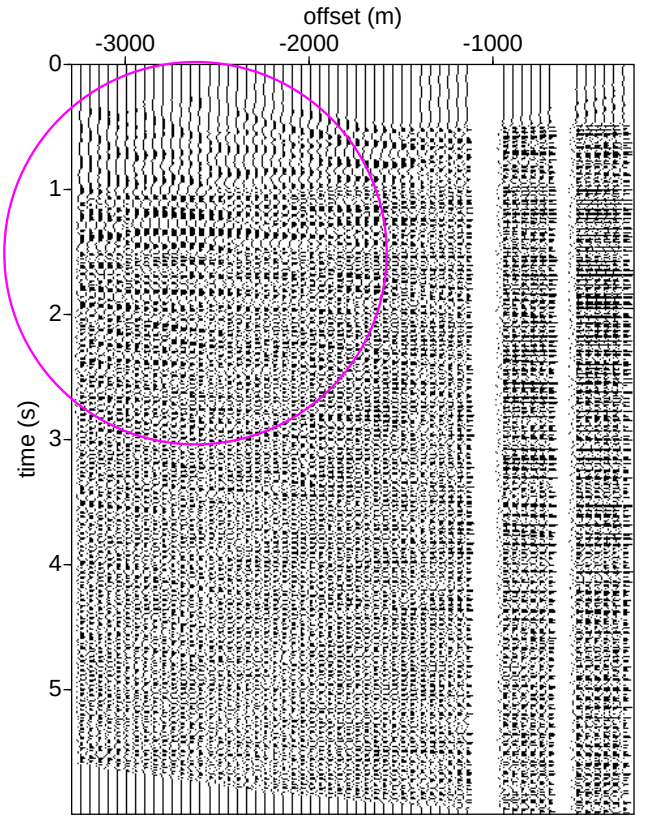


Figure 10.9: CDP 265 NMO corrected with the velocity function **vnmo=1500,1800,2300 tnmo=0.0,1.0,2.0** but with no stretch mute parameter applied. NMO stretch artefacts appear in the long offset, shallow portion of the section.

```
$ sunmo

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

  sunmo <stdin >stdout [optional parameters]

 Optional Parameters:
 tnmo=0,... NMO times corresponding to velocities in vnmo
 vnmo=1500,... NMO velocities corresponding to times in tnmo
 cdp= CDPs for which vnmo & tnmo are specified (see Notes)
 smute=1.5 samples with NMO stretch exceeding smute are zeroed
 lmute=25 length (in samples) of linear ramp for stretch mute
 sscale=1 =1 to divide output samples by NMO stretch factor
 invert=0 =1 to perform (approximate) inverse NMO
 upward=0 =1 to scan upward to find first sample to kill

 Notes:
 For constant-velocity NMO, specify only one vnmo=constant and omit tnmo.

...
```

you will note that there are two parameters **smute=1.5** and **lmute=25** which control the amount of stretch muting and the tapering of the mute. We may run **sumute** with the stretch mute parameter turned off by choosing a large number for the value of **smute=**

```
$ sunmo vnmo=1500,1800,2300
              tnmo=0.0,1.0,2.0 smute=8 < gain.jon=1.cdp=265.su |...
```

The result is in Figure 10.9. We can clearly see the NMO stretch phenomenon. The default values of **smute=** and **lmute=** work pretty well for most applications, however if you see long period artifacts on your stacked section, it is possible that you may need to adjust the values of the stretch mute. Conversely, we do have the possibility of losing data if the stretch mute is too agressive. In either case the stretch mute may need to be adjusted.

10.3.2 Muting specific arrivals.

The term *muting* simply means zeroing out parts of the data that we don't want. The items to mute consist of *random noise* that may appear on the traces before the onset of the actual arrivals, *direct arrivals* from the source that have traveled in the water layer, *refracted arrivals* also called *head waves* from the shallower layers of the water bottom, and *wide angle reflections* from the shallow layers of the water bottom that may be seen at longer offsets. These items are undesirable because they do not fit the traveltime moveout of reflectors, or do not fit with the theory of seismic reflection that we assume when migrating the data.

10.3.3 Lab Activity #16 – muting the data

Muting is a simple process, we define a curve in the data, such that for those times and positions, all values at earlier time are simply set to zero.

10.3.4 Identifying waves to be muted

On any part of the data where the waterbottom has more or less the same sort, we can create an average of all shot profiles in that area. In the case of these data, the entire dataset has a fairly flat waterbottom. We can resort the data so that it is in increasing offset, with the traces of the same offset side by side

```
$ susort dt offset < seismic.su > junk1.su
```

where we have chosen **dt** as the first parameter because it is a header field that is the same for every trace. We then can stack the data

```
$ sustack key=offset < junk1.su > supershot.su
```

so that each resulting trace is the average of all of the trace at that offset. The effect is that **supershot.su** is an average of all of the shot gathers in the data. We may view the **supershot.su** by putting some display gain on this with **sugain** and display with **suxwigb**

```
$ sugain jon=1 < supershot.su | suxwigb key=offset
```

It is important for picking that **suxwigb** is run with **key=offset** so that the horizontal scale will be in offset.

10.3.5 How to pick mute values.

On the super shot gather, we may pick (time,offset) pairs to supply to **sumute**. This may be done by placing the cursor on the desired point, and typing the letter "s". The ordered pair of time and offset will be printed on your terminal window.

Once the desired times and offsets are obtained, the entire dataset may be muted via

```
$ sumute < seismic.su  tmute=t1,t2,...  xmute=x1,x2,... \
                                        key=offset > mute.seismic.su
```

You can make this a bit more automated by doing the picks via following:

```
$ suxwigb key=offset perc=99 mpicks=picks.txt < supershot.su  &
```

and as before, pick values by placing the cursor on the desired location on the plot and typing the letter "s". When you are finished type "q" and the picks will be written into the file **picks.txt**. Now use **mkparfile** to make a par file for sumute. This is done by typing

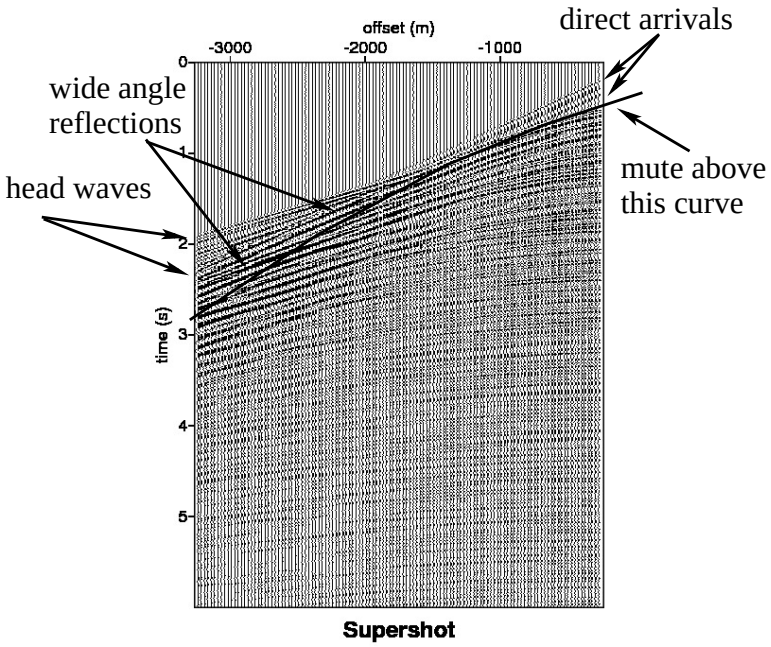


Figure 10.10: An average over all of the shots showing direct arrivals, head waves, wide angle reflections, and a curve along with muting may be applied to eliminate these waves.

```
mkparfile string1="tmute" string2="xmute" < picks.txt > mute.par
```

Then you would do:

```
$ sumute < seismic.su  par=mute.par key=offset > mute.seismic.su
```

then you proceed with sorting **mute.seismic.su** into CDP gathers and then multiple suppressiona and velocity analysis.

10.3.6 The shape of the wavelet

It may have occurred to the reader that we have done nothing to ensure that the waveform is actually optimal for stacking. Seismic signals are assumed to have what is called the "minimum phase" property, which is to say that most of the energy is located at the beginning of the wave form. Manufacturers of marine air guns do their best to fit this prescription, but there are still issues that make real seismic data deviate from this assumption.

For air guns, there is a reverberation known as a "bubble pulse" which is caused by a reverberation of the air bubble generated by the air gun. The second problem with marine data is known as "ghosting." Ghosting is caused because there is are reflections off of the water surface at both the source and the receiver that tend to turn the waveform into something more like a doublet or triplet. Ghosting is evidenced by a notch in the spectrum of the seismic data.

These deviations from perfect minimum phase are handled by deconvolution. Indeed, we may also use *predictive deconvolution* to suppress multiples.

10.3.7 Further processing

We may consider processing the full dataset with the shell script **Radon.final**

```
#! /bin/sh

# input your data sorted in cdps.
# susort < data.su cdp offset > data_cdp.su
data=data_cdp.su
radondata=radon.$data

# the tnmo=  and vnmo= values go in a text file called "radon_nmo_vel.par"
# you can use the radon_nmo_vel_x.par file made by Velan.radon here
parfile=radon_nmo_vel.par

interoff=-262
offref=-3237
depthref=1000

# do pmin and pmax need to be this big?
pmin=-2000
pmax=2000
dp=8
igopt=2
lenx=7
xopt=1

# turn of stretch mute, sometimes suradon fails on muted data
smute=20

# The values of pmula and pmulb define the filter in the Radon domain
# the values set here may not be optimal, but are offered as starting values
# we rely on the choice of tnmo= and vnmo= to separate data and multiples
pmula=20
pmulb=200

# filter data in radon domain
choose=1  # data - multiples
sunmo par=$parfile smute=$smute <  $data |
suradon xopt=$xopt lenx=$lenx offref=$offref depthref=$depthref  \
```

```
pmula=$pmula pmulb=$pmulb interoff=$interoff pmin=$pmin \
pmax=$pmax dp=$dp choose=$choose igopt=$igopt |
sunmo par=$parfile invert=1 smute=$smute  > $radondata



\n up (don't use this if there are multiple radon jobs running)
#/bin/rm -f radontmp*
exit 0
```

It takes several hours (yes, that's right *hours* perhaps 5 to 6 hours, maybe longer to run this process on the full dataset.

We may view the results of the Radon transform multiple suppression by making an NMO correction to some relevant speed, such as those used in assignment #5 and stacking to make a brute stack of the data, for example

```
$ sunmo vnmo=1500,2200,3000 tnmo=0,1,2 < radon.gain.jon=1.cdp.su key=offset |
      sustack | suximage perc=99 &
```

If multiples are still prominent, then we may need to perform the τ-p filtering with better parameters, or we may need to apply different filters to different ranges of CMPs.

10.3.8 The at command: using the computer while you are asleep

There is a famous adage that you cannot get rich unless you can find a way to make money while you are asleep. In our case, we don't efficiently use the computer unless we can run jobs when we are not physically at the machine, such as when we are at home or asleep. Such processing jobs are called *batch* jobs. A batch job is a process that is submitted for later execution. When computers were first invented, all jobs were batch jobs.

Batch jobs using the at command

On Unix and Unix-like systems, the **at** command allows processing jobs to be run at a pre-specified time by the user, whether or not the user is logged in on the system. The Unix man page for **at**

```
$ man at
```

shows the basic usage.

Suppose you have a shell script called **Myshell** located in some directory /mydirectory. If you wanted to run this script in the middle of the night, you could do the following:

```
$ cd /mydirectory
$ at 1am tomorrow -f Myshell
```

An important point is that you need to have the *-f*. (Today is 13 October 2011, for this discussion.) To see if your **at** job is on the list of jobs to be executed, you would type

```
$ atq
2       Wed Oct  13 01:00:00 2009 a yourusername
```

The first number is a number designating a job number, the rest of the fields show the date and time of execution, and your username on the system.

You are now free to log out, and go on with other tasks. The system will send you an email message. However, if you have special messages that you want the script to email you, you might have lines like these in your script

```
echo "Run completed" |
/usr/bin/mail -s "Status of job "  myusername@mymail.mines.edu
```

Here, the **echo** command is sending a message to your CSM email address with Subject line saying "Status of job" and message contents saying "Run completed." You could have other information emailed to you. If your script fails, the system will email you with that information as part of the standard operation of **at**.

Before using this for anything big, try rerunning the **Migtest** scripts that you ran for Homework Problem #4 as **at** jobs. For example, try running **Migtest.gb** at 1 am tomorrow morning

```
$ cd Temp4
$ cp Migtest.gb .
$ cp newvelzx.bin .
$ cp seismic3.su .
$ at 1am tomorrow -f ./Migtest.gb
```

Then log out, and check your email in the morning.

Make sure that it works as desired before running bigger jobs. The script should run as before, and you should get an email with the same screen output that you saw when you ran this on the commandline in the lab. There will be an error because an **at** job cannot open an X-window for **suxwigb** but everything else should be ok.

To get your email, you may need to have a file called **.forward** (dot forward) in your home directory on the lab machines, with your email address (e.g. youruser-name@mymail.mines.edu) as the single line of text in the file.

Killing an at job

We all make mistakes. Sometimes we launch **at** that we want to remove. If it turned out that you changed your mind, and didn't really want to run the job, you would type:

```
$ atq
2       Wed Oct  7 01:00:00 2009 a john

$ atrm 2
```

where this is the job id number that is in the first line of the **atq** output.

10.4 Homework Assignment #7 due Thursday 15 Oct 2015 and Tuesday 27 October 2015, before 9:00 AM.

Perform the following operations:

- Mute and perform gaining on your data, then perform the analysis that you did in Homework #6 to a single CDP from each of these gathers to get $tnmo =$ and $vnmo =$ values for multiple suppression. You will have a different set of $tnmo =$ and $vnmo =$ values for each subset of the data.

- Break your data in to several parts via

  ```
  $ suwind < gain.jon=1.mute.cdp.su key=cdp
                  min=0 max=500 > gain.jon=1.cdp=0-500.su
  $ suwind < gain.jon=1.mute.cdp.su key=cdp
                  min=501 max=1000 > gain.jon=1.cdp=501-1000.su
  $ suwind < gain.jon=1.mute.cdp.su key=cdp
                  min=1001 max=1500 > gain.jon=1.cdp=1001-1500.su
  $ suwind < gain.jon=1.mute.cdp.su key=cdp
                  min=1501  max=2142 > gain.jon=1.cdp=1501-2142.su
  ```

  Note that the input file is the full dataset, sorted into cdp gathers, gained, and muted.

- Then adapt the shell script **Radon.final** located in /data/cwpscratch/Data5 to perform the radon multiple suppression on each of these subsets of the data using the respective **tnmo=** and **vnmo=** This part is time consuming, taking several hours for each part so **start early**. Note that the **tnmo=...** and **vnmo=..** values go in a files with names specified on the line in the shell script **Radon.final** that starts **par=filename**. The default filename is **radon_nmo_vel.par**. (All SU programs have a hidden feature that the commandline argument for parameters can by kept in a file, called a "parfile," and read into the appropriate program via **par=parfilename**).

- You should have at least one of these subsets processed for class next week. Perform the "brute stack" operation from Homework #5 on that subset. For your report, show the semblance plot of the single CDP gather after multiple suppression, and show the brute stacks of the 500 CDP panel for the data before and after multiple suppression.

- If you succeed in performing the multiple suppression on all 4 subsets of the data, concatenate these together to form the full processed dataset via a command of the form

  ```
  $ cat radon.gain.jon=1.cdp=0-500.su radon.gain.jon=1.cdp=501-1000.su
          radon.gain.jon=1.cdp=1001-1500.su
             radon.gain.jon=1.cdp=1501-2142.su > radon.gain.jon=1.cdp.su
  ```

  In this case, your report should show the semblance plot of a single CDP gather and show a brute stack as in Homework #5 of the full processed dataset and submit that, instead. Remember to show all commands and their parameters that perform actual processing steps. (Good for 5 more points. Please note. This is not extra credit. It is an "obstacle." This is a 15 point problem, but not doing the second part will not be considered incomplete.)

You might want to read about the **at** command in the previous section, before doing this assignment.

Hint: making a radon_nmo_vel.par file

To see what the structure of the **radon_nmo_vel.par** file is, note that you put **tnmo=** and **vnmo=** pairs in the **radon_nmo_vel.par** file as you would type them on the commandline. You can also have entries on different lines, and have blank line separators. One example would be to have for the contents of radon_nmo_vel.par for the entire dataset:

```
tnmo=0,1,2
vnmo=1500,1800,2300
```

Or if you had a bunch of them for specific CDPs the **radon_nmo_vel.par** file would look like:

```
cdp=265,589,1087,1900
tnmo=0,1,2
vnmo=1500,1800,2300
tnmo=0,1.3,2,2.5
vnmo=1500,1900,2300,2350
tnmo=0,1.3,2.2
vnmo=1500,1900,2320
tnmo=0,1.3,2.2
vnmo=1500,1900,2320
```

It may be, however, that for your 4 blocks of CDPs you need to create 4 separate **radon_nmo_vel.par** files containing the **tnmo=** and **vnmo=** pairs for that particular block.

Note that the numbers in these examples are totally made up. You don't have to have the same number of velocities and times in each pair. Within a pair you have to have the same number of times as velocities, but the pairs themselves can have differening numbers of velocities reflecting the values you pick from the semblance.

10.5 Concluding remarks

Our discussion of velocity analysis and multiple-suppression by the radon transform here is just a warm-up for a more "production level" treatment in Chapter 12.

Obviously, there are many issues that come into play when performing velocity analysis. We do velocity analysis for multiple suppression, but we also are doing velocity analysis for stacking. We do not know *a priori* what the correct NMO velocities are, so there is an iterative aspect to the process. Once we have suppressed the multiples we may recognize that our gaining is not very good and we likely need to regain the data and pick NMO velocities again. Remember also, that we should have muted certain arrivals at the beginning of all of these operations.

Finally, there is the issue of time. Time is money. We cannot afford to be perfectionists! There is a line that has to be drawn. We must not expect more out of our data than we have a right to expect, but we must also not give up too easily.

References

- Philip S. Schultz and Jon F. Claerbout (1978). Velocity estimation and downward continuation by wavefront synthesis. Geophysics, 43(4), 691-714.

- Radon, Johann (1917), "ber die Bestimmung von Funktionen durch ihre Integralwerte lngs gewisser Mannigfaltigkeiten", Berichte ber die Verhandlungen der Kniglich-Schsischen Akademie der Wissenschaften zu Leipzig, Mathematisch-Physische Klasse [Reports on the proceedings of the Royal Saxonian Academy of Sciences at Leipzig, mathematical and physical section] (Leipzig: Teubner) (69): 262277; Translation: Radon, J.; Parks, P.C. (translator) (1986), "On the determination of functions from their integral values along certain manifolds", IEEE Transactions on Medical Imaging 5 (4): 170176,

- Taner, Turhan and Fulton Koehler (December 1969). "Velocity Spectra-Digital Computer Derivation and Application of Velocity Functions". Geophysics 34 (6): 859881.

Chapter 11

Spectral methods and advanced gaining methods for seismic data

There is a class of methods that are best called *spectral methods* because they modify the amplitude and/or phase spectrum of the data. There are several reasons for performing such operations. We may find that there is high or low frequency noise in the data that may be present in the original data, or which may be introduced by processing tools. The second is to suppress multiples, and the third is to sharpen the waveform to more clearly define reflection arrivals. There is, of course, the issue of correlating vibrator profile data with the vibroseis sweep as a prelude to further processing.

Some of these techniques are clearly *filtering* operations, wherein a particular filter is *convolved* with the data. Other techiquess are best thought of as a *deconvolutional* processes, that is to say, a process by which data are "inverted" in some sense, to remove a particular response. We may think of both operations as being related, in that deconvolution is a convolution with an inverse of a signal. He we find that deconvolution is also a "filter."

There are many such methods that have been developed over the decades since digital data processing was first introduced in seismic data processing in the mid 1950s, but we will discuss only a small subset of these, to give you a general idea of what to expect in a commercial environment.

We apply many of these operations prior to velocity analysis, or in conjunction with velocity analysis, or as a prelude to migrating the data.

11.1 Common assumptions of spectral method processing

Seismic data result from the introduction of seismic energy into the subsurface followed by the subsequent recording of reflections either on the surface of the earth or in a well bore. Such data have a natural frequency band and a natural phase spectrum. We may seek to change some of these characteristics as part of processing, but we have

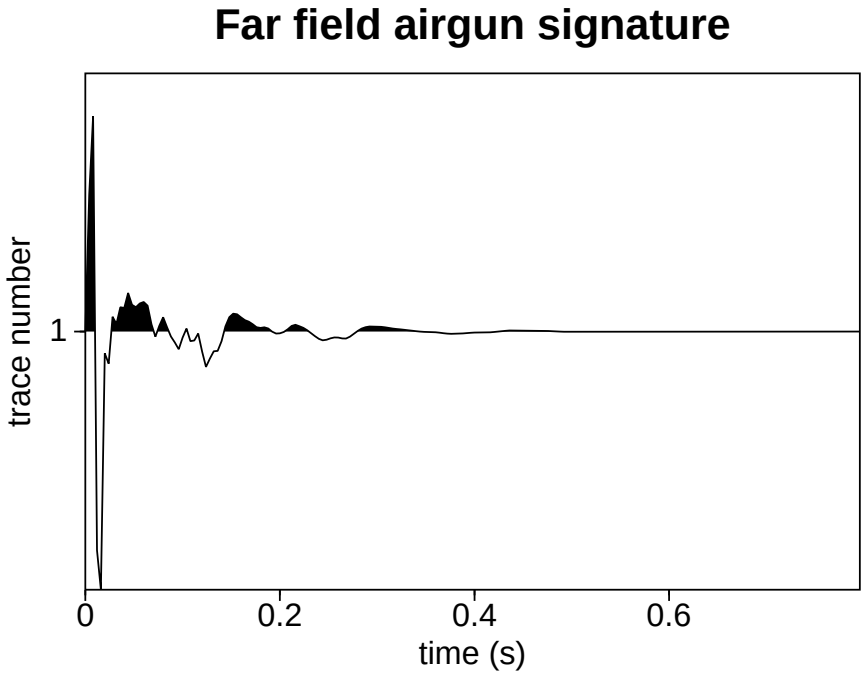


Figure 11.1: Example of a far-field airgun source signature

common assumptions about our data that we make, or that we impose on the data as a precondition for further processing.

The seismic source may have a time history that makes it appear complicated. For example, a marine air gun signature may have a *bubble pulse* that follows some time after the main signal, see Figure 11.1. Because we may think of all resulting reflections as resulting from a convolutional process involving this source wavelet, complications in the source waveform may cause the reflections to appear unduely complicated.

Another source of waveform complication in ocean seismic surveys results from a phenomenon called *ghosting*. In addition to the direct arrival from the source, there is an additional arrival that originates from a reflection path that begins at the source, bounces off of the water surface, and then travels to the subsurface. Similarly, in addition to the direct reflection from the subsurface, the receiver may record an additional signal that has traveled to the ocean surface. These ghost reflections are also convolved with the reflectivity series.

Yet another source of data complication are multiples. These include reverberations in the water column, with waves bouncing between the sea surface and the sea bottom. This is not just the first reflection from the sea bottom, but can include large portions of the seismic arrivals. Again, these reverberations, called *pegleg* (named for a pirate's artificial limb) multiples. While it may be tempting to think of multiples as being merely added to the data, these too act as secondary sources and are thus convolved with the

data.

Finally, there may be ambient noise which is added to the data. This can include cultural noise, or natural noise from such sources as wind, or in ocean surveys noise from sea creatures or from ships, including the ship that is dragging the towed airgun/hydrophone array.

To deal with these issues, we apply deconvolutional methods. To apply such methods, we make some physically reasonable simplifying assumptions. These are *causality* and the *minium phase (delay)* and *white spectrum* assumptions. Underlying all of this is the assumption that seismic wave propagation is a *linear system.*

11.1.1 Causality

If the source is an explosion or a pulse from an airgun, or even a sweep from a vibrator, the resulting data have the property that they are *causal*, which is to the say that the wavelets have a definite beginning time. Causality means that there can be no signal before time zero, or in the case of propagating arrivals, there can be no arrivals before the shortest traveltime determined by the velocity function for the medium. This, of course, is a basic principle of physics and should not be a surprise.

Many processes that change the frequency spectrum also will tend to cause a distortion in the waveform resulting in signals that may appear later than the time predicted by the wavespeed. Sometimes we deliberately change the phase characteristics of the wavelets in the data as to make the wavelet symmetric about the expected arrival time, thus giving the appearance that energy is coming in a bit earlier than the predicted arrival time. In this case, we Such signals are called *zero-phase* waveforms. If the data are then processed to appear to be similar to sinc functions, which is to say, symmetric zero-phase signals, then the reflectors will occur at times of the peaks of these "bandlimited delta functions".

11.1.2 Minimum phase (aka minimum delay)

A signal which has a definite time of beginning and also has the majority of its energy in the beginning part of the waveform is called a *minimum phase (aka minimum delay)* wavelet. There is a precise mathematical definition, but physically we may consider any "front-loaded" signal to be or to approximately a minimum phase waveform. Again, many linear, or mildly nonlinear physical processes will produce signals that have this property. Errors that we see, then could be expected to result from a deviation from this frontloadedness.

11.1.3 White spectrum

The term "white spectrum" is an allusion to the notion of white light being composed of a full visual spectrum of frequencies. For seismic data, many processes that we apply would become unstable if amplitudes at particular frequencies were to be zero or near zero. For *deterministic processes* the instability comes from division by zero or division by a small number, as might be encounted by performing deconvolution in the Fourier

transform domain. For *statistical processes* which are viewed in some sense as matrix operations, the issue is to introduce a perturbation that moves the system away from null space of the matrix in the representation.

We know that all data are bandlimited so the remedy is to include a small "white noise" term that will prevent this instability. The expense is that all such operators will introduce noise. Generally, frequency filtering is a remedy for this noise issue. Such a white-noise parameter is usually a small number multipled by the maximum of the autocorrelation, so it reflects the size of the amplitude spectrum.

There are four operations that are the working tools of digital singnal processing. These are *convolution*, *cross-correlation*, *autocorrelation*, and *deconvolution*.

11.1.4 Linear systems

Geophysicists have realized a tremendous benefit from a simple concept. That is the concept called the *convolutional model* of seismic wave propagation. Because the wave equation is a linear partial differential equation equation, the simplest way of looking at all processes involving the wave equation is to consider the processes as being a *linear system*.

The common metaphor is that of a "black box." That is, we are assuming that a geophysical process is a linear system with an input and an output, but the only other thing that we know about the black box is that we *assume* it behaves in a linear fashion. That is, we assume that the output from a process depends linearly on the input.

A result known as "Green's theorem" tells us that if we have a particular solution called the "Green's function" of the linear system, which is to say the output of the linear system given the input of a Dirac delta function, we may form all possible solutions by the convolution of a given input with the Green's function. The Green's function is also known as the "impulse response" or the "transfer function." (In modern mathematical usage the term "fundamental solution" is often seen.) This is a feature of the principle of superposition.

We have three mathematical systems that we will switch back and forth freely between. The first is the *continuous* representation. In this representation we write signals and filters as continuous functions of time, apply the Fourier transform, and perform computations in the *frequency domain*. While this formulation is straightforward and yields a great deal of insight into our problems, this is a more computationally expensive approach than to perform our operations in the time domain.

When we represent data as discrete samples in the time-domain, there is a representation known as the Z-transform, which allows digital interactions to be represented as the multiplication and division of signals and filters as polynomials.

Yet a third representation is to note that operations that are represented as multiplications of discretely represented time-domain signals may be represented in linear algebra form as matrix multiplications, or matrix-vector multiplications.

Each of these representations has its own merits in perimitting insight into the processes being discussed.

What does "linear" mean and why is it good?

In the experience of mathematical physics, many processes can be described or may be approximated by linear ordinary or partial differential equations. The short answer of why linear is good, is that linear makes the mathematics easier.

If multiply the input to a linear system by a scalar then the output is scaled by the same value. If we shift the input to the black box, the output is shifted by the same amount. That's it.

Furthermore, linear systems have a property called *the principle of superposition*. This means that new solutions of a linear system may be found by forming the linear combination of previously determined solutions. This fact allows transform theory to be applied. That is, we may use invertible mathematical transformations to decompose input functions into a family of simpler functions, apply the linear system to these simpler functions, and then add up the results to yield the output for the full function.

11.2 The three mathematical languages of signal processing

There are three formulations that are useful in signal processing. These are the *continuous function*, *the Z polynomial*, and the *matrix and vector* representations.

The continous function representation is made via the Fourier transform. Though, formally we can write signals, wavelets, and filters as contiuous functions, these are applied discretely as digital data equivalents.

The Z-transform representation replaces all functions with a polynomial representation that has many of the same properties of the frequency domain representation.

Finally, we can represent digital data operations as matrix-on-matrix, or matrix-on-vector multiplications.

11.2.1 The Forward and Inverse Fourier Transform

The most common decomposition is called the *Fourier transform* which decomposes a given signal into sines and cosines. The formal representation of the forward Fourier transform

$$F(\omega) = \int_0^\infty f(t) e^{i\omega t} dt \tag{11.2.1}$$

and the inverse Fourier transform

$$f(t) = \int_{-\infty}^\infty f(\omega) e^{-i\omega t} d\omega. \tag{11.2.2}$$

Here $\exp(i\omega t) = \cos(\omega t) + i \sin(\omega t)$. In the first expression, the function $f(t)$ is decomposed into to the values of all of the sines and cosines of different frequencies. The inverse process collapses the sines and cosines, sums up the values for each frequency, yielding the original function $f(t)$ back.

We can use this definition of the forward and inverse Fourier transforms to formulate *convolution*, *cross correlation*, and *autocorrelation*.

11.3 Convolution, cross-correlation, and autocorrelation

There are three related operations that are encountered in signal processing. Thes are *convolution*, *cross-correlation*, and *autocorrelation*.

11.3.1 Convolution

The formal definition of convolution is given by the continuous integral relation

$$F(t) \star G(t) = \int_{-\infty}^{\infty} d\tau F(\tau) G(t-\tau). \tag{11.3.1}$$

This is not a "definition" but a relation that arises naturally when solving boundary value problems using Green's theorem. If we replace F and G by their inverse Fourier transform definitions, we obtain the result that

$$F(t) \star G(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(\omega) g(\omega) \exp(-i\omega t) d\omega, \tag{11.3.2}$$

which means that *convolution is multiplication in the frequency domain*.

11.3.2 Lab Activity #18: Frequency filtering

The simplest, yet one of the most important spectral methods is simple *frequency filtering*. If we look at the spectra of the traces in CMP gather 265

```
$ suspecfx < gain.jon=1.cdp=265.su
                    | suxgraph  title="spectra"
                            title="spectra" label1="frequency"
                                                label2="amplitude" &
```

we see that the data have most of their frequency spectral values between 5 Hz and 80 Hz. The rest can be considered to be noise, which could be boosted and distorted by futher processing steps. When engaging in signal processing the old computer science adage "garbage in garbage out" it is often amplified to "small garbage in, a lot of garbage out."

Applying simple frequency filtering with **sufilter** we see in the frequency domain

```
$ sufilter < gain.jon=1.cdp=265.su
                f=0,5,70,80 amps=0,1,1,0 | suspecfx
                    | suxgraph  title="spectra"
                            title="spectra" label1="frequency"
                                                label2="amplitude" &
```

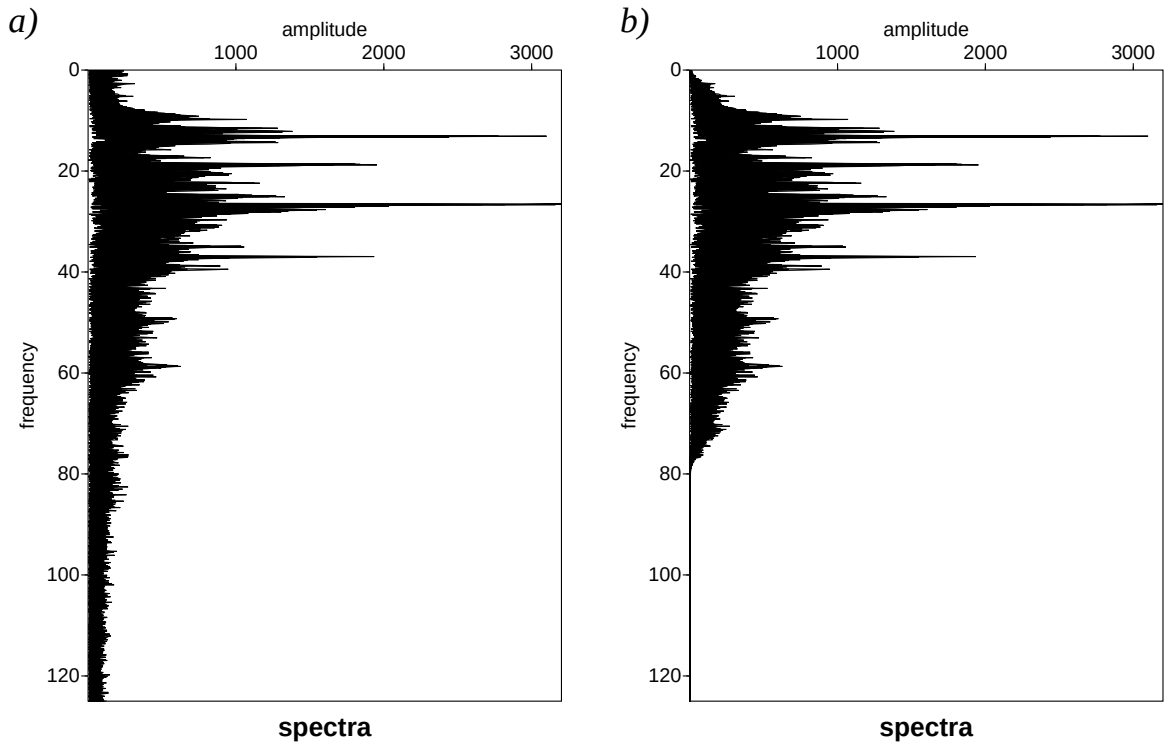


Figure 11.2: a) Amplitude spectra of the traces in CMP=265, b) Amplitude spectra after filtering.

that the data are truncated. Note that it is up to the user to figure out the full range of frequencies in the data that are to be kept. It may take some experimentation with further processing steps to find the correct filter range.

The program **sufilter** applies only simple tapered zero-phase filters to the data. There is another class of filter known as "Butterworth" filters, which in SU may be performed via the program *subfilt*. Butterworth filters are described by a solution a class of ordinary differential equation. Such filters were originally applied as analog preprocessing during the time of data acquisition.

Are we done with frequency filtering? Often, not. Other spectrum modifying processes, as well as other transforms we may use, may introduce noise into the output. We may need to apply our simple bandpass filter again owing to these noise sources.

11.3.3 Lab Activity #19: Spectral whitening of the fake data

It may have occurred to the reader after seeing the spectra of the real data that it may be of benefit if the spectra of the traces were flat, instead of having the many peaks and valleys. This is the process known as *spectral whitening*. We expect that such an operation would tend to sharpen data, but with the caveat that we know it will sharpen the noise, as well, making everything more "spike-like."

The basic idea is simple. We take the data into the frequency domain, and consider the representation of the data $d(\omega)$ as a complex-valued function

$$d(\omega) = |d(\omega)|e^{i\phi(\omega)}.$$

We then multiply the amplitude $|d(\omega)|$ by a function $1/|d(\omega)|$, taking care not to divide by zero, so that $|d_{\text{new}}(\omega)| = 1$. This may be over the full range of 0 to the Nyquist frequency, or over some partial range. We then inverse Fourier transform the data. This operation is guaranteed to change the relationship between the amplitude and the phase function, as

$$d(\omega) = d_r + id_i.$$

where d_r is the called the *real part* of d and d_i is the called the *imaginary part* of d. Here, the amplitude is given by the *modulus* of d as

$$|d| = \sqrt{d_r^2 + d_i^2}$$

and the phase by

$$\phi = \arctan\left(\frac{d_i}{d_r}\right)$$

We can experiment with spectral whitening in SU using the command **suwfft**. This program gives the user the choice of whitening, from moderate to extreme. The plots labeled "traditional" use the default settings of the program, which are not really full spectrum whitening.

For example

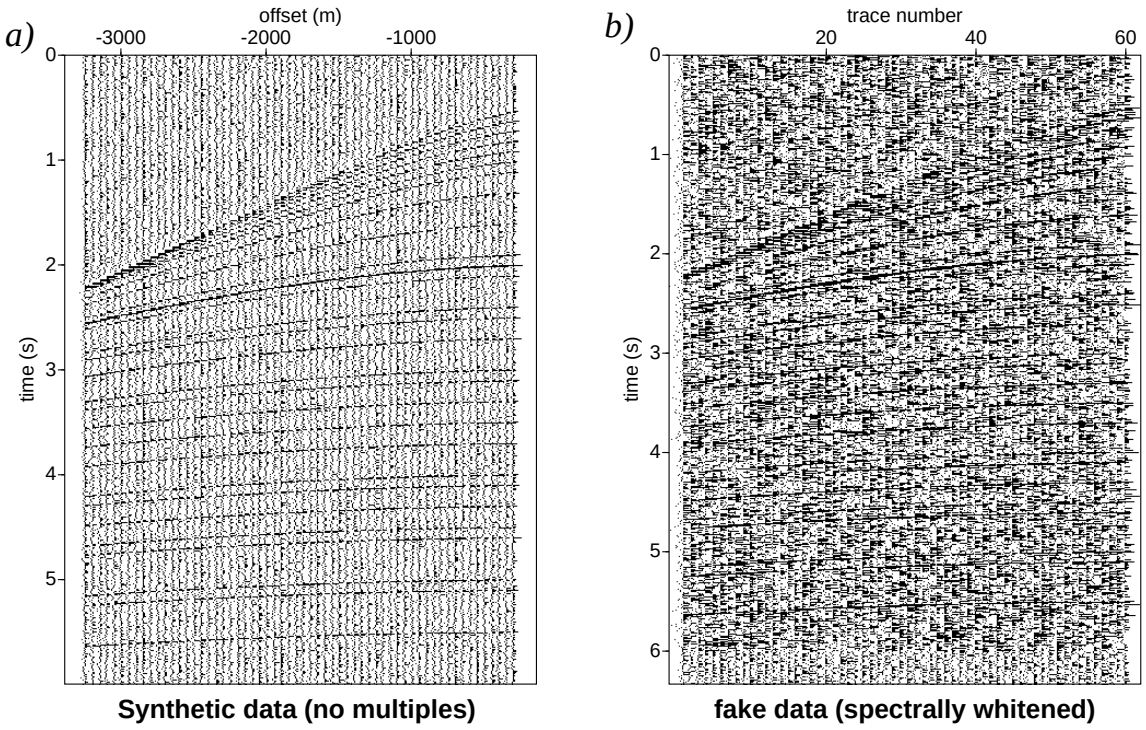


Figure 11.3: a) Original fake data b) fake data with spectral whitening applied. Note that spectral whitening makes the random background noise bigger.

```
$ suwfft < fake.su w0=0 w1=1 w2=0
                      | suifft | suxwigb xcur=2 title="fully white spectrum"
$ suwfft < fake.su
                      | suifft | suxwigb xcur=2 title="traditional spectral whithening"
```

and

```
$ suwfft < gain.jon=1.cdp=265.su w0=0 w1=1 w2=0
              | suifft | suxwigb xcur=2 title="fully white spectrum"
$ suwfft < gain.jon=1.cdp=265.su w0=0 w1=1 w2=0
              | suifft | suxwigb xcur=2 title="traditional spectral whitening"
```

show us what this does to our data. The data are sharpened, but then so is the noise.

Another test is to apply spectral whitening to our CMP 265 data. Recall that frequency filtering may need to be applied before and after the whitening process. We see that the spectrum is an idealized white amplitude spectrum whose shape is the filter

```
$ sufilter f=0,5,80,90 < gain.jon=1.cdp=265.su |
     suwfft w0=0 w1=1 w2=0 | suifft | sufilter f=0,5,80,90
            | suspecfx | suxgraph title="Spectrum after whitening" &
$ sufilter f=0,5,80,90 < gain.jon=1.cdp=265.su |
     suwfft | suifft | sufilter f=0,5,80,90
            | suspecfx | suxgraph title="Spectrum after whitening" &
```

and though it appears to be a single curve, these are really 55 identical spectra on on top of the other, for each of the 55 traces in our gather.

The spectral whitening process involves the cascade of forward and inverse Fourier transforms. Owing to zero padding in these transforms, there may be more samples per trace on the output than on the input, so an extra step of windowing the data with **suwind** in time is required

```
$ sufilter f=0,5,80,90 < gain.jon=1.cdp=265.su |
     suwfft | suifft | sufilter f=0,5,80,90  |
         suwind itmin=1 itmax=1500 |
           | suxwigb title="data after traditional spectral whitening" &
```

(Don't forget the **suifft** step!) The windowing passes samples from sample 1 through sample 1500 on each trace. As with our fake data, the real data have additional arrivals.

We can run **Radon.test** on versions of the data after spectral whitening has been applied. Observe the changes in the semblance plots after spectral whitening. It may be that we would prefer to use spectral whitening after multiple suppression.

Should we run spectral whitening? The type of spectral whitening we discuss here is a brute force modification of amplitudes, which will certainly introduce noise into the output. Frequency filtering likely will be needed to remove frequency information that is totally fabricated by the spectral whitening process. We run spectral whitening (and spiking deconvolution) to improve resolution of velocity picks and to make reflectors sharper. To the end that the tools do that job, we may apply them. You definitely need to experiment with the operation to see if it helps.

11.4 The Discrete Representation of Seismic Data

In the past section, we discussed the application of filters that modify the spectrum of the input seismic signal. Though we are implementing the operations on sampled data, we use a continuous function representation of the processes that were being applied to formlulate each technique.

We find, however, that the act of digitizing a signal introduces its own peculiarities.

11.4.1 The Forward and Inverse Z-transform

If we take the (causal) Fourier transform of our reflectivity series $R(t)$, we obtain a series in terms of shifted complex exponentials

$$\begin{aligned}\hat{R}(\omega) &= \int_0^\infty R(t)e^{i\omega t}\,dt = \int_0^\infty \sum_{k=0}^{N} R_k\delta(t-\tau_k)e^{i\omega t}\,dt \\ &= \sum_{k=0}^{N} R_k e^{i\omega\tau_k} = \sum_{k=0}^{N} R_k e^{i\omega k\Delta t}\end{aligned}$$

where we note that the time sampling interval is a constant Δt such that $\tau_k = k\Delta t$. If we define $\phi = \Delta t$. The "Z" in Z-transform is this shifted complex exponential, $Z = \exp\{i\omega\Delta t\} = \exp\{i\omega\phi\}$. The "transform" is the transformation of a this sampled data into a polynomial in Z. This polynomial is called the *Z-transform* representation of $R(t)$

$$R(Z) = \sum_{k=0}^{N} R_k Z^k.$$

The advantage of this representation is that we have effectively taken the Fourier transform of our initial digitally sampled data by inspection!

All we have to do is to multiply the k-th term of our sequence digital value with Z^k and add up the resulting terms to form the k-th order polynomial in Z. For a Z-transform representation of a finite number of terms this is all we need to know.

The Z-transform representation inherits the property that convolution and deconvolution of signals is represented as multiplication and division, respectively of the transformed data, as we see with Fourier transformed data.

More mathematics

For an infinite series, such as we might obtain through a Taylor expansion of a function or through a process of long division as in the case of the geometric series, we must also specify a region of convergence, which is a circle in the complex plane. Poles in the Z-transform representation will lie outside that circle of convergence. Zeros of the polynomial may lie inside the unit circle of convergence.

11.4.2 The inverse Z-transform

We can use the properties of the residue theorem from complex variables to get our original series back. The inverse Z-transform has to have the form

$$R_k = \frac{1}{2\pi i}\int_C R(Z)Z^{k-1}\,dZ, \tag{11.4.1}$$

for each term where the contour C encloses the origin.

We can see why this is so by considering that for $Z = \exp\{i\omega\phi\}$ and for C being a circular contour enclosing the origin. Simply substituting for Z and noting that $dZ = iZd\phi$

$$I = \int_C Z^{-1}\,dZ = i\int_0^{2\pi} e^{-i\omega\phi}e^{i\omega\phi}\,d\phi = 2\pi i,$$

which shows where the division by $2\pi i$ comes from.

The other possibility is to consider for $n \geq 0$

$$I = \int_C Z^n dZ = 0.$$

This integral vanishes by Cauchy's theorem because the integrand is an analytic function of Z. The contour integral in equation (11.4.1) sifts through the each term of the series representation of $R(Z)$ and returns the original sequence of values as discrete values, giving us our original series of digital samples back.

The inverse Z-transform is effectively the inverse Fourier transform, as long as the contour C is the unit circle $|Z| = 1$.

11.5 Deconvolution

We have another way of whitening the spectrum. This method is to deconvolve the data. This may be either a deterministic process, where an estimate of the wavelet is obtained, and is divided out of the data in the frequency domain. More commonly we apply deconvolution via a *statistical estimate* of the wavelet, which is based on the assumption that the data are minimum phase (aka minimum delay), under an error minimization criterion. Thus we assume that our entire data consists of spikes convolved with minimum phase (aka minimum delay) wavelets.

Before launching into the application of minimum phase (aka minimum delay) deconvolution, we discuss the operations that we will be applying to data in general mathematical terms.

11.5.1 Convolution of a wavelet with a reflectivity series

The simplest model of a seismic trace is to consider the notion of the *reflectivity series*. The idea is simple. The world is assumed, to first order, to consist of simple reflectors, each with its own arrival time and its own reflection coefficient $\mathcal{R}_k$ for the k-th reflector.

Seismic waves are represented as rays that travel from the source to each reflector and back, taking τ_k for the two-way traveltime. In this ideal world, we have only single scattering, so there are no multiples (yet).

The simplest seismogram that could be recorded would then be a collection of spikes of each of a respective height $\mathcal{R}_k$, having values which could be positive or negative, arriving at the respective time τ_k

$$R(t) = \sum_{k=1}^{N} \mathcal{R}_k \delta(t - \tau_k)$$

where $\mathcal{R}_k$ is the reflection coeficient of the k-reflector and $\delta(t - \tau_k)$ is the Dirac delta. We may think of this Dirac delta as a spike that only "turns on" when $t - \tau_k = 0$ and is "turned off" (zero) the rest of the time.

This series is called the *reflectivity series*—a popular notion in exploration seismology. If we want to make a seismogram, then we would convolve a wavelet $W(t)$ with the reflectivity series to form a seismogram. **Note that when we write the reflection coefficient of the k-th reflector, we write that as $\mathcal{R}_k$, but when we write the k-th sample of the digitized reflectivty series $R(t)$, we write R_k.**

A remarkable result of digital signal processing is that the process of digitizing a signal yields the Z-transform of a function. Simply stated, the Z-transform of a signal is polynomial representation of the discretely sampled signal with the k-th sample multipled by the factor Z^k.

Thus, without taking expensive Fourier transforms, we are able to perform convolution and deconvolution of digital data by serial multiplication of digital representations, often making digital data processing in the time domain inexpensive. This "serial multiplication" is the multiplication of the Z-transform polynomial representation of the given functions.

The division of polynomial representations would then be deconvolution. As with division in the Fourier domain, the issue of avoiding division by zero also critical in the Z-transform representation of deconvolution.

Minimum phase in Z-transforms

In the world of digital signal processing in geophysics, we are dealing with causal functions. This means that there is a specific beginning time for signals. We also have the issue of where the energy is in the signal.

We call waveforms that are "front loaded" or contain most of their energy at the beginning of the wavelet "minimum phase" (aka minimum delay) signals,

Such a Z polynomial could be of some degree m but it might be that only the first few terms of the Z polynomial representing the wavelet are of importance, the rest could be near zero, which is to say that the low degree terms in the Z-transform contribute the most.

11.5.2 Convolution with a wavelet

Our digital data are convolved with a wavelet given by $W(t)$

$$\begin{aligned} D(t) &= W(t) \star R(t) \\ &= \int_{-\infty}^{\infty} W(\tau) R(t-\tau) d\tau \\ &= \frac{1}{2\pi} \int_{-\infty}^{\infty} s(\omega) w(\omega) e^{-i\omega t} d\omega. \end{aligned}$$

Thus, recorded data $D(t)$ is the convolution of a wavelet $W(t)$ with the reflectivity series $R(t)$. The last line shows the Fourier transform domain form of convolution. **Convolution is multiplication in the frequency domain.**

Convolution of Z-transform representations

In the language of Z-transforms, convolution of two signals is the multiplication of the two polynomial representations in Z of the functions.

11.5.3 Deconvolution

Deconvolution then is the inverse process, which is to say, the process of removing the effect of a waveform, to produce a desired output.

Symbolically, we have recorded data $D(t)$ with a particular waveform $W(t)$ possibly distorting the arrival time of a given reflection. What we want ideally is to reconstruct the reflectivity series by applying the inverse process

$$R(t) = W^{-1}(t) \star D(t).$$

We need only determine what the "inverse of $W(t)$" given by $W^{-1}(t)$ is. If we write this out in the Fourier domain representation, then we see that

$$\begin{aligned} R(t) &= W^{-1} \star D(t) \\ &= \int_{-\infty}^{\infty} W^{-1}(t_1) D(t-t_1) dt_1 \\ &= \frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{D(\omega)}{W(\omega)} e^{-i\omega t} d\omega. \end{aligned}$$

Thus, we see that **deconvolution is division in the frequency domain**.

11.5.4 Deconvolution of functions represented by their Z-transforms

In terms of Z-transforms, we would then be dividing the polynomial representation of the signal by the polynomial representation of the wavelet. The zeros of the Z-transform polynomial become poles in the deconvolution result, which, if were were performing the inversion by contour integration would be the contribution to the contour integral.

11.5.5 Division in the frequency domain - Deterministic deconvolution

There is a problem, however, when we consider the Fourier transform as a spectrum. If the function is zero over a range of values in the frequency domain (as opposed to isolated zeros in the Z-transform) in the Fourier transform form of the wavelet given by the function $w(\omega)$, then deconvolution is unstable or undefined. This follows because division by a small number introduces computational instability, and of course, division by zero is not defined.

We recall that $w(\omega)$ is a complex valued function, which may be written in complex exponential form as

$$w(\omega) = |w(\omega)|e^{i\omega\phi(\omega)}$$

or as the sum of real and imaginary parts as

$$w(\omega) = w_r(\omega) + iw_i(\omega).$$

We define the *complex conjugate* of $w(\omega)$ as

$$\bar{w}(\omega) = |w(\omega)|e^{-i\omega\phi(\omega)}$$

or as the sum of real and imaginary parts as

$$\bar{w}(\omega) = w_r(\omega) - iw_i(\omega).$$

If we multiply $w(\omega)$ by its complex conjugate, we have the square of the modulus of $w(\omega)$ as above

$$|w(\omega)|^2 = w(\omega)\bar{w}(\omega).$$

Returning to our deconvolution problem, multiplying top and bottom of the integrand by $\bar{w}(\omega)$, we have

$$\begin{aligned} R(t) &= W^{-1} \star D(t) \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty} \frac{\bar{w}(\omega)d(\omega)}{|w(\omega)|^2} e^{-i\omega t} d\omega. \end{aligned}$$

We still haven't solved the problems of division by zero in $w(\omega)$ because if $w(\omega)$ has a zero, then so will $|w(\omega)|$. We solve this problem by adding a small number ε to the denominator

$$\begin{aligned} R(t) &= W^{-1} \star D(t) \\ &= \frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{\bar{w}(\omega) d(\omega)}{(|w(\omega)|^2 + \varepsilon)} e^{-i\omega t} d\omega. \end{aligned}$$

The quantity ε is the *noise* or *whitening* or *white noise* parameter. This parameter is chosen to be small enough to stabilize the inverse, but not so big as to skew the results, and, as such is scaled by the maximum of the observed autocorrelation. Thus, formally we can define the inverse waveform $W^{-1}(t)$ by its Fourier transform representation

$$W^{-1}(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{\bar{w}(\omega)}{(|w(\omega)|^2 + \varepsilon)} e^{-i\omega t} d\omega.$$

It is important to remember that no matter how a deconvolutional process is performed, we think of deconvolution as division in the frequency domain. All deconvolution schemes must then have the equivalent of a white noise parameter to stabilize the division process, by preventing division by a small number or by zero.

Deterministic deconvolution in SU - sucddecon

In SU the program **sucddecon** performs deconvolution by a direct division in the frequency domain, given an input waveform as the **sufile=filename**.

```
$ sucddecon


 SUCDDECON - DECONvolution with user-supplied filter by straightforward
              Complex Division in the frequency domain

 sucddecon <stdin >stdout [optional parameters]

 Required parameters:
 filter=                  ascii filter values separated by commas
                 ...or...
 sufile=                 file containing SU traces to use as filter
                        (must have same number of traces as input data
                           for panel=1)
 Optional parameters:
 panel=0                 use only the first trace of sufile as filter
                          =1 decon trace by trace an entire gather
```

```
 pnoise=0.001                white noise factor for stabilizing results
                                        (see below)
 sym=0                not centered, =1 center the output on each trace
 verbose=0                silent, =1 chatty
...
```

For example, if we use the far field air gun signature in Fig 11.1 as our input waveform we can apply **succdecon** to deconvolve a panel of our data with this waveform. Here we deconvolve **cdp 265**

```
$ sucddecon sufile=farfield_gun.su < gain.jon=1.cdp=265.su |
                    suxwigb title="deterministic deconvolution" key=offset perc=99 &
```

which looks rather bad. We can see that there have been frequencies manufactured by the filtering process, so applying a bandpass filter

```
$ sucddecon sufile=farfield_gun.su < gain.jon=1.cdp=265.su |
          sufilter f=5,10,70,80 |
                    suxwigb title="deterministic deconvolution" key=offset &
```

we obtain a more acceptable result.

Note also, that the value of the **pnoise=** parameter can make a big difference. In fact the default value is low. Don't be afraid to try numbers like **pnoise=1** or **pnoise=10**.

```
$ sucddecon sufile=farfield_gun.su pnoise=10 < gain.jon=1.cdp=265.su |
          sufilter f=5,10,70,80 |
                    suxwigb title="deterministic deconvolution" key=offset &
```

11.5.6 Signature deconvolution using homomorphic wavelet estimation

If we had a far field gun signature for each shot, or some other estimate of the wavelet, then we could apply deterministic deconvolution to each shot. Such a shot by shot deconvolution is called"signature deconvolution."

There are a number of ways we could proceed to estimate the wavelet. Each method is based on the following assumptions:

- the waveform is minimum phase
- the wavelet is the only part of the data that does not change
- the reflectivity series is random.

An airgun signature is made to approximate a minimum phase (delay) waveform, so for ocean data with an airgun source this may not be a bad assumption. Even if we had a farfield airgun signature for each shot, this would not take into account the loss of higher frequencies due to anelastic attenuation.

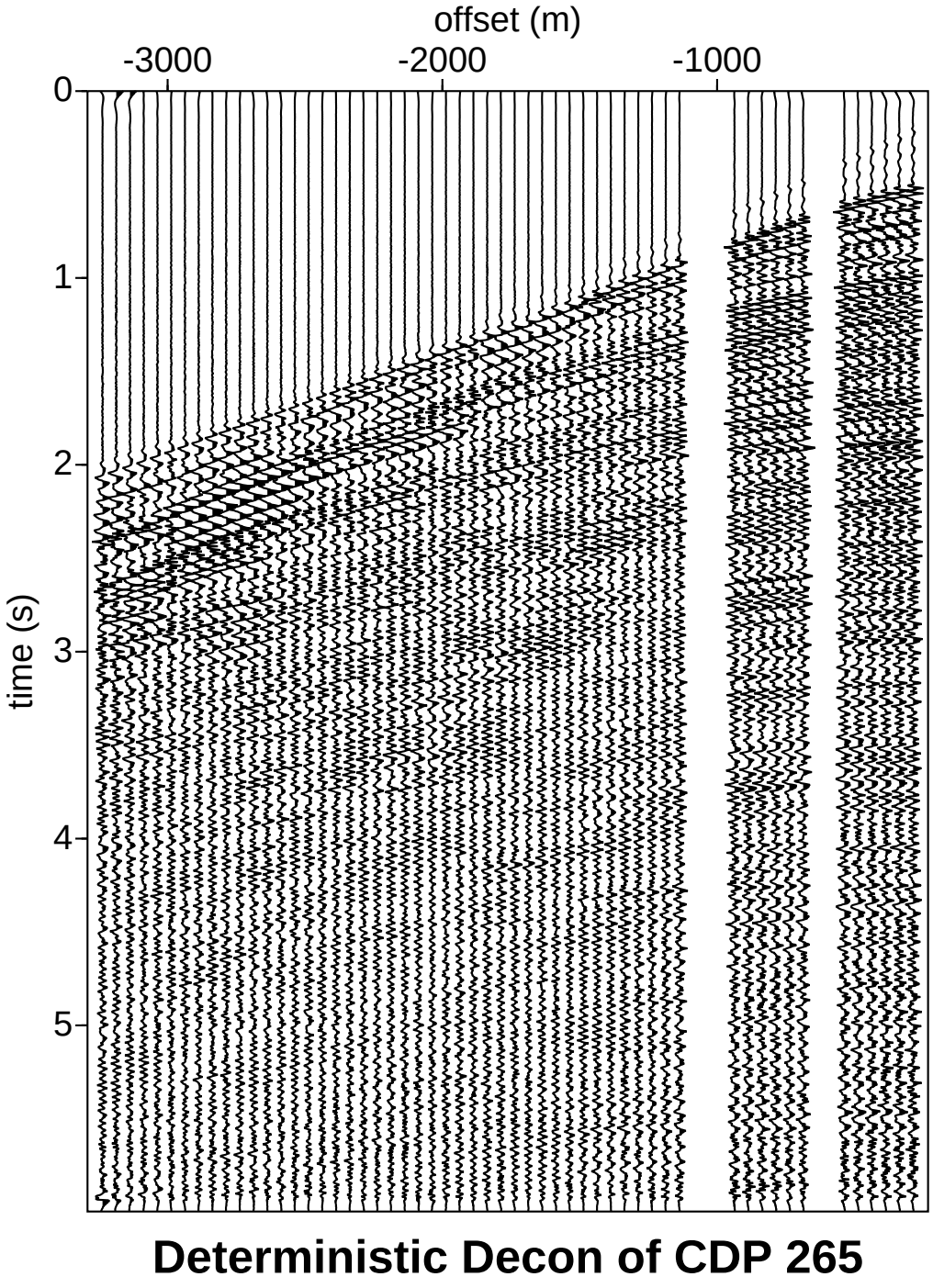


Figure 11.4: Deterministic decon of CDP 265 using the farfield airgun signature estimate from Fig 11.1

11.6 Cross- and auto-correlation

A related mathematical operation to convolution is the *cross correlation*. **The cross-correlation of two functions is the multiplication of one function by the complex conjugate of the other in the frequency domain**. Here we represent the cross-correlation by the symbol "xcor," which for digital data is the serial multiplication of the discrete representations of $A(t)$ and $B(t)$. We write this as in the Fourier domain as

$$\begin{aligned} A(t) \text{ xcor } B(t) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} a(\omega)\bar{b}(\omega)e^{-i\omega t}d\omega. \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty} \bar{a}(\omega)b(\omega)e^{-i\omega t}d\omega. \end{aligned}$$

We can see that the *auto-correlation* is the **product of a function with its own complex conjugate in the frequency domain**.

$$A(t) \text{ xcor } A(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} a(\omega)\bar{a}(\omega)e^{-i\omega t}d\omega.$$

Thus the frequency domain representation of the autocorrelation of our waveform is given by the $|w(\omega)|^2$, which appears the denominator of the frequency domain form of the deconvolution, and in the Fourier transform representation of inverse wavelet $W(t)$.

Whether deconvolution is performed in the time-domain, or in the frequency domain, the common elements of the **auto-correlation**, the ε **noise or whitening** parameter, and the wavelet $W(t)$ are present.

11.6.1 Z-transform view of cross-correlation

Given the Z-transform representations of two signals $B(Z)$ and $A(Z)$

$$A(Z) = \sum_{k=1}^{N} a_k Z^k \qquad \text{and} \qquad B(Z) = \sum_{l=1}^{N} b_l Z^l$$

we represent the complex conjugate $\bar{B}(Z)$ as the same series, but with terms represented by the negative powers of Z

$$\bar{B}(Z) = \sum_{l=1}^{N} b_l Z^{-l}.$$

The cross-correlation of $A(Z)$ and $B(Z)$ is then the product of the polynomials

$$A(Z)\bar{B}(Z) = \left(\sum_{k=1}^{N} a_k Z^k\right)\left(\sum_{l=1}^{N} b_l Z^{-l}\right).$$

The effect is to flip the order of the $B(z)$ series in the multiplication to the opposite order as would be done with convolution.

11.6.2 Cross correlation and auto correlation in SU suxcor and suacor

```
$ suxcor

SUXCOR - correlation with user-supplied filter

 suxcor <stdin >stdout  filter= [optional parameters]

 Required parameters: ONE of
 sufile=                file containing SU traces to use as filter
 filter=                user-supplied correlation filter (ascii)

 Optional parameters:
 vibroseis=0               =nsout for correlating vibroseis data
 first=1                supplied trace is default first element of
                         correlation.  =0 for it to be second.
 panel=0                use only the first trace of sufile as filter
                         =1 xcor trace by trace an entire gather
 ftwin=0                first sample on the first trace of the window
                                 (only with panel=1)
 ltwin=0                first sample on the last trace of the window
                                 (only with panel=1)
 ntwin=nt                number of samples in the correlation window
                                 (only with panel=1)
 ntrc=48                number of traces on a gather

...

$ suacor

SUACOR - auto-correlation

 suacor <stdin >stdout [optional parms]

 Optional Parameters:
 ntout=101      odd number of time samples output
 norm=1 if non-zero, normalize maximum absolute output to 1
 sym=1          if non-zero, produce a symmetric output from
                        lag -(ntout-1)/2 to lag +(ntout-1)/2
```

11.7 Lab activity #20: Wiener (least-squares) filtering

There is a class of deconvolutional processes known as *Wiener filters* or *prediction error filters*, which have been found to be useful in exploration seismic methods. The method is called "predictive" because it assumes that the data have a specific character that allow later parts of the data to be predicted from earlier parts of the data.

Wiener filtering assumes that the data are *minimum phase (aka minimum delay).* While there is a requirement that the spectrum of the data is white, a small "noise" parameter is added or assumed in the algorithm to prevent division by zero. Physically, if a waveform is minimum phase (aka minimum delay) its energy is located in the front part of the waveform.

11.7.1 A matrix view of the convolution model

The convolutional model of seismic waves holds that the **data** $D(t)$ are formed by the convolution of a **wavelet** $W(t)$ with a **reflectivity series** $R(t)$. Symbolically this is represented as

$$D(t) = W(t) \star R(t)\,, \tag{11.7.1}$$

where

$$R(t) = \sum_{k=0}^{N} \mathcal{R}_k \delta(t - \tau_k).$$

Here $\delta(t - \tau_k)$ is the Dirac delta function, which turns on only at time τ_k, the two way traveltime to the k-th reflector, and $\mathcal{R}_k$ is the reflection coefficent (either positive or negative) of the k-th reflector.

As an integral, the convolution of the reflectivity series $R(t)$ with the wavelet $W(t)$ is defined as

$$D(t) = W(t) \star R(t) = \int_{-\infty}^{\infty} W(t - \tau) R(\tau) d\tau.$$

The discrete version of this operation can be written as

$$D_n = \sum_{k=-N}^{N} W_{n-k} R_k.$$

Note how the integration variables in the continuous version correspond to the indexes in the discrete version.

We can write this numerically as the matrix multiplication

$$\mathbf{W}R = D \tag{11.7.2}$$

where $\mathbf{W}$ is a band matrix, whose rows are composed of shifted versions of a discrete representation of the wavelet $W(t) = \{w_0, w_1, w_2, ...\}$, the reflectivity series $R(t) = \{r_0, r_1, ...\}$,

and the data $D(t) = \{d_0, d_1, ...\}$.

$$\begin{bmatrix} w_0 & w_1 & w_2 & ... & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & w_0 & w_1 & w_2 & ... & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_0 & w_1 & w_2 & ... & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_0 & w_1 & w_2 & ... & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & ... & ... & ... & ... & 0 & 0 \\ 0 & 0 & 0 & 0 & ... & ... & ... & ... & 0 & 0 \\ ... & ... & ... & ... & 0 & w_0 & w_1 & w_2 & ... & 0 \\ ... & ... & ... & ... & ... & 0 & w_0 & w_1 & w_2 & ... \\ ... & ... & ... & ... & ... & ... & 0 & w_0 & w_1 & w_2 \end{bmatrix} \begin{bmatrix} r_0 \\ r_1 \\ r_3 \\ ... \\ ... \\ ... \\ ... \\ ... \\ ... \\ ... \\ ... \\ ... \\ r_n \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ d_3 \\ ... \\ ... \\ ... \\ ... \\ ... \\ d_m \end{bmatrix}. \tag{11.7.3}$$

We want to solve for R, but W is, in general, a non-square $(m \times n)$ matrix. Our solution is the *pseudoinverse* or *least-squares* solution. We multiply by the transpose of $\mathbf{W}$, which we write as $\mathbf{W^T}$, we obtain

$$\mathbf{W}^T\mathbf{W}R = \mathbf{W}^T D \tag{11.7.4}$$

where ideally the solution for D is given by taking the inverse of $\mathbf{W}^T\mathbf{W}$, yielding

$$R = (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T D. \tag{11.7.5}$$

For stability εI is added to the $\mathbf{W}^T\mathbf{W}$ to yield the final form

$$R = (\mathbf{W}^T\mathbf{W} + \varepsilon I)^{-1}\mathbf{W}^T D. \tag{11.7.6}$$

As written, this expression describes a mathematician's view of the problem, but this form is not really practical to implement.

If we go back a step

$$\mathbf{W}^T\mathbf{W}R = \mathbf{W}^T D \tag{11.7.7}$$

The right hand side of the contains $\mathbf{W^T}D$ which is the cross-correlation of the wavelet W with the recorded data D. This is a spiked version of the data. The quantity W^TW is the matrix of autocorrelations of the wavelet. The quantity $\mathbf{W}^T D$ of bandlimited spikes, whose heights are proportional to the reflection coefficients. If the original data have no multiples, then the cross-correlation of the wavelet with the data will be a band-limited "spiked" version of the reflectivity series and would be a form of "spiking deconvolution". Again, we are less satisfied with this because we do not have the wavelet W needed so we can compute $\mathbf{W^T}D$.

The left hand side of the expression contains $\mathbf{W^T W}$, which is a matrix whose rows are composed of the autocorrelation of the wavelet, shifted successively by one sample

on each row

$$W^T W = \begin{bmatrix} \phi_0 & \phi_1 & \phi_2 & ... & \phi_{n-1} \\ \phi_{-1} & \phi_0 & \phi_1 & ... & \phi_{n-2} \\ \phi_{-2} & \phi_{-1} & \phi_0 & ... & \phi_{n-3} \\ ... & ... & ... & ... & ... \\ \phi_{n-1} & ... & ... & \phi_{-1} & \phi_0 \end{bmatrix}. \tag{11.7.8}$$

Here, $\phi_{-(n-1)}, .., \phi_{-1}, \phi_0, \phi_1, ..., \phi_{n-1}$ is the autocorrelation of the wavelet $W(t)$. We note that this is symmetric such that $\phi_{-k} = \phi_k$. If the reflectivity series is random, such that the later values of of $R(t)$ cannot be predicted from earlier values, then the autocorrelation of the data $D(t)$ is approximately the same as the autocorrelation of the wavelet $W(t)$.

At most, these discussions tell us about the problem of deconvolution, without giving us a practical solution to implement. We must look further.

11.7.2 Designing wavelet shaping filters – Wiener filtering

If the reflectivity series R is random, then the autocorrelation of the recorded data D is a good approximation to the autocorrelation of the wavelet W, because at most the autocorrelation of a random sequence is a constant value, as matricies $R^T R$ is the identity matrix I).

We don't want the entire autocorrelation, only the *autocorrelation waveform*, that is, the sinc-function like part that is in the middle of the autocorrelation output. We will call this waveform $\Phi(t)$.

Suppose that we want to create a filter that will take data with given input wavelet $W(t)$, and yield a desired output wavelet $V(t)$. Then we want to design a filter $F(t)$ such that the application of the filter to $W(t)$ yields $V(t)$,

$$W(t) \star F(t) = V(t). \tag{11.7.9}$$

Writing this convolution in the discrete representation, we have

$$\sum_{k=0}^{n-1} w_{m-k} f_k = v_m. \tag{11.7.10}$$

Likely we cannot solve this problem exactly, so there will be an error vector E obtained by subtracting the right hand side from the left hand side

$$E_m = \left(\sum_{k=0}^{n-1} w_{m-k} f_k - v_m \right). \tag{11.7.11}$$

11.7.3 Least-squares (Wiener) filter design

An approach developed independently by N. Wiener and A. N. Kolmogorov in the early 1940s is to apply least-squares optimization to the problem of designing the wavelet shaping filter. The filters that result from this approach are often called *Wiener filters*. Because noise is always present we cannot solve the system exactly.

The square of the error is given by

$$E^2 = \sum_{j=0}^{m} \left(\sum_{k=0}^{n-1} w_{j-k} f_k - v_j \right) \left(\sum_{k=0}^{n-1} w_{j-k} f_k - v_j \right). \tag{11.7.12}$$

The extra summation is required because the error is a vector, but the square of the error is taken as the dot product of the error vector with itself, which is a scalar. We much prefer dealing with a scalar quantity than a vector quantity in this context.

Minimization means **taking the derivative and setting the result to zero**. Which derivative? The thing that is varying is the filter, so we should be differentiating with respect to the filter values

$$\frac{\partial}{\partial f_p} E^2 = \frac{\partial}{\partial f_p} \sum_{j=0}^{m} \left(\sum_{k=0}^{n-1} w_{j-k} f_k - v_j \right) \left(\sum_{k=0}^{n-1} w_{j-k} f_k - v_j \right). \tag{11.7.13}$$

Applying the chain rule, and setting $E^2 = 0$

$$0 = \sum_{j=0}^{m} \left(2 \sum_{k=0}^{n-1} w_{j-k} \frac{\partial f_k}{\partial f_p} (\sum_{k=0}^{n-1} w_{m-k} f_k - v_m) \right). \tag{11.7.14}$$

We note that $\partial f_k / \partial f_p = 1$ when $k = p$ and is zero when $k \neq p$, hence we have (canceling the factor of 2)

$$0 = \sum_{j=0}^{m} w_{j-p} (\sum_{k=0}^{n-1} w_{m-k} f_k - v_m). \tag{11.7.15}$$

which may be rewritten as

$$\sum_{j=0}^{m} w_{j-p} \sum_{k=0}^{n-1} w_{m-k} f_k = \sum_{j=0}^{m} w_{j-p} v_m. \tag{11.7.16}$$

In matrix notation this is

$$\mathbf{W}^T \mathbf{W} F = \mathbf{W}^T V. \tag{11.7.17}$$

The matrix $\mathbf{W}^T\mathbf{W}$ is the matrix of autocorrelations of the wavelet, as before, and the right hand side is the crosscorrelation of the wavelet, with the desired output $V(t)$. The matrix of autocorrelations is in a form called a *Toeplitz* matrix. There is a recursive method of solution of Toeplitz systems pioneered by N. Levinson in 1947, and improved by Durbin in 1960, and others since that time. The recursive method allows for the filter F to be solved for directly. This makes what might seem like a difficult process rather simple.

Several programs in the SU package use this method for performing *spiking deconvolution*, *predictive or gapped-deconvolution*, and *wavelet shaping* using these facts.

11.8 Spiking deconvolution

Our approach must be something different. The approach that we will use will be to design a filter that will modify the autocorrelation. **The plan of attack will be to model the part of the autocorrelation that we do not want, and to subtract that part from the data.** If we actually have a minimum delay wavelet, this filter will modify the data to eliminate that part of the data that corresponds to that part of the autocorrelation we do not want.

Suppose that we want to do spiking deconvolution. If the data were already spiked, then the autocorrelation would be a delta function at sample number zero, and the autocorrelation vector would consist of a single nonzero value $(\phi_0, 0, 0, ...)$, meaning that the autocorrelation matrix would be a diagonal matrix, with ϕ_0 repeated down the diagonal. The value of ϕ_0 might be normalized to 1, or not. Thus we want to annihilate every point in the autocorrelation vector except for the first point.

Our issue is to solve the problem

$$\begin{bmatrix} \phi_0 & \phi_1 & \phi_2 & ... & \phi_{n-2} \\ \phi_{-1} & \phi_0 & \phi_1 & ... & \phi_{n-3} \\ \phi_{-2} & \phi_{-1} & \phi_0 & ... & \phi_{n-4} \\ ... & ... & ... & ... & ... \\ \phi_{n-2} & ... & ... & \phi_{-1} & \phi_0 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ ... \\ ... \\ f_{n-2} \end{bmatrix} = \begin{bmatrix} \phi_1 \\ \phi_2 \\ ... \\ ... \\ \phi_{n-1} \end{bmatrix}. \tag{11.8.1}$$

Notice that the vector on the right beings with ϕ_1 not ϕ_0.

Thus we want to solve the Toeplitz system for the filter $f_0, f_1, ..., f_{n-2}$. This allows us to compute the values of the autocorrelation after the first value. We then want to subract this from our data.

Actually filtering and then subtracting is unnecessary. We can build the subtraction into the filter by writing $1 - F = (1, -f_0, -f_1, ..., -f_{n-1})$. The only thing that we need to know is how many points long the filter should be. This is the *maximum lag* of the spiking filter. **We take the length of the autocorrelation waveform as the value of the maximum lag for spiking deconvolution**. The *minimum lag* we use is the default value of 1 sample.

The reader should be aware that this algorithm is somewhat insensitive to choice of the number of samples in the autocorrelation. We do not need to worry about picking an exact value, as there are a number of values that will work more or less equally well.

11.8.1 What does "lag" mean?

If a time delay is in seconds, or other time units, we call that difference in time simply a *delay*. If we express a time delay in samples, then it is the convention to call that a *lag*. In the SU programs that have "lags" those lags are expressed in seconds, which are much easier to find on a plot than to compute a sample number.

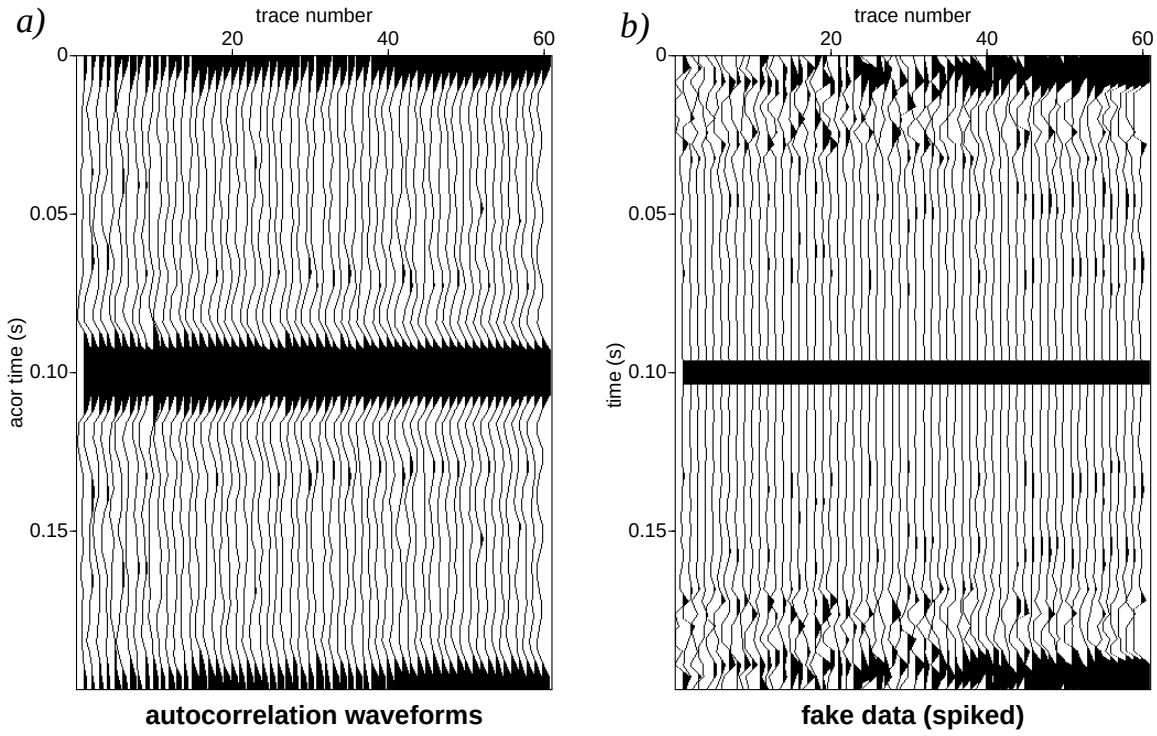


Figure 11.5: a) Autocorrelation waveforms of the **fake.su** data b) Autocorrelation waveforms of the same data after predictive (spiking) decon.

11.8.2 Spiking Deconvolution in SU

In SU the program **supef** may be used to perform spiking deconvolution

```
$ supef
SUPEF - Wiener predictive error filtering

 supef <stdin >stdout  [optional parameters]

 Required parameters:
 dt is mandatory if not set in header

 Optional parameters:
 cdp=                    CDPs for which minlag, maxlag, pnoise, mincorr,
                         maxcorr are set (see Notes)
 minlag=dt               first lag of prediction filter (sec)
 maxlag=last             lag default is (tmax-tmin)/20
 pnoise=0.001                 relative additive noise level
...
```

Note also, that the value of the **pnoise=** parameter can make a big difference in the output.

If our data consist of a wavelet W convolved with the reflectivity series R, further convolved with multiples M, then our model in the previous section is not quite right. The autocorrelation will contain repetitions due to the multiples.

If the wavelet is minimum phase (aka minimum delay), then most of the energy will be located at the beginning of the waveform, and the autocorrelation of the data will produce a sinc-like autocorrelation waveform that is localized to the values near the center of the output of the autocorrelation. This is the diagonal region of the autocorrelation represented as a matrix.

If we select this window in the autocorrelation for processing, then spiking decon filter will be generated approximately correctly.

In **supef** the value of $maxlag =$ is set to the width of the autocorrelation waveform, which we must determine by taking the autocorrelation using **suacor**. (Remember that a "lag" is just a time delay.)

For example, consider the **fake.su** data

```
$ suacor < fake.su  ntout=101  | suxwigb perc=90.
```

Another possibility is to stack the autocorrelations

```
$ suacor < fake.su  ntout=101
                | sustack key=dt | suxgraph  style=normal f1=-.2
```

which yields a sinc-like waveform.

The choice of **key=dt** was to ensure that the traces were all stacked with respect to a header field that does not change in the gather and the choice of **f1=-.2** is so that

the peak of the autocorrelation is at zero lag. The choice of **ntout=101** means that we want 101 samples on the resulting traces. This number is chosen to be sufficiently large to capture the side lobes of the wavelet that appears in the center of each of the resulting traces.

This waveform is the *autocorrelation waveform*. For data that are dominated by spikes, or are spectrally white, the autocorrelation waveform would also be a spike.

We pick the width of the autocorrelation waveform. In the case of our example, this is between approximately 0.0667 and 0.1340 seconds, making the width of the autocorrelation waveform approximately .0673 seconds. We apply **supef** setting this value as the value of **maxlag**

```
$ supef maxlag=.0673 < fake.su | suxwigb perc=99
```

The data are made more spike-like by the operation. Try different values of **maxlag=** to see what the effect of changing this parameter is.

Also you might want to view autocorrelation waveform of the deconvolved data to see what happens

```
$ supef < fake.su  maxlag=.0673
            | suacor ntout=101 | suxwigb perc=90.
or
$ supef < fake.su  maxlag=.0673
            | suacor ntout=101 | sustack key=dt | suxgraph style=normal
```

The effect

We apply the same operations on CDP 265

```
$ suacor ntout=101 < gain.jon=1.cdp=265.su
            | suxwigb perc=99
```

The autocorrelation waveform is a sinc-like function. We define the "width" of this waveform to be the window of time just large enough to include the side lobes on each side of the main lobe. If we measure the time from the beginning to the end of the autocorrelation waveform, which is to say about .169 seconds to about .247 seconds see that the width is about .078 seconds. Your values may differ.

This is the value of **maxlag** that we will set in **supef**

```
$ supef < gain.jon=1.cdp=265.su maxlag=.078 | suxwigb xcur=3
```

To see how well this has spiked the data, we may view the autocorrelation waveform with **suacor**

```
$ supef < gain.jon=1.cdp=265.su maxlag=.078
             | suacor ntout=101 | suxwigb perc=90
```

which should show that the autocorrelation waveform is now a spike. Again, we may vary the value of **maxlag=** to see the effect of changing this parameter.

11.8.3 Multiple suppression by Wiener filtering—Gapped prediction error filtering.

We now seek to eliminate multiples by *prediction error filtering* also known as *predictive deconvolution*. Predictive decon relies on the minimum phase (aka minimum delay) assumption and the notion that the data contain repetitions owing to the series of multiples M.

One of the reasons that multiples are damaging to processing and hard to eliminate is that multiples are not merely added to the data, they are *convolved* with the reflectivity series R, which is in turn convolved with the wavelet W

$$\begin{aligned} D &= W \star M \star R \\ D_n &= \sum_{m=0}^{N} M_{n-m} \sum_{k=0}^{K} W_{l-k} R_k, \end{aligned} \tag{11.8.2}$$

which we could write as a cascaded matrix multiplications on to the vector R

$$D = \boldsymbol{M}\boldsymbol{W}R \tag{11.8.3}$$

where both $\boldsymbol{M}$ and $\boldsymbol{W}$ matricies composed of shifted versions of the M and W series, respectively. Not also, that there is a noise vector N that is added, as well

$$D = W \star M \star R + N \tag{11.8.4}$$

which we have been quietly ignoring.

If we form the autocorrelation of the data D, the result will be the same as the autocorrelation of the series of reverberations M convolved with the wavelet W. The reflectivity series R is considered to be random because later values of reflectivity may not, in general, be predicted from earlier values. (This ignores the possibility of transgressive and regressive sequences in geology, which may be repetetive.) We may also consider the noise N to be composed of the sum of a random and non-random part.

The autocorrelation of the reflectivity series will be assumed to not contribute to the autocorrelation of the data. We call the matrix of shifted autocorrelations Φ which will be approximately

$$\begin{aligned} \Phi &= (\boldsymbol{M}\boldsymbol{W}R)^T \boldsymbol{M}\boldsymbol{W}R \\ \Phi &= R^T \boldsymbol{W}^T \boldsymbol{M}^T \boldsymbol{M}\boldsymbol{W}R \\ \Phi &\approx \boldsymbol{W}^T \boldsymbol{M}^T \boldsymbol{M}\boldsymbol{W} \end{aligned} \tag{11.8.5}$$

where

$$\Phi = \begin{bmatrix} \phi_0 & \phi_1 & \phi_2 & \cdots & \phi_{n-1} \\ \phi_{-1} & \phi_0 & \phi_1 & \cdots & \phi_{n-2} \\ \phi_{-2} & \phi_{-1} & \phi_0 & \cdots & \phi_{n-3} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \phi_{-(n-1)} & \cdots & \cdots & \phi_{-1} & \phi_0 \end{bmatrix}. \tag{11.8.6}$$

Here each row is a shifted version of the autocorrelation of the full data D which is the symmetric waveform $\phi_{-(n-1)}, \phi_{-(n-2)}, \phi_{-(n-3)}, ..., \phi_{-2}, \phi_{-1}, \phi_0, \phi_1, \phi_2, ..., \phi_{(n-3)}, \phi_{(n-2)}, \phi_{(n-1)}$.

The character of the autocorrelation will be as follows. If the wavelet W is minimum phase (aka minimum delay), the portion in the middle about the value ϕ_0 will be a sinc-like waveform consisting of a main lobe centered at ϕ_0, symmetric with just a couple of side lobes, allowing us to estimate the autocorrelation of the wavelet W—hence our use of the width of the autocorrelation waveform for **maxlag=** in **supef** for spiking deconvolution.

The autocorrelation will contain repetitions owing to the autocorrelation of the multiples M with the reflectivity series. That repetition time will be related to the two-way traveltime in the water column. If we apply spiking decon to our data, the autocorrelation waveform will be a spike, and what remains will be repeating spikes.

Our autocorrelation will start repeating at some sample k, so we are going to predict the repetitions in the autocorrelation (which correspond to reverberations in our data) and then subtract them off. We must find F given

$$\begin{bmatrix} \phi_0 & \phi_1 & \phi_2 & ... & \phi_{n-1} \\ \phi_{-1} & \phi_0 & \phi_1 & ... & \phi_{n-2} \\ \phi_{-2} & \phi_{-1} & \phi_0 & ... & \phi_{n-3} \\ ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... \\ \phi_{-(n-1)} & ... & ... & \phi_{-1} & \phi_0 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_3 \\ ... \\ ... \\ ... \\ ... \\ f_{n-1} \end{bmatrix} = \begin{bmatrix} \phi_k \\ \phi_{k+1} \\ \phi_{m+2} \\ \phi_{m+3} \\ ... \\ ... \\ ... \\ \phi_{(k+n-1)} \end{bmatrix}. \tag{11.8.7}$$

Thus, we are finding a least-squares filter F that will predict the repetitions in the autocorrelation by using the earlier values of the autocorrelation. Hence the notion of "predictive" decon. The repetitions start at sample k, so this is the delay of the filter. We don't want the repetitions, so the filter we apply to the data is formed by subtracting our filter (delayed by k samples).

In our notation, the prediction error filter is given by $1-F = \{1, 0, 0, ..., 0, -f_0, -f_1, ..., -f_{\max}\}$. Here there are $k-1$ zeros, which is the "gap" in the gapped decon. The "prediction error" is the difference between the data and the predicted value from the Weiner filter.

11.8.4 Applying gapped decon in SU – supef

By selecting the appropriate combination of **minlag=** and **maxlag=** defining the Wiener filter, we can eliminate repetitions in the data, such as those caused by multiples. This is known as *gapped predictive decon* in the paralance of the geophysical community.

We begin by spiking our **fake+water+pegleg.su** which are our data with water-bottom and pegleg multiples

```
$ supef < fake+water+pegleg.su maxlag=.0673 |  suxwigb perc=99 xcur=2
```

We then view the autocorrelation of the data in a broader window choosing **ntout=1024** samples in the output. The idea is to look for repetitions in the autocorrelation

```
$ supef < fake+water+pegleg.su  maxlag=.0673 |
          suacor ntout=1024 |  suxwigb perc=90
```

What we are looking for are repetitions in the autocorrelation. We know that the two-way traveltime for the water speed is about .5 s, and we see stripes that are at about .51 s above and below the autocorrelation waveform spike. Also we notice that there is an offset effect. Thus, we apply a moveout correction to flatten the data

```
$ sunmo vnmo=1500 smute=20 < fake+water+pegleg.su  | supef maxlag=.0673 |
          suacor ntout=1024 |  suxwigb perc=90
```

where we have used *smute* = 20 in **sunmo** to turn off the stretch mute. Notice that the result is sensitive to the value of *vnmo*. It might be that making *vnmo* = slightly bigger gives a slightly flatter collection of spikes

```
$ sunmo vnmo=1800 smute=20 < fake+water+pegleg.su  | supef maxlag=.0673 |
          suacor ntout=1024 |  suxwigb perc=90
```

The repetition time of the signal is the value that is needed to define the "gap" in the gapped decon. In this case the gap is .51 seconds. This value is our choice for **minlag**. The **maxlag** will be the value of **maxlag** we used for spiking the data added to the value of the gap, **maxlag = maxlag for spiking + gap**.

Finally, we finish by doing inverse NMO

```
$ sunmo vnmo=1800 smute=20 < fake+water+pegleg.su |
       supef maxlag=.0673 |
           supef minlag=.51 maxlag=.5773 |
               sunmo invert=1 vnmo=1800 smute=20 |  suxwigb perc=99 xcur=2
```

Here we have performed **spiking decon** and have followed this with a **gapped decon**. It may be better to do the gapped decon only, which would be done via

```
$ sunmo vnmo=1800 smute=20 < fake+water+pegleg.su |
          supef minlag=.51 maxlag=.5773 |
              sunmo invert=1 vnmo=1800 smute=20 |  suxwigb perc=99 xcur=2
```

where we note that the **maxlag=.51 + .0673** is chosen as if we had performed spiking decon. Again we have used *smute* = 20 to turn off the stretch mute in **sunmo**. **The value of minlag= must not exceed the value of the actual reverberation time, but it may be less.** The value of **maxlag=** again is not so sensitive.

Again the value of **pnoise=** may be adjusted to improve the result.

We may view effect on the multiples by comparing semblance panels

```
$ suvelan < fake+water+pegleg.su nv=150 dv=15 fv=1450 |
              suximage d2=15 f2=1450 cmap=hsv2 bclip=.5 title="cdp 265" &
$ suvelan < pef.fake+water+pegleg.su nv=150 dv=15 fv=1450 |
              suximage d2=15 f2=1450 cmap=hsv2 bclip=.3 title="PEF" &
```

The multiples with speeds near the water speed have been suppressed, as have some of the multiples from the strong reflector near 2 sec and 2000 m/s. However, this is not as clean as the radon transform filtered data.

We may repeat the process to eliminate other repetitions in the data, such as those from pegleg multiples. As with radon domain filtering, we choose appropriate **nmo** velocities to flatten the arrivals we choose to remove. You may want to try repeating the last several steps using the data **fake+water+pegleg.su**.

11.9 What (else) did predictive decon do to our data?

The fact that we are applying an inverse filter to our data means that in some sense we are making the output look "more like a bunch of spikes" or "more like a bunch of Dirac delta functions". Because we know that a spike contains all frequencies, the term *spectral whitening* is applied to describe the effect of such filters in the frequency domain. This bug/feature may be observed in your data by comparing the amplitude spectra

```
$ suspecfx < fake+water.su | suximage title="data before spiking decon"
$ suspecfx < pef.fake+water.su | suximage title="data after spiking decon"
```

On one hand, it may seem that the increased frequency content is a good thing. However, can we really trust that those frequencies have been correctly added to the data? These may be simply an artifact of the filter that causes more harm than good. Some spectral whitening is desirable, but most should probably be suppressed by filtering. For example we might consider simply applying a filter to the data as part of the processing

```
$ .... | sufilter f=0,2,60,70 | ...
```

where the values of the corner frequencies of the filter are chosen to reflect a reasonable range of frequencies in the data that can be trusted. So finally, the processing sequence for our fake data with waterbottom multiples is

```
$ sunmo vnmo=1800 smute=20 < fake+water+pegleg.su |
      supef maxlag=.0673 |
        supef minlag=.51 maxlag=.5773 |
            sunmo invert=1 vnmo=1800 smute=20 |
              sufilter f=0,2,60,70 | suxwigb perc=99 xcur=2
```

or, if we seek to do multiple-suppression only, without spiking decon

```
$ sunmo vnmo=1800 smute=20 < fake+water+pegleg.su |
        supef minlag=.51 maxlag=.5773 |
            sunmo invert=1 vnmo=1800 smute=20 |
              sufilter f=0,2,60,70 | suxwigb perc=99 xcur=2
```

For the real data, some variation on this processing flow, in terms of the values of **minlag=** and **maxlag=** will exist. Indeed, these values are guaranteed to vary some across the survey.

11.9.1 Deconvolution in the Radon domain

Another possibility is to apply the prediction error filtering in the radon domain. For example, employing the linear $\tau - p$ transform we forward radon transform the data, apply the prediction error filtering

```
sunmo vnmo=1500 smute=20 < gain.jon=1.cdp=265.su |
   suradon choose=0 igopt=3 pmin=-1500
        pmax=1000 interoff=-262 offref=-3237 |
         supef minlag=.15 maxlag=1.0 |
   suradon choose=4 igopt=3 pmin=-1500 pmax=1000
         interoff=-262 offref=-3237 |
   sunmo vnmo=1500 invert=1 smute=20 > radonpef.su
```

The process will do a good job on simple water-bottom reverberations, but other multiples will not be as well suppressed, unless these can be made exactly periodic in the radon domain.

11.10 FX Decon

There is application of prediction error filtering in the frequency domain, called "fx decon" that was created in the 1984 by L.L. Canales. This technique uses predictive decon in the space-frequency domain to identify and eliminate random noise.

For example, consider the spectrally whitened version of **fake.su**

```
$ suwfft w0=0 w1=1 w2=0 < fake.su | suifft  > white.fake.su
```

Applying **sufxdecon** to these data

```
$ sufxdecon < white.fake.su | suxwigb perc=99
```

Try this operation on different versions of CDP 265.

It may be best to reserve 'FX decon" for the later stages of processing, after the stack.

11.11 Lab Activity #20: Wavelet shaping

Many papers written in the 1970s dealt with the issue of *wavelet estimation*. That is, using statistical methods to determine the shape of the average wavelet throughout the dataset, or in regions in a dataset. The motivation for this is to use deconvolution to change the waveforms of the data to a new desired output waveform.

Currently in SU, there is no sophisticated wavelet estimation code as yet. The user can get a crude estimate of the wavelet by selecting the waveform from a horizontal portion of a reflector in the data. Knowing the trace number and the time window of the wavelet, we may use suwind to capture this "average wavelet" via:

```
suwind key=tracl min=TRACE max=TRACE tmin=TMIN tmax=TMAX > wavelet.su
```

where TRACE, TMIN, and TMAX are replaced with the actual values of the trace number, and minimum and maximum times that where the wavelet of choice is located.

Also, we can make a desired output waveform by using the program **suwaveform**

```
SUWAVEFORM - generate a seismic wavelet

suwaveform <stdin >stdout [optional parameters]

Required parameters:
   one of the optional parameters listed below

Optional parameters:
   type=akb       wavelet type
                  akb:     AKB wavelet defined by max frequency fpeak
                  berlage: Berlage wavelet
                  gauss:   Gaussian wavelet defined by frequency fpeak
                  gaussd:  Gaussian first derivative wavelet
                  ricker1: Ricker wavelet defined by frequency fpeak
                  ricker2: Ricker wavelet defined by half and period
                  spike:   spike wavelet, shifted by time tspike
                  unit:    unit wavelet, i.e. amplitude = 1 = const.

   dt=0.004       time sampling interval in seconds
   ns=            if set, number of samples in  output trace

   fpeak=20.0     peak frequency of a Berlage, Ricker, or Gaussian,
...
```

For example

```
$ suwaveform > dfile.su type=ricker1 fpeak=15
```

where **dfile.su** contains a Ricker wavelet with peak frequency **fpeak** of $15Hz$. The frequency content of the desired output waveform should approximately match the frequency content of the input wavelet.

Given the wavelet (**wavelet.su**) and the desired output waveform (**dfile.su**) we may use the wavelet shaping code, called **sushape**

```
SUSHAPE - Wiener shaping filter

 sushape <stdin >stdout  [optional parameters]

Required parameters:
w=             vector of input wavelet to be shaped or ...
...or ...
```

```
 wfile=        ... file containing input wavelet in SU (SEGY trace) format
 d=             vector of desired output wavelet or ...
 ...or ...
 dfile=        ... file containing desired output wavelet in SU format
 dt=tr.dt              if tr.dt is not set in header, then dt is mandatory

 Optional parameters:
 nshape=trace          length of shaping filter
 pnoise=0.001          relative additive noise level
 showshaper=0          =1 to show shaping filter

...
```

For example, our waveforms may be shaped via:

```
$ sushape dfile=dfile.su wfile=wavelet.su < data.su > shaped_data.su
```

The shaping filter works by effectively by performing the operation of deconvolving the data to remove **wavelet.su** and the convolution of the resulting "spiked" data by the desired output waveform **dfile.su**. The **sushape** program makes use of Wiener-Levinson theory to perform this operation in the time domain.

Finding the wavelet and making target waveforms

A great deal of work has been put into "wavelet estimation" techniques in the exploration seismic community. Ideally we should know the wavelet for each shot, and even the wavelet as a function of angle from the shot. Here, we assume for simplicity that waveforms chosen carefully off of the data, using **suwind** are sufficient for our purposes.

To construct the target waveform **dfile.su** we may use one of the wavelets generated from the program **suwaveform** with either **type=ricker1** or **type=ricker2** being the best choices, although the Berlage waveform is not a bad choice, either. When constructing a target waveform, make sure that the frequency content of the desired output waveform is roughly the same as that of the data, so that values not be "manufactured" by the program.

11.12 Filling in missing shots

We have noted that in the Viking Graben Dataset, there are a number of missing shots. These may be identified by viewing a "shooting chart" via:

```
$ suchart < seismic.su key1=sx key2=gx |
          xgraph n=120120 linewidth=0
                     label1="sx" label2="gx" marksize=2 mark=8 &
```

By zooming in on the plot, we can see that there are gaps in the data between shots located between **sx=** 5187 and 5262 meters, between 5387 and 5487 meters, between 14412 and 14512 meters, and between 22162 and 22262 meters. Because there is a 25 meter spacing between successive shot positions this means that shots at the following locations are missing: 5212, 5237, 5412, 5437, 5462, 14437, 14462, 14487, 22187, 22212, and 22237 meters.

Missing data is big issue in exploration seismology because holes in our data are a problem for processing algorithms, such as most migration routines, that expect that data are uniformly sampled and are "complete."

One method that can be employed is to simply replace missing shots with nearest neighbor shot gathers, or by the average of nearest neighbor shot gathers. For example we could capture and average two nearest neigbor shot gathers by windowing the data

```
...
# capture 5187 and 5262
suwind key=sx min=5187 max=5262 < seismic.su > junk1.su
...
```

and then by sorting into offsets, much as we did when we made a supershot gather in a previous section

```
...
# sort and stack into an average shot gather
susort dt offset < junk1.su > junk2.su
sustack key=offset < junk2.su > average5187_5262.su
...
```

We then make approximations to the missing shots by setting the trace headers to the values that are necessary so that these new shot gathers take the place of the missing shots

```
...
## make shot 5212
#
# set sx,ep,nhs header fields fields compute gx from sx and offset
sushw key=sx,ep,nhs a=5212,180,0 < average5187_5262.su
     | suchw key1=gx key2=sx  key3=offset a=0 b=1 c=1 > shot5212.su

## make shot 5237
# set sx,ep,nhs header fields fields compute gx from sx and offset
sushw key=sx,ep,nhs a=5237,181,0 < average5187_5262.su
       | suchw key1=gx key2=sx  key3=offset a=0 b=1 c=1 > shot5237.su
...
```

Similar commands would be applied for to create average shot gather approximations of the other missing shots.

Finally, the original data and the new shot gathers are concatenated together to produce a modified version of the shot gathers

```
...
# concatenate, sort to shot gathers, reset cdp field.
cat shot22237.su shot22212.su shot22187.su shot14487.su
         shot14462.su shot14437.su shot5462.su shot5437.su
            shot5412.su shot5237.su shot5212.su seismic.su > seismic1.su

susort < seismic1.su sx offset > seismic2.su
```

The file **seismic2.su** still has one additional change to be made. That is to set the **cdp** field of the headers.

We begin by setting the value of **cdp** to the midpoint value, multiplied by 10 as to not lose accuracy by round off error. We have to do this, because the **cdp** header field can be only integer valued

```
suchw key1=cdp key2=sx key3=gx b=10 c=10 d=2 < seismic2.su |
suchw key1=cdp key2=cdp key3=cdp a=-16060 b=1 c=0 |
suchw key1=cdp key2=cdp key3=cdp a=0 b=1 c=0 d=125  > seismic3.su
```

The second line subtracts off 16060 so that the first **cdp** header field value is 125, which is 10 times the midpoint spacing in meters. In the third line, we divide the values of **cdp** by 125, which, upon inspecting the headers, causes the first **cdp/** to have a value of 1, and the last value to be 2142, which is what the original data had. New trace count is 121440 traces.

Finally, the data can be sorted into cdp's

```
susort cdp offset < seismic3.su > seis_repaired.cdp.su
```

and all of the operations we have discussed so far can be applied to the new file **seis_repaired.cdp.su**.

11.13 Advanced gaining operations

Before gaining our data, we would like to remove the effect of the differing source strengths, and receiver gains on our data. These effects tend to cause vertical striping in our data. Indeed, this section should probably appear in the section on gaining, but as this requires some additional sorting of the data, we discuss the operation here.

We must use some estimate for source strength, but we also know that there are likely variabilities due to the receiver gains, so a statistical approach is used. Here, we apply RMS power balancing. The approach we use here is fast and simple, but it is not the only approach that may be applied.

Alternatively, if we had an estimate of the waveform for each shot, and an estimate of the receiver response for each receiver, we could apply *signature deconvolution* to create

a surface-consistent correction for these source and receiver characteristics, as well as the individual variations in the response.

While the great pains are taken to make sources, such as airguns or vibrators reproduceable and to make receivers that all have the same response, failures in reproucibility happen either because of unexpected behaviors of the instruments, or because of local conditions in the source and receiver environments.

For the Viking Graben data, we can exploit the fact that the each hydrophone is at a fixed distance to make these differences quantifiable.

11.13.1 Differing source strengths

We may study the source strength by looking at the rms power of each shot. We do this by windowing the data to a single offset. For example we might consider looking at the first arrival at a fixed offset, such as **offset**=-262 meters. Rather than look at the full trace, we might consider looking at the maximum of the RMS value of the first reflected arrival at -262 meters.

This would be done via:

```
$ suwind key=offset min=-262 max=-262 < shot_gathers.su
                                  > seis_offset_m262.su
$ suwind tmin=.48 tmax=.56 < seis_offset_m262.su |
         suxmax mode=rms label2="amplitude"
                label1="energy point number" x1beg=101
                  title="RMS amplitude comparisons at offset=-262 m" &
```

We first would want to remove source strength, by beginning with our data as shot gathers. Here this is the file **seismic.su** before any other processing. There is a program called **susplit** which will split the data out into separate files based on header field value To make separate files of shots, we first move the data into a convenient location

```
$ mkdir Temp
$ mv seismic.su Temp
$ cd Temp
$ susplit key=ep close=1 < seismic.su
```

There are 1001 shots, so there will be 1001 files that begin with the word **split**. We may loop over these, performing a gaining operation that balances the data by shot gather panel. The gaining of choice is to divide by the *RMS power* of the data, which is the square root of the sum of the squares of the seismic data values in all of the traces of a given shot gather. In a shell script we run

```
rm pbal.shot.su
for i in ‘ls split_* ‘

        sugain panel=1 pbal=1 < $i >> pbal.shot.su
```

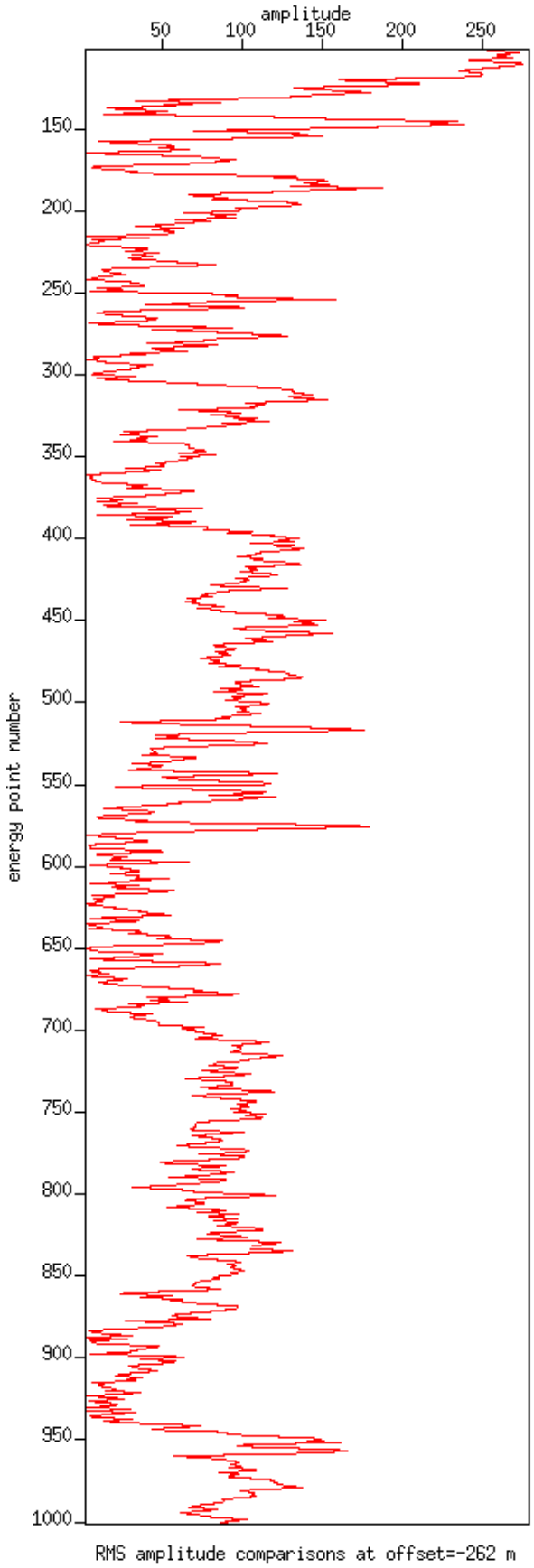


Figure 11.6: RMS power of the first reflected arrival at offset=-262m

```
done

rm split*
```

The double redirect out >> says "append values", so the file **pbal.shot.su** contains all of the power balanced shots. We are free remove the separate shot files after the process is complete via

```
rm split_*
```

This is certainly crude. We might improve this by applying sunormalize with the windowing option

11.13.2 Correcting for differing receiver gains

Similarly, we can take the resulting power balanced shot gathers and sort these into *receiver gathers* via

```
$ susort gx offset < pbal.shot.su |
          susplit key=gx close=1
```

The result now is a collection of files whose names begin in the word **split** each containing a single receiver gather. As before, we run in a shell script the operations

```
rm pbal.rec.su
for i in `ls split_* `
do
        sugain panel=1 pbal=1 < $i >> pbal.rec.su
done
rm split*
```

to yield the power balanced receiver gathers.

The final file **pbal.rec.su** of receiver gathers can be re-sorted into CDP gathers and gained, and the other processing we have discussed already can begin

```
$ susort < pbal.rec.su cdp offset | sugain jon=1 > gain.jon=1.cdp.su
```

where here, we recognize that "gain" also includes power balancing for shot strength and receiver gain. Again, we are free remove the separate shot files after the process is complete via

```
$ rm split_*
```

Both of these operations are captured in the shell script **Pbal** located in /data/cwpscratch/Data5/.

This technique is, at best, approximate, and may indeed not generate much improvement in the data. The best situation is if we had the source wavelet for each shot, and an actual receiver response for each receiver.

There are many attempts in the industry to perform *wavelet estimation* and to estimate receiver response. Such responses could be used to perform a *surface consistent deconvolution.*

11.14 Advanced deconvolution— Homomorphic Wavelet Estimation and signature decon

One method of wavelet extraction that we can try is called *homomorphic wavelet estimation*. The principle is simple. If the only thing that does not change in a collection of seismic traces is the wavelet, then the average of the Fourier representations of the traces would, by the law of large numbers, tend to the spectrum of the wavelet.

We can see why this is so by the following. If we consider the signal $s(t)$ to be the convolution of the wavelet $w(t)$ with the reflectivity series $r(t)$, further convolved with the multiple series $m(t)$, with each having its own noise, we have

$$s(t) = w(t) \star r(t) \star m(t). \tag{11.14.1}$$

In the frequency domain, the convolutions become multiplications

$$S(\omega) = W(\omega)R(\omega)M(\omega). \tag{11.14.2}$$

To make matters simpler, we would represent the data by the natural logarithm of the spectrum

$$\begin{aligned} \ln(S(\omega)) &= \ln\left(A_w(\omega)A_r(\omega)A_m(\omega)e^{i(\phi_w(\omega)+\phi_r(\omega)+\phi_m(\omega))}\right) \\ &= [\ln|A_w(\omega)| + \ln|A_r(\omega)| + \ln|A_m(\omega)|] \\ &\qquad +i\left[\phi_w(\omega) + \phi_r(\omega) + \phi_m(\omega)\right]. \end{aligned} \tag{11.14.3}$$

Thus the natural log of the amplitude spectra can be averaged and the phases can be averaged over a collection of traces, the result exponentiated, and the inverse transform performed.

The amplitude spectra should all be similar to the source spectrum, with a loss of higher frequencies due to anelastic attenuative loss. If the wavelet is the only thing that does not change very much trace by trace, then sum over these should tend to the log-amplitude spectrum and phase of the wavelet.

The result can be exponentiated and inverse Fourier transformed giving an estimate of the wavelet for each shot.

Phase unwrapping

There is an added complication in that phase that is calculated by taking the arctangent of of the ratio of the imaginary part over the real part of the Fourier transformed data must be "unwrapped." The arctangent function only returns the principal branch, which means that the arctangent function only returns phase angles between $-\Pi$ and π. There are several strategies for doing unwrapping the phase. The trend in the phase is also removed to aid in averaging the phase values.

The output wavelet is assumed to be minimum phase. As a last step we convert the output wavelet to its minimum phase equivalent. This is done in what is called the

cepstral domain, which is the inverse Fourier transform of the resulting log frequency domain representation. In SU the program *suminphase* performs this function.

The resulting wavelet may then be deconvolved from the data using **sucddecon**.

The shell script **Signature_Decon** located in **Data5** sorts the data into shot gathers, splits the shot gathers into separate files, performs homomorphic wavelet extraction and deconvolution with the extracted wavelet, and concatenates the result on **$outfile**.

The wavelet estimation seems to work best on the raw data or on the the muted raw data that has been gained.

```
#! /bin/sh

# estmiate source wavelet by shot and receiver response
# and deconvolve input data

set -x

infile=shot_gathers.su
outfile=shot_receiver_sigdecon_$infile

echo "Signature decon with homomorphic wavelet estimation "

## Assumptions:
## 1) wavelet is constant within a shot gather
## 2) reflectivity and multiple series are random
## 3) wavelet is minimum phase

## correcting for souce wavelet
# remove output files
rm shot_$outfile
rm $outfile
rm all_shot_wavelets.su
rm all_receiver_responses.su

# split the original data into shot gathers
# split shot data
susort  sx offset < $infile > shot_gathers.su
susplit < shot_gathers.su key=sx


# loop over shot gather files
for i in `ls split_sx* `
do
```

```
        # stack real part of the complex log transform (log|A|)
        suclogfft < $i  | suamp mode=real | sustack key=dt > real.su

        # stack imaginary part of the complex log transform (phase)
        suclogfft < $i  | suamp mode=imag | sustack key=dt > imag.su

        # combine real and imag and inverse clog transform
        # and output a minimum phase version of
        suop2 real.su imag.su op=zipper | suiclogfft  |
        suminphase | suwind itmax=199 > wavelet_est.su

        cat wavelet_est.su >> all_wavelets.su

        # deconvolve with sucddecon
        sucddecon sufile=wavelet_est.su < $i >> shot_$outfile


done

rm split_sx*

...
```

This shell script can be easily extended to the problem of removing the receiver response, by re-sorting the resulting output into common receiver gathers, and performing the same wavelet extraction and deconvolution, much as done in the **Pbal** script in the previous section.

The term *surface consistent deconvolution* is often used as a label for deconvolutional processes that correct for both source and receiver effects.

11.15 Muting NMO corrected data

The program **sumute** may be used to surgically remove undesirable noise on the CMP gathers that occurs for times early than the water-bottom reflection. Because our prospect has a roughly flat surface the time of the reflection of the water bottom is at approximately time .48 seconds. In addition to the noise before the water-bottom reflection, there are some unsuppressed multiples, or other arrivals on the far offsets that are undesirable.

We may use predictive deconvolution to clean up those near offset traces, or we may consider eliminating the near offset traces entirely with **suwind** before further processing. Putting these together, after NMO we may insert the commands

```
... | suwind key=offset min=-3237 max=-450 |
        sumute key=offset tmute=.45,.45 xmute=-3237,-450 | ...
```

prior to the stack to clean up the image. The choice of nearest offset to include is a matter of personal preference. The value of -450 is not necessarily the best value.

11.16 Ghost reflections

The waves that travel from the source to the water surface and then propagate down in the model, as well as the reflection that travel from the subsurface, to the water surface and to the receiver array interfere with the more direct reflections to produce what are called *ghosts*. The issue of ghosting, and *deghosting* can be complicated.

Because the delay between the primary reflections and the ghost reflection is short, the phenomenon reveals itself as a notch in the spectrum at the frequency where the peak from the ghost reflection cancels the trough from the primary. Thus, from the source and receiver depth, and the speed of sound in water, we can estimate this ghost notch as

$$f_{\text{notch}} = \frac{1}{2}\frac{V_{\text{water}}}{h_{\text{source or receiver}}} \tag{11.16.1}$$

where h is the depth to the source or receiver.

11.17 Surface related multiple elimination

A modern approach to multiple elimination is the Surface Related Multiple Elimination (SRME) method invented in 1991 by Erich Verschuur, then a Ph.d. student at Delft University of Technology. The method is a data driven annihilation method that makes assumptions about the structure of multiples based on an autoconvolution model of multiples. Verschuur began with the observation that multiples could be made by the convolution of a seismic trace with itself (suitably shifted) and reasoned that it should be possible to use the data itself to model the multiples, and then use adaptive subtraction to remove the multiples from the data.

11.17.1 The auto-convolution model of multiples

The SRME method operates on a very simple model of multiples. If we consider the **auto-convolution** of data with itself, then such resulting autoconvolved data are the same as multiples, assuming simple single layer reflection.

For example if we convolve the noise free version of the **fake.su** data with itself and add this to the noise free fake data

```
$ suconv < fake_no_noise.su sufile=fake.su panel=1 > autoconv.su
$ susum autoconv.su fake_no_noise.su > fake+autoconv.su
$ suxwigb < fake+autoconv.su title="autoconvolution multiples"
```

we see that the result is similar to what we would expect for data with multiples.

In theory, we could use the earliest arrivals on a seismic reflection profile to build a model of the first multiples through autoconvolution, and then adaptively subtract these out of the data. The demultipled portion of the data would then be autoconvolved to generate a model of the second order bounces, which in turn would be subtracted. The process of modeling followed by adaptive subtraction can then be repeated until the data are completely cleaned of multiples, as best as the algorithm could handle this.

Both 2D and 3D versions of SRME have been implemented.

Unfortunately, we do not yet have an SRME code in the SU.

11.18 Homework Assignment #8, Due Thursday 5 Nov, before 9:00am and Tuesday 3 Nov 2015

This exercise is similar to problem #7 however you will be applying more processing techniques.

- Start with the ungained and unmuted data.
- Mute and gain dataset.
- Apply one of the spectral methods to sharpen the waveform of the data.
- Use **suwind** to break the full dataset into blocks that are about 500 CDPs in size. The number of blocks is up to you. For example, the last block was more than 500 CDPs, so you could split that one up. Or maybe you want to use bigger blocks.
- Perform the Radon-domain multiple suppression on each of these smaller blocks, to do a better job of multiple suppression. It is more important to preserve the real reflections than it is to try to suppress all multiples, at the expense of data. You may have done all of this already in the previous assignment. Redo parts only if feel that you need to improve the result.
- Concatenate the blocks together to form the full multiple-suppressed version of the dataset. You can call this dataset
  radon.gain.yourgainparameters.cdp.su
- Repeat exercise #5 on this multiple-suppressed dataset (except do not show near and far offset stacks. Stack over all traces, and use your best stacking velocities.) Save your NMO velocities in a file called, say **radon_nmo_vel.par**. For example, if you used the CDPs at 250,750,1250,1750 to get your NMO velocities, then the contents of the file would look something like:

```
cdp=250,750,1250,1750
tnmo=0,.,..,..,..,..,..
vnmo=1500,..,..,..,..,..
```

```
tnmo=0,.,..,..,..
vnmo=1500,..,..,..
tnmo=0,.,..,..,..,..
vnmo=1500,..,..,..,..
tnmo=0,.,..,..,..,..,..
vnmo=1500,..,..,..,..,..
```

where the **tnmo= vnmo=** are those that you got for the respective CDPs. Note that the number of **tnmo=** and **vnmo=** values per pair have to be the same, but each pair may have a different number of values from other pairs.

The idea is to concatenate and then NMO-correct the data.

Use this **radon_nmo_vel.par** for performing the NMO correction before stacking.

```
$ sunmo par=radon_nmo_vel.par < radon.gain.yourgainparameters.cdp.su
                        | sustack > stack.su
```

Obviously, you cannot complete this assignment on time if you do not start working on this immediately. Ideally, everyone will have a Radon multiple-suppressed version of the full dataset
radon.gain.yourgainparameters.cdp.su ready for class on the due date so we can proceed with fancier velocity analysis.

Other tips:

- Feel free to use **at** to run the jobs at night.
- Furthermore, you might consider regaining the data after you have done multiple suppression.
- Who says that you need to stack all of your data? It may be that the far offsets and the nearest near offsets could be omitted from your data, and make the dataset a bit smaller. However, is it worth the loss of data? You decide.

11.18.1 How are we doing on multiple suppression and NMO Stack?

The subset approach that is pursued in the Homework 7 and 8 suffers from a serious flaw. While we have a set of stacking velocities for each block, we are not taking advantage of the ability of our programs to *interpolate* these values across the section. We may see a blocky appearance.

While performing this procedure in subsets makes it a bit quicker, this is for *instructional purposes only.* **From now on, we work with the full dataset. We do not break the data into blocks.**

11.19 Concluding Remarks

Much of exploration seismic research conducted prior to the mid 1980s was focused on the problem of seismic deconvolution and wavelet estimation. The CWP/SU:Seismic Unix package was largely developed during a time right after this, when exploration seismic research was focused on amplitude preserving depth migration, so consequently we have a lot of migration related tools and a comparatively few deconvolution-related programs, so deconvolutional methods are not yet well represented in the SU package.

Though we have not really done justice here to the broad topic of deconvolution and the other spectral methods, we can see that the application of these techniques is more involved than merely applying the operation. Considerable preconditioning of the data is required to make the traces look more alike, so that the deconvolutional process may remove the parts (such as multiples) that we don't want.

Predictive decon really means that we use the first part of the data to predict the repetitions in the latter part of the data, and to use those predictions to annihilate those repetitions (multiples). For this to work the repetitions must closely match the initial waveforms. Hence, making the amplitudes as uniform as possible is desireable for such techniques to be applied.

In the modern world there is an increasing demand for amplitude information for the extraction of amplitude versus angle (AVA) also known as amplitude versus offset (AVO) information. Balancing away all of the amplitude variability in the data is not desirable, so methods that preserve amplitudes and are data driven are preferred.

Chapter 12

Velocity Analysis on more CDP gathers and Dip Move-Out

A shell script called **Velan.radon** is supplied in /data/cwpscratch/Data5. This is a general purpose shell script for velocity analysis. The script **Velan.radon** is designed to aid in the generation of velocities for Radon transform based mulitiple suppression, but the NMO velocities picked with this script may also be used for stacking.

These shell script makes use of a number of programs that you have used already, including **suvelan**, **suwind**, **suximage**, **suxgraph**, **sunmo**, **suradon**, and **suxwigb** as well as some programs you have not used, such as **unisam2**, **velconv**, and **smooth2**.

The begining of this script gives you an idea of how to run it, and what parameters you may adjust:

```
#! /bin/sh
# Velocity analyses for the cmp gathers
# Authors: Dave Hale, Jack K. Cohen, with modifications by John Stockwell
# NOTE: Comment lines preceeding user input start with  ##
#set -x

## Set parameters
datadir=.

velpanel=gained_data_sorted_in_cdps.su  # gained data sorted in cdps
vpicks=radon_nmo_vel.par       # output file of vnmo= and tnmo= values
normpow=0               # see selfdoc for suvelan
slowness=0              # see selfdoc for suvelan
cdpfirst=1              # minimum cdp value in data
cdplast=2142            # maximum cdp value in data
cdpmin=132              # minimum cdp value used in velocity analysis
cdpmax=2110             # maximum cdp value used in velocity analysis
dcdp=512                # change in cdp for velocity scans
```

```
fold=120                # maximum number of traces per cdp gather
dxcdp=12.5              # distance between successive midpoints
                        # in full data set
mix=0 # number of adjacent cdp panels on either side
# of a given panel to mix

## Set velocity sampling and band pass filters
nv=150 # number of velocities in scan
dv=15 # velocity sampling interval in scan
fv=1450.0 # first velocity in scan
nout=1500 # ns in data

## Set interpolation type for velocity function plots
interpolation=mono # choices are linear, spline, akima, mono

## set filter values for wiggle trace plots
f=1,5,70,80 # bandwidth of data to pass
amps=0,1,1,0 # don't change

## suximage information
wclip=0 # This number should be between 0 to .15 for real data
bclip=.3 # this number should be between .2 and .5
cmap=hsv2 # colormap
perc=97 # clip above perc percential in amplitude
xcur=1 # allow xcur trace xcursion in wiggle trace plots
curvecolor=black # color of stacking velocity picks curve


#average velocity
vaverage=2100        # this may be adjusted

# radon transform parameters
dp=8 # increment in p in radon transform
pmin=-2000 # minimum value of p in radon transform
pmax=2000 # maximum value of p in radon transform
pmula=20 # t=tmax intercept of radon filter
pmulb=200 # t=0 intercept of radon filter
offref=-3237 # offset at maximum moveout
interoff=-262 # offset at minimum moveout

## unisam parameters
# sloth parameter: Interpolate as sloth=0 velocities, sloth=1 slownesses,
# sloth=3 sloths
```

```
sloth=1

# smoothing
smooth=1
r=3.5 # values should probably not exceed 5 or 6

# preprocess with predictive decon? decon=0 no , decon=1 yes
# minlag= width of autocorellation waveform, maxlag= repetition time + minlag
decon=1
minlag=.2
maxlag=.7

######## You shouldn't have to change anything below this line ###########
```

The idea of the script is to take the full dataset, window it into specific CDPs, assuming an increment in CDP, allow the user to pick a semblance panel and view the resulting NMO corrected version of the data panel. If the velocity picks are to the liking of the user, then the script proceeds to the next CDP panel. The end product is a collection of **tnmo=** and **vnmo=** values, in a file called **nmo_vel.par** for **Velan** and **radon_nmo_vel.par** for **Velan.radon**. The distinction is made, because the velocities used for Radon transform based multiple suppression may not be quite the same as those used for stacking.

If we go in 512 cdp increments across the data, we will sample the velocity approximately 4 times (depending on the values of **cdpmin=** and and **cdpmax=**) across the section. This will give a representative collection of velocities that will be more repesentative of the actual velocity change across the profile. The **Velan** script runs a number of SU programs to make an estimate of the velocity profile in time, and when you are finished doing velocity analysis, will generate uniformly sampled velocity profiles of both RMS and interval velocities. These are "quick and dirty" representations of the velocity field, which tend to be bad, because they have spurious errors due to errors in interpolation and in conversion from RMS to interval velocities. These can be used as a guide for the construction of a background velocity model.

This procedure may be repeated, changing, for example, the value of the cdp increment **dcdp=256**, **dcdp=128**, or **dcdp=64** to obtain increasingly dense coverage in RMS velocities. The idea is to go in increments of powers of 2 so that you only have to do every other CDP. Or, the user may start with a different value than 512 and bisect that.

Output from Velan.radon

If you look carefully in **Velan.radon** you will see that there are a number of output files. These include the (tnmo,vnmo) time and velocity pairs that is the result of semblance picking. However along the way there are a number of generated files that are useful for both diagnostics and for further processing. These are:

```
...

vpicks=radon_nmo_vel.par       # output file of vnmo= and tnmo= values

...

# binary files output
vrmst=vrmst.bin # VRMS(t) interpolated rms velocities
vintt=vintt.bin # VINT(t,x) as picked
vinttav=vinttav.bin # average  VINT(t) of VINT(t,x)
vrmstav=vrmstav.bin # average  VRMS(t) of VRMS(t,x)
vinttuni=vinttuni.bin # interploated Vint(t,x)
vintzx=vintzx.bin # VINT(z,x)interpolated interval velocities
vintzav=vintzav.bin # average  VINT(z) of VINT(z,x)
vintxz=vintxz.bin # VINT(x,z)interpolated interval velocities
```

We may view the resulting velocity files by running the shell script **Xvelocity**, which will show RMS, average RMS, interpolated RMS, as well as average interval velocities. These automatically generated velocity files tend to be lumpy, so they are not really suitable to be used for migration as is, but may provide important information for constructing velocity models later on.

The file **radon_nmo_vel.par** has contents that are similar to

```
cdp=128,192,256,320,384,448,512,576,640,704,768,832,896,960,1024,1088
tnmo=0.0,0.636979,1.15987,1.31199,1.91094,2.9187
vnmo=1500,1557.09,1734.03,1822.5,2041.34,2488.34
tnmo=0.0,0.579936,0.893672,1.35002,1.8539,2.41482,2.7951
vnmo=1500,1561.75,1659.53,1831.81,2032.03,2446.44,2693.22
tnmo=0.0,0.598951,0.922194,1.35952,1.50213,1.89192,2.90919
vnmo=1500,1557.09,1678.16,1794.56,1855.09,2027.38,2562.84
tnmo=0.0,0.684515,1.29297,1.87291,2.3958
vnmo=1496.56,1571.06,1734.03,1971.5,2404.53
tnmo=0.0,0.751065,0.998251,1.18839,1.40706,1.81587,2.47186
vnmo=1500,1608.31,1720.06,1808.53,1859.75,2008.75,2399.88
tnmo=0.0,0.713037,0.988744,1.32149,1.65424,1.90143,2.89017
vnmo=1500.25,1626.94,1696.78,1817.84,1887.69,2050.66,2623.38
tnmo=0.0,0.665501,1.01727,1.38804,1.70178,2.02502,2.93771
vnmo=1500,1594.34,1724.72,1850.44,1915.62,2157.75,2576.81
tnmo=0.0,0.636979,1.04579,1.41657,1.68277,1.87291,2.32925,3.53666
vnmo=1500,1575.72,1771.28,1817.84,1887.69,2069.28,2278.81,2777.03
tnmo=0.0,0.465851,0.789094,1.09332,1.4641,1.90143,2.43383
vnmo=1500,1538.47,1631.59,1780.59,1850.44,2022.72,2330.03
```

```
tnmo=0.0,0.675008,0.865151,1.20741,1.48312,1.92045,2.5289,3.13736
vnmo=1500,1585.03,1710.75,1771.28,1831.81,2041.34,2460.41,2823.59
tnmo=0.0,0.779587,1.14086,1.92045,2.2532,2.63348,3.3275
vnmo=1500,1640.91,1743.34,2027.38,2222.94,2371.94,2860.84
tnmo=0.0,0.646486,0.846137,1.15037,1.51164,1.82537,2.49087,2.98525
vnmo=1500,1575.72,1687.47,1817.84,1938.91,1980.81,2446.44,2735.12
tnmo=0.0,0.655994,0.874658,1.17889,1.57819,1.89192,2.47186
vnmo=1500,1575.72,1715.41,1817.84,1943.56,2008.75,2520.94
tnmo=0.0,0.598951,0.874658,1.17889,1.55917,1.9965,2.50038,3.80286
vnmo=1500,1585.03,1710.75,1827.16,1948.22,2078.59,2395.22,3135.56
tnmo=0.0,0.684515,0.931701,1.25494,1.56868,1.9965,2.1296,2.63348
vnmo=1500,1580.38,1738.69,1827.16,1920.28,2064.62,2153.09,2413.84
tnmo=0.0,0.789094,1.26445,1.64474,2.2532,3.00426
vnmo=1500,1654.88,1836.47,1971.5,2199.66,2683.91
```

Here, some hand editing has been applied.

12.0.1 Applying migration

We then can apply any of the post stack migration algorithms that we have discussed earlier in the text.

Any of these corresponding **tnmo=..,..,.. vnmo=..,..,...** pair sets may be copied from **radon_nmo_vel.par** a new file, say **stolt.par** and used as the "par" file as input for **sustolt**. Select only one **tnmo=..,..,.. vnmo=..,..,..**, change **tnmo=** and **vnmo=** to **tmig=** and **vmig=**, and you are ready to run **sustolt**.

For example

```
$ sustolt par=stoltmig.par cdpmin=1 cdpmax=2142 dxcdp=12.5
                          < stack.nmo.radon.gain.jon=1.su
                               > stolt.stack.nmo.radon.gain.jon=1.su
```

The file **vintt.bin** is an approximate interval velocity as a function of time $v_{int}(t)$, and thus may be used as the **vfile** in **sugazmig**, **sumigps**, or **suttoz**. For example

```
$ sugazmig < stack.nmo.radon.gain.jon=1.su vfile=vintt.bin dx=12.5
                          > gazmig.stack.nmo.radon.gain.jon=1.su
$ sumigps < stack.nmo.radon.gain.jon=1.su vfile=vintt.bin dx=12.5
                          > migps.stack.nmo.radon.gain.jon=1.su
$ suttoz vfile=vintt.bin nz=1500 < stolt.stack.nmo.radon.gain.jon=1.su
                           > depth.stolt.stack.nmo.radon.gain.jon=1.su
```

Warning! At best these "automated" velocities are for testing purposes only! Similarly, there is file **vintzav.bin** that may be used with the depth migration programs. Again, there tend to be systematic errors between stacking derived interval velocities and true interval velocities, so these should be used for "quick looks" only.

12.0.2 Homework #9 - Velocity analysis for stack, Due Thurs 12 Nov 2015, before 9:00am and Tuesday 10 November 2015. (This assignment is paired with Homework #10 in the next chapter, so be aware of this.)

- Perform velocity analysis on the multiple-suppressed data using **Velan.radon**. These stacking velocities, with corresponding cdp values will be in a file called **radon_nmo_vel.par**. Note that if you have good stacking velocities from previous assignments, please feel free to include this in your **radon_nmo_vel.par** file. Feel free also to hand edit these velocities as you see fit.

- Apply NMO and Stack, using

```
$ sunmo par=radon_nmo_vel.par < radon.gain.jon=1.cdp.su |
     sustack > stack.nmo.radon.gain.jon=1.su
```

  Again, the file names you use should be the names of your corresponding files.

- Discuss the degree of improvement over the image quality given these better NMO velocities.

- Now perform a Stolt migration of your multiple-suppressed, NMO-corrected, and stacked data. Because Stolt migration uses RMS, which is to say NMO (stacking) velocities, take one of the **tnmo=** and **vnmo=** pairs from your **radon_nmo_vel.par** file as your **tmig=** and **vmig=** values. Or make up what you view as a representative or average set of **tmig=** and **vmig=** values for **sustolt**.

- Use **suintvel** to convert your RMS velocities into *interval* velocities. Using **dz=3 nz=1500** with these velocities, use **suttoz** to make an approximate depth section out of your Stolt-migrated image. To do this you may use the shell script **RMStoINT** located in /data/cwpscratch/Data5. The resulting **suttoz.par** file may be used for this. For example

```
$ suttoz par=suttoz.par dz=3 nz=1500 < stolt.su > stolt.depth.su
```

  (We performed some of these operations in the early part of the semester. You may consult the earlier chapters of the notes for details. Note, also that there are shell script examples in Data5.)

- Show your Stolt-time section and your Stolt-depth section and list your velocity-time pairs. Don't be afraid to use small clip= values to accentuate weaker arrivals that you see in the section.

- Discuss what you see.

Additional tips

For students who wish to redo their multiple-suppression, please make use of the shell script **Velan.radon** located in /data/cwpscratch/Data5. This shell script is a version of the Velan script set up to allow you to make picks explicitly for multiple suppression. You may also want to apply predictive deconvolution to suppress the near-offset multiples before applying **Velan.radon**, and **Radon.final**.

The **dz=3** and **nz=1500** in **suttoz** is the source of the 4500 you see for the maximum depth on your depth section.. With **d1=3** in the headers on the output to **suximage** we are showing the data to a depth of 4500 meters.

The question for the processor is how much of the output to show.

You may try different values of **dz=** and **nz=** to see what happens to your depth-stretched image.

12.1 Other velocity files

There are several velocity files that are generated by the Velan script. These velocity files are provided only as a guide for later velocity model building.

The file **vintzx.bin** is a very approximate $v_{int}(z, x)$, but likely contains irregularities that would not make this the best velocity file to use for migration, though it is in the correct format to be used as the input **vfile** in **sumiggbzo**, or for the input velocity file for **rayt2d**, which generates the traveltime tables for **sukdmig2d**. Similarly, while the file **vintxz.bin** is in the correct format to be used as the input for **sumigfd**, **sumigffd**, **sumigsplit**, or **sumigpspi**, it too will likely be too "lumpy" to give a good result.

We *can* however use these estimated velocities for some interval velocity information when we build velocity models by horizon picking.

12.1.1 Velocity analysis with constant velocity (CV) stacks

We have used a method of semblance picking to estimate the stacking velocities. This is an estimate, but it is not the only way to get the stacking velocities. An alternate method is the *constant velocity stack* (CVS).

In /data/cwpscratch/Data5 are two shell scriptsp **MakeCVSstackMovie**

```
#! /bin/sh

data=multiple_suppressed_gained_data.su
movie=stackmovie.su

fvel=1500
dvel=10
vmax=3000
```

```
vel=$fvel

rm $movie

while [ "$vel" -lt $vmax ]
do

echo $vel
sunmo tnmo=0.0 vnmo=$vel < $data |
sustack >> $movie

vel=‘expr $vel + $dvel‘


done



suxmovie < $movie perc=98 n1=1500 n2=2142 loop=1 sleep=5 title="$g" fframe=$fvel dframe

exit 0
```

and **ViewStackMovie**

```
#! /bin/sh

movie=stackmovie.su

fvel=1500
dvel=10

suxmovie < $movie perc=99 d1=.004 n1=1500 n2=2142 sleep=10000 loop=1 title="vnmo=$fvel

exit 0
```

The idea of CVS is to perform constant velocity NMO on the data and stack it, repeating for a large range of velocities. Here, this idea is implemented by sweeping through stacking velocities starting with **vnmo=1500** in increments of 10 m/s. The data are NMO corrected and then stacked. Each stacked section becomes a frame in a movie. In the title line of the movie, the stacking velocity is expressed as **vnmo=1500 + devel*frame number**.

It takes about 20 minutes on a modern multi-core PC to generate the **stackmovie.su**.

12.2 Dip Moveout (DMO)

Dip moveout is a partial migration that will convert NMO corrected prestack data to more closely approximate true zero offset data. In the original formulation of dip moveout by Dave Hale, the operation is applied to common offset data that have been NMO corrected. The program **sudmofk** can be used to perform dip moveout on our NMO corrected data.

Hale's method was not designed to be amplitude preserving. More modern applications of this type of data transformation that preserve amplitude have been developed. Two of these general transformations are called *transformation to zero offset* TZO or *migration to zero offset* MZO. As the names suggest data are tranformed through a migration-like operation to synthetic zero-offset data. The motivation for developing such operations follow from the computation cost of doing full prestack migrations. Alternately, these techniques can be applied as velocity analysis techniques. Indeed, NMO or NMO followed by DMO are really first approaches to transformation to zero offset.

12.2.1 Implementing DMO

Note, you need storage space for one or more copies of the full dataset if you want to try this. For example, we may NMO correct the data via

```
$ sunmo par=nmo\_vel.par < radon.gain.jon=1.cdp.su > nmo.radon.gain.jon=1.cdp.su
$ susort offset gx  < nmo.radon.gain.jon=1.cdp.su > nmo.radon.gain.jon=1.co.su
```

where it is important to note that it is not a good idea to try to use pipes with **susort**. The **.co.** in the extension name indicates that these data are now in common offset gathers. We may then perform dip moveout processing with **sudmofk**. This program requires an average set of **tdmo=..,..,..** **vdmo=..,..,..** pairs, which may are an average set of times and velocities taken from **nmo_vel.par**, and which are copied into a file named, say, **dmofk.par**

```
$ sudmofk par=dmofk.par  cdpmin=1 cdpmax=2142 dxcdp=12.5  noffmix=7
         < nmo.radon.gain.jon=1.co.su
              > dmofk.nmo.radon.gain.jon=1.co.su
```

Here we **noffmix=7** is chosen for this example. You may need to experiment with this parameter on your data. The entries in **dmofk.par** must be named **tdmo=** and **vdmo=** for **sudmofk** to be able to see them. The DMO process may take some time to complete time to complete.

After the process has finished, you will need to resort your data back into CMP gathers via:

```
susort cdp offset <  dmofk.nmo.radon.gain.jon=1.co.su
                       >   dmofk.nmo.radon.gain.jon=1.cdp.su
```

These new data may then be stacked and migrated. Again, you must have sufficient storage capacity to save a couple of copies of the full dataset.

We expect improvement of the stack with DMO only in areas where dips are sufficiently large that the NMO approximation fails to completely flatten the data. If the input data are reflections over a generally flat, or low dip geology, then we don't expect to gain much by performing this operation.

12.3 Concluding Remarks

The notion of NMO followed by DMO as a way of building equivalent zero-offset datasets naturally let to the notion of "transformation to zero offset (TZO)" and "migration to zero offset (MZO)". The basic notion of TZO and MZO is the following.

Suppose that you could do a perfect amplitude-preserving prestack depth migration of data with offset. Then the result would be a representation of the earth—a geologic model with bandlimited delta functions describing the reflectors. Then suppose we did a remodeling from that representation to yield zero-offset data. The cascade of these processes would be a "migration followed by remodeling" to the desired zero-offset traces. In fact, you could remodel to any type of data.

But if you could do this, why bother? With all of those integrals it would be horribly expensive in computer time? The answer is that a number of the integrals in the cascade of migration-remodeling can be done approximately, using asymptotic methods, so that the resulting algorithm looks like a migration algorithm, but migrates the data not to the earth model, but to approximate zero-offset data. These data could then be stacked and then migrated with a conventional migration program.

One motivation for doing this is to create a more advanced type of velocity analysis. The usual NMO—DMO—STACK procedure is only approximate for real data. The TZO methods work better, though they are more expensive. In fact, there are a number of forms of "migration velocity analysis" that make use of the ideas stated here. The other motivation was to apply DMO as a substitute for full 3D depth migration, back when computer speed and storage was more limited.

Chapter 13

Velocity models and horizon picking

One seldom uses velocity models derived directly from velocity analysis, though this is possible. What is more common is to use a preliminary depth-section seismic image combined with an estimate of interval velocities to construct a background wavespeed profile that is used for either poststack or prestack depth migration.

For example, we may perform Stolt migration and convert this to a depth section with **suttoz** to obtain first approximation of the migrated image

```
$ sustolt par=stolt.par cdpmin=1 cdpmax=2142 dxcdp=12.5
                          < stack.nmo.radon.gain.jon=1.su
                               > stolt.seis.su
$ suttoz par=suttoz.par nz=1500 < stolt.seis.su > stolt.depth.seis.su
```

(Note. Do not call this a "depth migration". It is a **depth section** derived from a time migration.)

The file **suttoz.par** may be made by running shell script **RMStoINT**, which employs **suintvel** to do the interval velocity conversion. This script is provided in /data/cwpscratch/Data5. The shell script produces as its output the file **suttoz.par**. The contents of this simple shell script are

```
#! /bin/sh

# convert a single RMS velocity to Interval velocity.

# put your values of tnmo= and vnmo=  here.
tnmo=
vnmo=

#output file
outfile=suttoz.par

suintvel t0=$tnmo vs=$vnmo  outpar=$outfile
echo t=$tnmo >> $outfile
```

```
echo "use v= and t=  values for suttoz "
echo " result output in $outfile"

exit 0
```

The entries for **vnmo=** and **tnmo=** are the same values that you used for **vmig=** and **tmig=** in **stolt.par**.

13.1 Horizon picking and smooth model building

From the contour map of **vintzx.bin** that we view with **Xvelocity**, one of the other velocity plots that is output by Velan, or from independent well log information we may be able to get an idea of a reasonable velocity trend that will be appropriate for migration. If you know the stratigraphy of an area, reasonable velocities estimates for given rock types provide this type of information. Again, simple is better than complicated, and fewer depth horizons (no more than 4) are better than more.

We then may use the script **Horizon** to pick depth horizons on the depth-migrated image. The shell script uses a similar blind picking technique as is used in **Velan** and prompts the user for velocities, as well as the space rate of change of velocity (these should be numbers on the order of **dvdz=.1** and **dvdx=.001** or **dvdx=-.001** where the units are (velocity units per meter).

The **Horizon** script is interactive and produces several output files. Two of these are **unif2.ascii** and **unif2.par**. These files is used by the shell script **Unif2.sh** to build a smoothed velocity file. The files produced by **Horizon** may be hand edited and **Unif2** run again to produce an updated set of velocity files.

The contents of **Unif2.sh** are

```
#! /bin/sh

#set -x

# set parameters here
method=mono                     # interpolation method
nz=1500
nx=2136

pickedvalues=junk1.picks          # this is the output of Horizon

cp junk1.picks temp.ascii


smoothint2 < temp.ascii r1=100 ninf=$ninf > unif2.ascii
```

```
unif2 < unif2.ascii par=unif2.par |
 smooth2 n1=$nz n2=$nx r1=50 r2=50 > junk.bin

cp junk.bin unewvelzx.bin

transp n1=$nz < unewvelzx.bin > unewvelxz.bin

exit 0
```

The value of **nx=** is the number of CDPs in the stacked data. The user may set the values of **r1=** and **r2=** to higher or lower values to control the level of smoothing.

The output velocity files are **unewvelzx.bin** and **unewvelxz.bin**, which may then be used for migration. time

13.2 Migration velocity tests

Likely you will not get the correct migration velocities on the first iteration of the velocity modeling process. The shell script **Gbmig**

```
#! /bin/sh
# Gaussian Beam

#set -x

# use a  v(z,x) velocity profile here
vfile=unewvelzx.bin

# vary the values of dz and nz as a test before running the full model
sumiggbzo < stack.dmo.nmo.myradon.gain.jon=1.su dx=12.5 dz=3 nz=1500 \
verbose=1 vfile=$vfile  > gb.seismic.su
```

applies Gaussian-beam migration to the stacked data using the velocity file **unewvelzx.bin**. This migration result may be viewed with **suximage** via

```
suximage < gb.seismic.su perc=99 legend=1 d1=3 d2=12.5  &
```

and the errors in migration velocity may be seen as either "frowns," indicating under-migration or "smiles" indicating over-migration of the data. The user may hand edit the file **unif2.par**, increasing or decreasing the velocities in response to the respective smiles or frowns. The user then runs **Unif2** again, and re-runs the migration. This process is repeated until the image has neither smiles nor frowns.

13.2.1 Homework #10 - Build a velocity model and perform Gaussian Beam Migration, Due 12 Nov 2015 for both sections.

Use the methods outlined in this chapter to build a velocity model and perform Gaussian beam migration using that model.

- Use the **RMStoINT** and **Suttoz** shell scripts to stretch the Stolt migrated version of your data from Homework #9 into a depth section.
- Use this depth section as input to the **Horizon** script and build a velocity model by picking several horizons in the data. Assign interval velocities based on the average interval velocities that are shown when you run **Xvelocity**
- Run the **Unif2.sh** script to generate the smoothed velocity files **unewvelxz.bin** and **unewvelzx.bin**.
- Use the **unewvelzx.bin** file and your stacked data in the script **Gbmig** to perform Gaussian beam migration on these data.

13.3 Concluding remarks

Seismic data and any other auxilliary data are combined in a step that involves the intelligence of the processor to build better background velocity profiles for further migrations, and for modeling.

We should not expect that this process can be made totally automatic. Indeed, some experimentation will show that it is quite easy for iterations of picking and migrating to yield a diverging series of modifications, that would ultimately result in a terrible and unrealistic velocity profile.

In the modern world new techniques such as **full waveform inversion** provide an approach that yields estimates of velocities. The preferred method of migration is **reverse time migration** (RTM), which is a prestack depth migration method.

Chapter 14

Prestack Migration

Owing to advances in computer technology it is now possible to perform *prestack* migration. The motivation for doing this is that information regarding the angular dependence of the reflectivity is preserved in prestack methods, but is not preserved in poststack migration. Having such information can give clues about the material properties at depth that are helpful in determining porosity and permeability as an aid in reservior characterization.

Furthermore, prestack migration may be used as a step in *migration velocity analysis*. If the output is in the form of *image gathers* which is to say migrated CMP gathers, then analysis of the curvature of the arrivals in the image gathers can be used as a guide for updating the wavespeed profile.

Finally, prestack migration is preferable if there is large vertical relief in subsurface structures. Data which are largely flat do not benefit from prestack migration.

The cost of prestack migrating a single CMP gather is comparable to migrating the entire poststack profile, so the computer costs increase correspondingly.

14.1 Prestack Stolt migration

The program **sustolt** may be used to perform prestack time migration by first sorting the multiple-suppressed, gained and NMO corrected data into *common-offset gathers* and then applying

```
$ sustolt par=stolt.par cdpmin=1 cdpmax=2142 dxcdp=12.5
          <  nmo.radon.gain.jon=1.co.su > prestack.stolt.nmo.radon.gain.jon=1.su
```

This takes about an hour to complete on a relatively fast PC. The result can be sorted into **image gathers**, which is to say "migrated CMP gathers." The motivation to do such a thing may be velocity analysis, but unfortunately, using **sustolt** we can only have a $v_{rms}(t)$ profile.

14.2 Prestack Kirchoff time migration

It is possible to perform Kirchhoff time migration using an algorithm based on the so-called *double square root operator* expressed as the program **suktmig2d**

```
 SUKTMIG2D - prestack time migration of a common-offset
section with the double-square root (DSR) operator


   suktmig2d < infile vfile= [parameters]  > outfile

 Required Parameters:
 vfile= rms velocity file (units/s) v(t,x) as a function
of time
 dx= distance (units) between consecutive traces

 Optional parameters:
 fcdpdata=tr.cdp first cdp in data
 firstcdp=fcdpdata first cdp number in velocity file
 lastcdp=from header last cdp number in velocity file
 dcdp=from header number of cdps between consecutive traces
 angmax=40 maximum aperture angle for migration (degrees)
 hoffset=.5*tr.offset half offset (m)
 nfc=16 number of Fourier-coefficients to approximate
low-pass
filters. The larger nfc the narrower the filter
--More--
```

The shell script **Suktmig2d** is provided to aid in the implementation of this code.

```
#! /bin/sh

#
vfile=vel_nmo.bin      # rms velocity as a function of cdp and time
dx=25                  # spacing between receivers
indata=radon_mute_filtered_repaired_co.su
outdata=ktmig.su

rm ktmig.su

# split the data into a bunch of common offset gaters
susplit < $indata key=offset
```

```
# loop over shot gather files
for i in ‘ls split_* ‘
do
suktmig2d vfile=$vfile dx=$dx  < $i >> $outdata
done

## clean up
# remove shot split files
rm split*
```

The input data, not NMO corrected, are sorted into shot gathers, with missing gathers replaced by the method discussed in Chapter 11. The input velocity file consists of a smooth model in RMS velocities, as would be produced by **sunmo** with the **voutfile=** option set.

The shell uses **susplit** to break the data into seprate shot files, which are then migrated individually. The resulting migrated shots are contcatenated to the file **ktmig.su**. This file may be sorted into CDP gathers. These migrated CDP gathers are called *image gathers*.

This takes about 7 hour to run, so this is practical for project assignments.

14.3 Prestack Depth Migration

It is common for prestack depth migration algoritms to be written with the assumption that the data are **not** gained. Prestack depth migration is a wave-equation based process, so the wave equation automatically takes care of the effect of geometrical spreading. The data also need to be in the form of shot gathers.

To undo geometrical spreading and resort the data into shot gathers

```
$ susort sx offset < myradon.gain.jon=1.cdp.su > junk1.su
           sugain tpow=-1  < junk1.su > myradon.shot.su
```

Or better yet, perform radon multiple suppression on the ungained, but muted data.

14.3.1 Pre-stack Kirchhoff Depth migration

The program **sukdmig2d** may be used to perform prestack depth migration. The program **rayt2d** is used to construct a traveltime table for the Kirchhoff migration algorithm. The shell script **Rayt2d.large** is provided for this purpose.

```
#! /bin/sh

rayt2d vfile=unewvelzx.bin \
dt=.004  \
nt=1500  \
fz=0 nz=1500 dz=3 \
fx=0 nx=2142 dx=25 ek=0 \
fa=-80 na=80 \
nxs=1012 fxs=3237 dxs=25 ms=1 \
tfile=tfile.unewvelzx

exit 0

\end{verbatime}

The script {\em Sukdmig2d\/} is provided
to run the acut

\begin{verbatim}

#! /bin/sh

rayt2d vfile=unewvelzx.bin \
dt=.004  \
nt=1500  \
fz=0 nz=1500 dz=3 \
fx=0 nx=2142 dx=25 ek=0 \
fa=-80 na=80 \
nxs=1012 fxs=3237 dxs=25 ms=1 \
tfile=tfile.unewvelzx

exit 0
```

The program requires a lot of ram. It will not run on all systems.

14.3.2 Pre-stack Fourier Finite difference depth migration

Given a good background velocity profile **unewvelxz.bin** the script **Prestackffd**.

```
#! /bin/sh

nxo=2142
nxshot=1012
```

```
nz=1500
dz=3
dx=12.5
vfile=unewvelxz.bin
fmax=90

data=radon_mute_filtered_repaired_shot.su

susort sx offset <  $data > data.shot.su

sumigpreffd < data.shot.su nxo=$nxo nxshot=$nxshot nz=$nz dz=$dz dx=$dx fmax=$fmax vfil


exit 0
```

This script can be modified and adapted to run **sumigprefd**, **sumigprepspi**, and **sumigpresp**.

Each take several days to a week to run on the Viking Graben data.